14th Brainstorming Week on Membrane Computing

Sevilla, February 1 – 5, 2016

Carmen Graciani
David Orellana-Martín
Agustín Riscos-Núñez
Álvaro Romero-Jiménez
Luis Valencia-Cabrera
Editors

14th Brainstorming Week on Membrane Computing

Sevilla, February 1 – 5, 2016

Carmen Graciani
David Orellana-Martín
Agustín Riscos-Núñez
Álvaro Romero-Jiménez
Luis Valencia-Cabrera
Editors

RGNC REPORT 1/2016

Research Group on Natural Computing
Universidad de Sevilla

Fénix Editora, Sevilla, 2016

ISBN: ??????
Depósito Legal: SE-????–06
Edita: Fénix Editora
Avda. de Cádiz, 7 – 1C
41004 Sevilla
fenixeditora@telefonica.net
Telf. 954 41 29 91

Preface

The 14th Brainstorming Week on Membrane Computing (BWMC) was held in Sevilla, from February 1 to February 5, 2016, in the organization of the Research Group on Natural Computing (RGNC) from the Department of Computer Science and Artificial Intelligence of Universidad de Sevilla. The first edition of BWMC was organized at the beginning of February 2003 in Rovira i Virgili University, Tarragona, and all the next editions took place in Sevilla at the beginning of February, each year.

In the style of previous meetings in this series, the 14th 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 40 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 2016 edition of the Brainstorming enjoyed a special visit from a group of undergraduate Physics students, under the initiative of Prof. Ricardo Graciani-Díaz, from the University of Barcelona. Their presence and interest was a powerful catalyst for many interactions, which have given rise to a couple of papers in this volume. In addition, each student wrote a short note on his/her experience.

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

A selection of papers from this volume will be considered for publication in special issues of *Theoretical Computer Science*, which will also contain a selection of papers from ACMC 2016 (the Asian Conference on Membrane Computing, to be held at Universiti Kebangsaan, in November 14th-16th).

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* – 2016.

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

***

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

1. María Arazo-Sánchez, University of Barcelona (Spain)
   `maria.arazo@gmail.com`
2. Fernando Arroyo-Montoro, Universidad Politécnica de Madrid (Spain)
   `farroyo@eui.upm.es`
3. Marc Barroso-Mancha, University of Barcelona (Spain)
   `marc.barroso4@gmail.com`
4. Diego R. Cabrera-Mendieta, Univ. Politécnica Salesiana (Ecuador)
   `dcabrera@ups.edu.ec`
5. Juan B. Castellanos-Peñuelas, Universidad Politécnica de Madrid (Spain)
   `jcastellanos@fi.upm.es`

6. Rodica Ceterchi, University of Bucharest (Romania)
rceterchi@gmail.com
7. Erzsébet Csuhaj-Varjú, Eötvös Loránd University (Hungary)
csuhaj@inf.elte.hu
8. Rudolf Freund, TU Wien (Austria)
rudi@emcc.at
9. Zsolt Gazdag, Eötvös Loránd University (Hungary)
gazdagzs@inf.elte.hu
10. Marian Gheorghe, University of Bradford (UK)
m.gheorghe@bradford.ac.uk
11. Carmen Graciani, Universidad de Sevilla (Spain)
cgdiaz@us.es
12. Ricardo Graciani-Díaz, University of Barcelona (Spain)
graciani@icc.ub.edu
13. José Luis Guisado-Lizar, Universidad de Sevilla (Spain)
jlguisado@us.es
14. Miguel Ángel Gutiérrez-Naranjo, Universidad de Sevilla (Spain)
magutier@us.es
15. Sergiu Ivanov, Université Paris Est (France)
sergiu.ivanov@u-pec.fr
16. Kristóf Kántor, University of Debrecen (Hungary)
kantor.kristof@inf.unideb.hu
17. Gábor Kolonits, Eötvös Loránd University (Hungary)
kolomax@inf.elte.hu
18. Alberto Leporati, Università degli Studi di Milano-Bicocca (Italy)
leporati@disco.unimib.it
19. Luis Felipe Macías-Ramos, Universidad de Sevilla (Spain)
lfmaciasr@us.es
20. Luca Manzoni, Università degli Studi di Milano-Bicocca (Italy)
luca.manzoni@disco.unimib.it
21. Giancarlo Mauri, Università degli Studi di Milano-Bicocca (Italy)
mauri@disco.unimib.it
22. Víctor Méndez-Muñoz, Autonomous University of Barcelona (Spain)
vmendez@caos.uab.cat
23. Alejandro Millán-Calderón, Universidad de Sevilla (Spain)
amillan@us.es
24. Laura Moreno-Valero, University of Barcelona (Spain)
95morenolaura@gmail.com
25. David Orellana-Martín, Universidad de Sevilla (Spain)
dorellana@us.es
26. Gheorghe Păun, Universidad de Sevilla (Spain) and Institute of Mathematics of the Romanian Academy (Romania)
gpaun@us.es

27. Mario de Jesús Pérez-Jiménez, Universidad de Sevilla (Spain)
marper@us.es
28. Antonio E. Porreca, Università degli Studi di Milano-Bicocca (Italy)
porreca@disco.unimib.it
29. Ariadna Ribes-Metidieri, University of Barcelona (Spain)
aribesmetidieri@gmail.com
30. Patricia Ribes-Metidieri,University of Barcelona (Spain)
ribesmetidieri@gmail.com
31. Agustín Riscos-Núñez, Universidad de Sevilla (Spain)
ariscosn@us.es
32. Gábor Román, Eötvös Loránd University (Hungary)
romangabor@caesar.elte.hu
33. Álvaro Romero-Jiménez, Universidad de Sevilla (Spain)
romero.alvaro@us.es
34. José María Sempere-Luna, Polytechnic University of Valencia (Spain)
jsempere@dsic.upv.es
35. Petr Sosík, University of Opava (Czech Republic)
petr.sosik@fpf.slu.cz
36. Óscar de la Torre Pérez, University of Barcelona (Spain)
oscar.delatorre.perez@gmail.com
37. Luis Valencia-Cabrera, Universidad de Sevilla (Spain)
lvalencia@us.es
38. György Vaszil, University of Debrecen (Hungary)
vaszil.gyorgy@inf.unideb.hu
39. Ana Ventura-Barroso, University of Barcelona (Spain)
a.venturabarroso@gmail.com
40. Julián Viejo-Cortés, Universidad de Sevilla (Spain)
julian@dte.us.es

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

We acknowledge the support obtained from the Department of Computer Science and Artificial Intelligence, as well as from "Vicerrectorado de Investigación" (*V Plan Propio de Investigación*), both of them from Universidad de Sevilla.

Mario J. Pérez-Jiménez
Agustín Riscos-Núñez
(August 2016)

Contents

Complexity of Simulating R Systems by P Systems

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

[1] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md
[2] Romanian Academy, Institute of Computer Science, Iaşi, Romania
Blvd. Carol I no.8, 700505 Iaşi, Romania
E-mail: bogdan.aman@gmail.com
[3] Faculty of Informatics, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria
E-mail: rudi@emcc.at
[4] Université Paris Est, France
E-mail: sergiu.ivanov@u-pec.fr

Summary. We show multiple ways to simulate R systems by non-cooperative P systems with atomic control by promoters and/or inhibitors, or with matter-antimatter annihilation rules, with a slowdown by a factor of constant. The descriptional complexity is also linear with respect to that of simulated R system. All these constants depend on how general the model of R systems is, as well as on the chosen control ingredients of P systems. Special attention is paid to the differences in the mode of rule application in these models.

1 Introduction. Differences between P and R

Membrane systems, also called P systems (non-distributed, with symbol-objects) are a formal model of (possibly controlled) multiset rewriting [8]. Reaction systems, also called R systems, is also a formal rewriting-like model of set evolution introduced in [6], see also a recent survey [5]. Both P systems and R systems are inspired by the functioning of the living cells. It is a natural task to compare R systems, which was introduced later, to P systems, by simulation. The application of a successful solution would be possibilities to use membrane computing tools and perspective for studying reaction systems. Some research comparing them was done in [10], more exactly, this paper considers P systems with no-persistence aspect of R systems, from the viewpoint of the computational power. We, however, first focus on comparing standard R systems to standard P systems by simulating the former with latter, and then revisit the direction of bringing aspects of R systems to the P systems model, verifying how closer this can make the models.

We start the explanation of the simplest case – triples of single objects. Rules in R systems have form (a, b, c), which loosely correspond to $a \to c|_{\neg b}$ in P systems, i.e., the first element is the reactant (in this paper we may also call it the left side) the second element is the product (in this paper we may also call it the right side), and the third element is the inhibitor, with the following differences in the mode of application.

The first difference is that the configuration is a set, not a multiset, and thus simultaneously producing the same symbol by multiple rules yields a single object. P systems with sets of objects instead of multisets of objects have been considered in [1], where they have been shown to be universal in the distributed P systems, both for the transitional model, and for the model with active membranes. However, in [1] the goal of showing universality was reached without actually using this first aspect (automatic reduction of multiple copies of identical object into one object), but rather by avoiding to ever need multiple copies of the same object (in the same region). This aspect, combined with the one below, are called the *threshold principle* in the literature. However, it is also meaningful to view them individually.

The second difference is that, if multiple rules with the same a in the left side exists, (if a is present in the configuration, for all of these rules where the inhibitors are not present in the configuration) **all** these rules are applied, simultaneously producing the corresponding products. (This comes from an inspiration that either the abundance of objects a is sufficient, or the replication and, possibly, proper control take place to guarantee the application of all such rules.) This second aspect is standard, e.g., in *H systems* [11] (together with the first one). The second aspect has been already considered also in P systems area, see, e.g., [3].

The third difference is that the objects are not persistent. This means that, even if an object does not undergo any rule, it still disappears from the configuration of the next step, unless, of course, it is produced by some rule. This third aspect is standard in time-varying distributed H systems [9, 12], (together with the first and second ones), and they relate especially naturally to *TVDH1* systems, see [7].

In the general case, the elements of the triples describing the rules of R systems are **sets** of objects. Hence, the meaning of the triple (A, B, C) is: the joint presence of objects in A, in the case when all objects in B are absent, leads to production of objects in C, and, moreover, the subsequent configuration is precisely equal to the union of the right sides of applicable rules (possibly united with the input context).

2 Preliminaries

The reader is assumed to be familiar with the basic notions of formal languages and membrane computing, see [13] for a comprehensive introduction and the webpage [15] of P systems.

The notation $(ncoo, pro_{k,l} + inh_{k',l'})$ describes the possible class of rules: non-cooperative evolution with at most k promoters of weight at most l and at most k'

inhibitors of weight at most l', see [4, 14]; the sign "+" here means both promoters and inhibitors are allowed to be used in the same rule, if it is not the case, we write a comma instead of a plus sign.

The notation $(ncoo, antim/pri)$ stands for non-cooperative evolution rules and matter-antimatter annihilation rules, with weak priority of *all* annihilation rules assumed over all other rules (the most studied variant of P systems with antimatter), see [2].

3 Using promoters and inhibitors

In fact, in terms of intuition from P systems, A is more similar to a promoter than a reactant (and there is no difference between a set of distinct atomic promoters and a corresponding one higher-weight promoter), and B corresponds to a set of atomic inhibitors (if B were a single higher-weight inhibitor, it would disable the rule when all its elements are present, not just any of them, which would not correspond to the correct definition). However, within the traditional P systems mode, we would additionally need to restrict the rule application to only once per step.

3.1 Using powerful rules

Hence, an arbitrary general R system with alphabet V of k symbols and rules (A_i, B_i, C_i), $1 \leq i \leq n$ could be written as the following P systems (non-cooperative, but with powerful promoters and inhibitors), having additional objects I_1 and d_i for all $1 \leq i \leq n$:

$$\begin{aligned}
&\Pi_0 = (O, \mu = [\ ]_1, w_1, R_1) \text{ where} \\
&O = V \cup \{d_i \mid 1 \leq i \leq n\} \cup I_1, \\
&R_1 = \{d_i \to \prod\nolimits_{c \in C_i} c|_{A_i, \{\neg b \mid b \in B_i\}},\ d_i \to \lambda|_{\neg A_i},\ d_i \to \lambda|_b \mid b \in B_i,\ 1 \leq i \leq n\} \\
&\quad \cup \{a \to \lambda \mid a \in V\} \cup \{I_1 \to I_1 \prod\nolimits_{1 \leq i \leq n} d_i\}.
\end{aligned}$$

This combination of features only takes one step to simulate a step of P systems, $n + k + 1$ symbols and $2n + k + 1 + \sum_{1 \leq i \leq n} |B_i|$ rules. Note that the first rule in the description of R_1 uses a higher-weight promoter *together* with a *set* of atomic inhibitors. Also note that in a special case when the rules of the simulated R system are triples of *single* symbols, the control used becomes atomic promoters *together* with atomic inhibitors.

In the rest of the paper we show how to achieve the same goal with P systems having more restricted rules, also discussing how to produce only *one* copy of symbols present in the simulated R system. We use promoters and inhibitors, then consider only one kind of these features, then we replace them by matter-antimatter annihilation rules, and finally, we discuss how much the problem is simplified if some of the aspects of R systems are assumed by the P systems model.

3.2 Triples of symbols

We start with the simplest case - when the elements of triples describing the rules are single elements. Consider such an R system S with alphabet V and rules $\{(a_i, b_i, c_i) \mid 1 \leq i \leq n\}$. We construct a P system Π_1 simulating S, where the initial configuration w_1 matches the initial configuration of S, and the following rules, the simulation taking only 2 steps:

$$\begin{aligned}
\Pi_1 &= (O, \mu = [\ ]_1, w_1, R_1) \text{ where} \\
O &= V \cup \{a' \mid a \in V\} \cup \{d_i \mid 1 \leq i \leq n\}, \\
R_1 &= \{a \to a' \prod\nolimits_{1 \leq i \leq n, a_i = a} d_i \mid a \in V\} \\
&\quad \cup \{d_i \to c_i|_{\neg (b_i)'},\ d_i \to \lambda|_{(b_i)'} \mid 1 \leq i \leq n\} \cup \{a' \to \lambda \mid a \in V\}.
\end{aligned}$$

The simulation task here is simple for two reasons: we took the simpler model of R systems, and using promoters besides inhibitors makes it possible to remove unneeded objects easily. We also note that the number of objects and rules can be decreased by not producing a' when a participates in the left side of any rule, and using $d_{\min\{j \mid 1 \leq j \leq n,\ a_j = b\}}$ instead of b' as promoter and inhibitor.

If $|V| = k$, then $|O| = 2k + n$ and $|R_1| = 2k + 2n$. Moreover, the optimization described in the previous paragraph decreases both $|O|$ and $|R_1|$ by the number of symbols appearing on the left side of some rule of S.

The multiplicities of symbols may grow. When the same symbol is produced simultaneously by multiple rules, the multiplicative effect happens. It is, however, fairly easy to reset the multiplicities of objects in V to one, at a cost of one more step, $2k + 3$ additional symbols in O and $3k + 3$ additional rules, also using an additional object I_1 in the initial configuration:

$$\begin{aligned}
\Pi_2 &= (O, \mu = [\ ]_1, w_1, R_1 = R_i \cup R_{ii} \cup R_{iii}) \text{ where} \\
O &= V \cup \{a', a'', \overline{a} \mid a \in V\} \cup \{d_i \mid 1 \leq i \leq n\}, \cup \{I_1, I_2, I_3\} \\
R_i &= \{a \to a' \prod\nolimits_{1 \leq i \leq n, a_i = a} d_i \mid a \in V\} \cup \{I_1 \to I_2\}, \\
R_{ii} &= \{d_i \to (c_i)''|_{\neg (b_i)'},\ d_i \to \lambda|_{(b_i)'} \mid 1 \leq i \leq n\} \\
&\quad \cup \{a' \to \lambda \mid a \in V\} \cup \{I_2 \to I_3 \prod\nolimits_{a \in V} \overline{a}\}, \\
R_{iii} &= \{\overline{a} \to a|_{a''},\ \overline{a} \to \lambda|_{\neg a''},\ a'' \to \lambda \mid a \in V\} \cup \{I_3 \to I_1\}.
\end{aligned}$$

3.3 Triples of sets

Now the task is more complicated. While generating a set C_i instead of symbol c_i is straightforward, instead of verifying that a_i is present and b_i is absent, rule applicability is defined as presence of **all** symbols from set A_i and absence of **all** symbols from set B_i. We recall that our task is a constant-time solution. Notice

that the rule is not applicable if and only if some symbol from A_i is absent or some symbol from B_i is present.

Consider such an R system S with alphabet V and rules $\{(A_i, B_i, C_i) \mid 1 \leq i \leq n\}$. We construct a P system Π_3 simulating S, where the initial configuration matches the initial configuration of S, plus an additional object I_1, and the following rules, the simulation taking only 3 steps:

$$\begin{aligned}
\Pi_3 &= (O, \mu = [\ ]_1, w_1, R_1) \text{ where} \\
O &= V \cup \{d_i \mid 1 \leq i \leq n\} \cup \{I_1, I_2, I_3\}, \\
R_1 &= \{I_1 \to d_1 \cdots d_n I_2\} \\
&\quad \cup \{d_i \to \lambda|_{\neg a},\ d_i \to \lambda|_b \mid a \in A_i,\ b \in B_i,\ 1 \leq i \leq n\} \cup \{I_2 \to I_3\} \\
&\quad \cup \{d_i \to \prod\nolimits_{c \in C_i} c|_{I_3} \mid 1 \leq i \leq n\} \cup \{a \to \lambda|_{I_3} \mid a \in V\} \cup \{I_3 \to I_1\}.
\end{aligned}$$

If $|V| = k$, then $|O| = k + n + 3$ and $|R_1| = k + n + 3 + \sum_{1 \leq i \leq k}(|A_i| + |B_i|)$. Notice also that, besides objects from V, no object ever appears in multiple copies. As for each object from V, its multiplicity represents the number of rules in S that has produced it in the last simulated step. Unlike the construction from the previous section, the multiplicative effect does not carry over to the next step of computation of S, since each object from V (except the instances in the starting configuration) is produced from some object d_i, produced in one copy, effectively resetting the multiplicities of the previous step. However, producing objects in V in a single copy requires additional overhead. Similarly to obtaining Π_2 from Π_1, we can obtain Π_4 from Π_3, at the price of one more step, $2k+1$ additional symbols in O and $3k + 1$ additional rules. We skip the details.

3.4 Using only promoters

It should not be any surprise that (in the maximally parallel mode) the effect of inhibitors can be obtained by non-cooperative rules with promoters only. Informally, to verify that some object b is absent, we first check if b is present by some rule $a \to a'|_b$, and it suffices to check in the next step whether a is unchanged. The reverse, i.e. replacing promoters with inhibitors, is even easier to see, since promoting a rule by b can be modeled by inhibiting a rule by some immediately-erased object b', creation of which is inhibited by b. We still think it is interesting to consider the use of only promoters or only inhibitors, for two reasons. First, the reduction of promoters/inhibitors in the *general* case of P systems is too complicated, and second, we would like to explore how little overhead in terms of slowdown and descriptional complexity would suffice to achieve our task.

First, as an exercise, we construct a P system for an R system S with triples of symbols $\{(a_i, b_i, c_i)\}$ as rules. The initial configuration matches the initial configuration of S, plus an additional object I_1.

$$
\begin{aligned}
\Pi_5 &= (O, \mu = [\ ]_1, w_1, R_1) \text{ where} \\
O &= V \cup \{a' \mid a \in V\} \cup \{d_i \mid 1 \le i \le n\} \cup \{I_1, I_2, I_3\}, \\
R_1 &= \{I_1 \to I_2\} \cup \{a \to a' \prod\nolimits_{1 \le i \le n, a_i = a} d_i \mid a \in V\} \\
&\quad \cup \{I_2 \to I_3\} \cup \{d_i \to \lambda|_{(b_i)'} \mid 1 \le i \le n\} \cup \{a' \to \lambda \mid a \in V\} \\
&\quad \cup \{I_3 \to I_1\} \cup \{d_i \to c_i|_{I_3}, \mid 1 \le i \le n\}.
\end{aligned}
$$

This construction is obtained from the first one with promoters and inhibitors, implementing the group of rules with inhibitors (contrasted with existing rules with the same objects as promoters) in the next step, promoted by "timer" I_3. We also note, similarly to Π_1, that the number of objects and rules can be decreased by not producing a' when a participates in the left side of any rule, and using $d_{\min\{j \mid 1 \le j \le n,\ a_j = b\}}$ instead of b' as promoter. Once again, this simulation has multiplicative effect, and the multiplicities can be reset to one, at the price of one more step, $2k+1$ additional symbols in O and $3k+1$ additional rules. Let us call the obtained system Π_6. We omit the details, only mentioning that instead of rules $\overline{a} \to \lambda|_{\neg a''}$ as in Π_2, we can erase these symbols in the next step by rules $\overline{a} \to \lambda|_{I_1}$.

Now consider the general case of simulating an R system S with alphabet V and rules $\{(A_i, B_i, C_i) \mid 1 \le i \le n\}$. The simulating P system below has the initial configuration which matches the initial configuration of S, plus additional objects I_1 and a' for each $a \in V$.

$$
\begin{aligned}
\Pi_7 &= (O, \mu = [\ ]_1, w_1, R_1 = R_i \cup R_{ii} \cup R_{iii}) \text{ where} \\
O &= V \cup \{a' \mid a \in V\} \cup \{d_i \mid 1 \le i \le n\} \cup \{I_1, I_2, I_3\}, \\
R_i &= \{I_1 \to d_1 \cdots d_n I_2\} \cup \{a' \to \lambda|_a \mid a \in V\}, \\
R_{ii} &= \{d_i \to \lambda|_{a'},\ d_i \to \lambda|_b \mid a \in A_i,\ b \in B_i,\ 1 \le i \le n\} \\
&\quad \cup \{a \to \lambda|_{I_2},\ a' \to \lambda|_{I_2} \mid a \in V\} \cup \{I_2 \to I_3\}, \\
R_{iii} &= \{d_i \to \prod\nolimits_{c \in C_i} c|_{I_3} \mid 1 \le i \le n\} \cup \{I_3 \to I_1 \prod\nolimits_{a \in V} a'\}.
\end{aligned}
$$

This construction is obtained from the second one with promoters and inhibitors, as follows. The role of objects a' is to survive for one step if and only if the corresponding object a is present, to be used as a promoter instead of inhibitor a; objects a' are recreated in the last step, for the next simulation cycle. Moreover, as now objects from V are no longer used as inhibitors, they can be removed one step earlier.

The system above needs only 3 steps to simulate a step of S, and if $|V| = k$, then $|O| = 2k + n + 3$ and $|R_1| = 3 + 3k + n + \sum_{1 \le i \le n}(|A_i| + |B_i|)$. Of course, alternatively, objects a' could be created from one additional initial object, at a price of an additional step and a few extra rules, but we currently focus on constructions that are efficient in time and descriptional complexity. We again comment that, although this construction has no multiplicative effect, the number

of copies of a symbol in V produced in the end of the simulation equals the number of rules in S that have produced this symbol in the last step. Producing exactly one copy needs one more step, $2k+1$ additional symbols in O and $3k+1$ additional rules. We call this system Π_8 and give no more details, since obtaining it from Π_7 is exactly like obtaining Π_6 from Π_5.

3.5 Using only inhibitors

First, as an exercise, we construct a P system for an R system S with triples of symbols $\{(a_i, b_i, c_i)\}$ as rules. The initial configuration matches the initial configuration of S, plus an additional object I_1.

$$
\begin{aligned}
\Pi_9 &= (O, \mu = [\ ]_1, w_1, R_1) \text{ where} \\
O &= V \cup \{a' \mid a \in V\} \cup \{d_i \mid 1 \le i \le n\} \cup \{I_1, I_2\}, \\
R_1 &= \{I_1 \to I_2\} \cup \{a \to a' \prod\nolimits_{1 \le i \le n, a_i = a} d_i \mid a \in V\} \\
&\quad \cup \{I_2 \to I_1\} \cup \{d_i \to c_i|_{\neg (b_i)'} \mid 1 \le i \le n\} \\
&\quad \cup \{d_i \to \lambda|_{\neg I_2} \mid 1 \le i \le n\} \cup \{a' \to \lambda|_{\neg I_2} \mid a \in V\}.
\end{aligned}
$$

This construction is obtained from the one with promoters and inhibitors, implementing the group of rules with promoters (contrasted with existing rules with the same objects as inhibitors) in the next step, inhibited by "timer" I_2. Moreover, removing objects a' is delayed for one step, to make sure that the rules inhibited by them in the second step are not applied in the third step. Notice also that the simulation of a computation step of S only takes two steps of computation in Π; the third step of computation cleaning objects d_i and a' overlaps with the first step of simulation of the next step in S. However, this produces no interference, since sub-alphabets $\{d_i \mid 1 \le i \le n\} \cup \{a' \mid a \in V\}$ and $\{I_1\} \cup V$ are disjoint. We also note, similarly to Π_1, that the number of objects and rules can be decreased by not producing a' when a participates in the left side of any rule, and using $d_{\min\{j \mid 1 \le j \le n,\ a_j = b\}}$ instead of b' as promoter.

The problem of multiplicative effect can be solved in the usual way, resetting multiplicities to one: produce one copy of each candidate-object, and erase the objects where the multiplicity is zero. However, with inhibitors it takes longer: one additional step to erase objects $\overline{a}$ when the corresponding object a'' is absent, and one further step to rewrite $\overline{a}$ into a.

$$
\begin{aligned}
\Pi_{10} &= (O, \mu = [\ ]_1, w_1, R_1) \text{ where} \\
O &= V \cup \{a', a'', \overline{a} \mid a \in V\} \cup \{d_i \mid 1 \le i \le n\} \cup \{I_1, I_2, I_3, I_4\}, \\
R_1 &= \{I_1 \to I_2\} \cup \{a \to a' \prod\nolimits_{1 \le i \le n, a_i = a} d_i \mid a \in V\} \\
&\quad \cup \{I_2 \to I_3 \prod\nolimits_{a \in V} \overline{a}\} \cup \{d_i \to (c_i)''|_{\neg (b_i)'} \mid 1 \le i \le n\}
\end{aligned}
$$

$$\cup \{I_3 \to I_4\} \cup \{d_i \to \lambda|_{\neg I_2} \mid 1 \leq i \leq n\} \cup \{a' \to \lambda|_{\neg I_2},\ \overline{a} \to \lambda|_{\neg a''} \mid a \in V\}$$
$$\cup \{I_4 \to I_1\} \cup \{\overline{a} \to a|_{\neg I_3},\ a'' \to \lambda|_{\neg I_3} \mid a \in V\}.$$

Hence, the total additional price for resetting the multiplicities of elements of V to one using only inhibitors is 2 more steps, $2k+2$ additional objects, and $3k+2$ rules.

Now consider the general case of simulating an R system S with alphabet V and rules $\{(A_i, B_i, C_i) \mid 1 \leq i \leq n\}$. The simulating P system below has the initial configuration which matches the initial configuration of S, plus additional objects I_1, J and b' for each $b \in V$.

$$\begin{aligned}
\Pi_{11} &= (O, \mu = [\ ]_1, w_1, R_1 = R_i \cup R_{ii} \cup R_{iii}) \text{ where} \\
O &= V \cup \{b', b'' \mid b \in V\} \cup \{d_i \mid 1 \leq i \leq n\} \cup \{I_1, I_2, I_3, J\}, \\
R_i &= \{I_1 \to d_1 \cdots d_n I_2 J\} \cup \{b' \to b''|_{\neg b} \mid b \in V\} \cup \{J \to \lambda\}, \\
R_{ii} &= \{d_i \to \lambda|_{\neg a},\ d_i \to \lambda|_{\neg b''} \mid a \in A_i,\ b \in B_i,\ 1 \leq i \leq n\} \\
&\quad \cup \{b' \to \lambda|_{\neg I_1} \mid b \in V\} \cup \{I_2 \to I_3,\ J \to \lambda\}, \\
R_{iii} &= \{d_i \to \prod\nolimits_{c \in C_i} c|_{\neg I_2} \mid 1 \leq i \leq n\} \\
&\quad \cup \{a \to \lambda|_{\neg J} \mid a \in V\} \cup \{b'' \to \lambda|_{\neg I_2} \mid b \in V\} \cup \{I_3 \to I_1 J \prod\nolimits_{b \in V} b'\}.
\end{aligned}$$

This construction is obtained from the second one with promoters and inhibitors, as follows. The role of objects b' is to change into b'' if and only if the corresponding object b is present, so b'' can be used as an inhibitor instead of promoter b; objects b' are recreated in the last step, for the next simulation cycle. Moreover, to make sure the rules erasing d_i in the absence of a are not applied in the third step, objects a can only be removed in the third step. This is why an additional object J is present in each of the first two steps of the simulation, inhibiting premature removal of objects a. The rule erasing J is written both in R_i and R_{ii} only to highlight that it is applied both in the first and in the second step.

The system above needs only 3 steps to simulate a step of S, and if $|V| = k$, then $|O| = 3k + n + 4$ and $|R_1| = 4 + 4k + n + \sum_{1 \leq i \leq n}(|A_i| + |B_i|)$. Of course, alternatively, objects b' could be created from one additional initial object, at a price of an additional step and a few extra rules, but we currently focus on constructions that are efficient in time and descriptional complexity. Resetting to one the multiplicities of objects in V can be done exactly how Π_{10} was constructed from Π_9. Hence, the new system Π_{12} will have, compared to Π_{11}, 2 more steps, $2k + 2$ additional objects, and $3k + 2$ rules.

4 Using antimatter

This section is devoted to a different control mechanism: matter-antimatter annihilation rules are used instead of promoters and/or inhibitors. The weak priority of annihilation rules over non-cooperative rules is assumed, which is the most common variant of the antimatter model. First, we notice that erasing with a promoter, say, $d \to \lambda|_b$, in the case the promoting object b is erased without being used anywhere else, and when the number of copies of d is bounded, can be modeled by antimatter as follows:

- replace the promoting object b by anti-object d^- of the promoted object, in sufficient copies to erase all possible copies of promoted object d,
- add erasing rules for this anti-object d^- to remove the copies of the anti-objects which did not annihilate.

We now construct the P system equivalent to Π_1 using antimatter.

$$\begin{aligned}\Pi_{13} &= (O = V \cup \{d_i, d_i^- \mid 1 \le i \le n\}, \mu = [\ ]_1, w_1, R_1) \text{ where}\\ R_1 &= \{a \to \prod\nolimits_{1\le i\le n, b_i=a} d_i^- \prod\nolimits_{1\le i\le n, a_i=a} d_i \mid a \in V\}\\ &\quad \cup \{d_i d_i^- \to \lambda,\ d_i \to c_i,\ d_i^- \to \lambda \mid 1 \le i \le n\}.\end{aligned}$$

In each rule of the first group of R_1, it is enough to produce a single copy of d_i^-, because at most one d_i may be generated by the system in the same step, since the rule uniquely determines its left side. The simulation only takes two steps, and uses $k + 2n$ objects and $k + 3n$ rules.

This construction, too, has multiplicative effect. Resetting multiplicities to one can be done by two-step annihilation. Say, we got some number (possibly zero) of objects c'', and we only want to know whether this number is positive. Then we produce one copy of $(c'')^-$ and rewrite it to c' if it does not immediately annihilate. One step later, we produce one copy of $(c')^-$, and rewrite it to c if it does not immediately annihilate. As a result, c will appear if and only if $(c')^-$ did not annihilate, i.e., c' did not appear one step before. But this happened if and only if $(c'')^-$ was annihilated, i.e., there was at least one copy of c'' two steps before. Performing this routine to objects in V of Π_{13}, we obtain the following system, using an additional starting object I_1:

$$\begin{aligned}\Pi_{14} &= (O, \mu = [\ ]_1, w_1, R_1) \text{ where}\\ O &= V \cup \{a', a'', (a')^-, (a'')^- \mid a \in V\} \cup \{d_i, d_i^- \mid 1 \le i \le n\} \cup \{I_1, I_2, I_3, I_4\},\\ R_1 &= \{a \to \prod\nolimits_{1\le i\le n, b_i=a} d_i^- \prod\nolimits_{1\le i\le n, a_i=a} d_i \mid a \in V\} \cup \{I_1 \to I_2\}\\ &\quad \cup \{d_i d_i^- \to \lambda,\ d_i \to (c_i)'',\ d_i^- \to \lambda \mid 1 \le i \le n\} \cup \{I_2 \to I_3 \prod\nolimits_{a\in V} (a'')^-\}\\ &\quad \cup \{a''(a'')^- \to \lambda,\ (a'')^- \to a' \mid a \in V\} \cup \{I_3 \to I_4\}\\ &\quad \cup \{a'(a')^- \to \lambda,\ (a')^- \to a \mid a \in V\} \cup \{I_4 \to I_1\}.\end{aligned}$$

As you can see, resetting multiplicities with antimatter has a price of two more steps, $4k+4$ additional objects and $4k+4$ additional rules.

Now consider the general case of simulating an R system S with alphabet V and rules $\{(A_i, B_i, C_i) \mid 1 \leq i \leq n\}$. The simulating P system below has the initial configuration which matches the initial configuration of S, plus additional object I_1.

$$
\begin{aligned}
\Pi_{15} &= (O, \mu = [\ ]_1, w_1, R_1 = R_i \cup R_{ii} \cup R_{iii} \text{ where} \\
O &= V \cup \{a', (a')^-, a'', \mid a \in V\} \cup \{d_i, d_i^- \mid 1 \leq i \leq n\} \cup \{I_1, I_2, I_3\}, \\
R_i &= \{I_1 \to I_2 d_1 \cdots d_n \prod\nolimits_{a \in V} a'\} \cup \{a \to (a')^- a'' \mid a \in V\}, \\
R_{ii} &= \{a'(a')^- \to \lambda,\ a' \to d_i^-,\ b'' \to d_i^-,\ (a')^- \to \lambda \\
&\qquad \mid a \in A_i,\ b \in B_i,\ 1 \leq i \leq n\} \cup \{I_2 \to I_3\}, \\
R_{iii} &= \{d_i d_i^- \to \lambda,\ d_i \to \prod\nolimits_{c \in C_i} c,\ d_i^- \to \lambda \mid 1 \leq i \leq n\} \cup \{I_3 \to I_1\}.
\end{aligned}
$$

Symbols from C_i are produced from d_i if and only if it is not annihilated, i.e., neither a' nor b'' should produce d_i^- for any $a \in A_i$, $b \in B_i$. Since a' is annihilated if and only if a is present, and b'' is not produced if and only if b is absent, the simulation of an application of rule i of the R system happens if and only if all symbols from the first set are present and all symbols from the second set are absent. The simulation takes 3 steps, using the alphabet of $4k + 2n + 3$ symbols and the set of $3k + 3n + 3 + \sum_{1 \leq i \leq n}(|A_i| + |B_i|)$ rules.

This construction produces each symbol in multiplicity equal to the number of rules of S that produced it, not carrying the multiplicative effect to the next step. If needed, resetting multiplicities can be done costing two more steps, $4k+2$ additional objects and $4k+2$ additional rules. We call this system Π_{16}, and provide no more details since it is obtained from Π_{15} exactly as Π_{14} is obtained from Π_{13}.

5 Non-standard P systems

Some difficulty of simulation of R systems by P systems lie in the difference of their standard derivation modes. We would like to discuss how varying this may affect the problem.

First, if we consider P systems **with sets** instead of multisets, where production of a symbol multiple times still yields a single copy of the result, then *all* constructions in this paper still hold literally, i.e., no changes in the description of these P systems is needed. However, some things may become simpler, e.g., in this case resetting multiplicities to one is done by the model, and does not require additional time, symbol and rule complexity.

We note that, in a non-distributed case, P systems with sets of objects are no longer universal, since the number of possible configuration is bounded by two to

the power of the cardinality of the alphabet. However, universality is not needed to simulate R systems (which has also been shown in the case of deterministic P systems with promoters and/or inhibitors).

Second, if we consider P systems which deterministically apply *all* individually applicable rules, even with overlapping left sides (i.e., competing for resources), then of course the existing solutions still literally hold, but in some cases there are much easier ways: we would have no need to explicitly produce multiple objects from one. For instance, the constructions in this paper usually involve production of rule labels d_i, either from the corresponding reactant a_i, or from some "timer" object I_j, and then have different rules processing these label objects. In this "auto-replication" mode, these various processing rules could be applied directly to the corresponding original object a_i or I_j, the replication being done by the model itself. This would definitely simplify the simulation. Let us refer to this aspect as **auto-replication**. For example, the set of rules of system Π_1 can be simplified to the following:

$$\{a_i \to c_i|_{\neg b_i} \mid 1 \le i \le n\} \cup \{a \to \lambda \mid a \in V\},$$

i.e., just one step, no additional objects and k additional rules. The problem with resetting the multiplicities is also simpler:

$$\begin{aligned}
&\Pi = (O, \mu = [\ ]_1, w_1, R_1 \text{ where}\\
&O = V \cup \{a' \mid a \in V\} \cup \{I_1, I_2\},\\
&R_1 = \{I_1 \to I_2\} \cup \{a_i \to (c_i)'|_{\neg b_i} \mid 1 \le i \le n\} \cup \{a \to \lambda \mid a \in V\}\\
&\quad \cup \{I_2 \to I_1\} \cup \{I_2 \to c|_{c'}\} \cup \{c' \to \lambda \mid c \in V\},
\end{aligned}$$

i.e., requiring only one more step, $k+2$ additional symbols and $2k+2$ additional rules (compared to increase of complexity of Π_2 over Π_1 by one step, $2k+3$ symbols and by $3k+3$ rules).

Third, if we consider P systems where idle objects (i.e., those not consumed by applied rules) do not contribute to the next configuration, we call this aspect "**no persistence**", then many erasing rules (in particular, all erasing promoted or inhibited by some "timer" I_j) would no longer be needed, while occasionally some renaming rules should be added when object was designed to be used later than in the next step after its production. For instance, in case of no-persistence, all $n+k'$ erasing rules of Π_1 may be removed, leaving just $n+k$ rules. Similarly, all erasing rules of Π_2 may be removed; hence, the subtask of resetting the multiplicities to one in this case only needs $k+3$ additional rules instead of $3k+3$.

However, testing for presence of some object b by "failing to apply a rule with inhibitor b and finding the reactant unchanged in the next step" would not work. The working solution is to use b as an inhibitor in a rule producing some object b', and to use b' as an inhibitor in the next step. Testing for absence by "failing to apply a rule with a promoter and finding the reactant unchanged in the next

step" would be no longer possible, so the model with promoters only seems to be considerably weaker in the case without persistence of idle objects.

We would like to note that, in case of P systems with sets and auto-replication, the aspect of no-persistence can be simulated as follows: add rules $a \to \lambda$ for each $a \in V$; they will make sure that such objects are not carried over to the next step, in the same time not adding anything to the result (as for productive objects, erasing them is just another option, which in the auto-replication case neither grows nor shrinks the set of objects obtained from them). This simulation takes one step, k objects and $n + k$ rules.

And, of course, if we consider P systems with all these differences, i.e., with sets, auto-replication, and without object persistence, then rule (a, b, c) of R systems becomes *identical* to rule $a \to c|_{\neg b}$ of P systems, while rule (A, B, C) of R systems becomes *identical* to rule $\prod_{a \in A} a \to \prod_{c \in C} c|_{\{\neg b | b \in B\}}$, so the simulation is trivial, requiring one step, k objects and n rules of type $(ncoo, pro_{1,*} + inh_{*,1})$.

6 Conclusions

We recall that although deterministic P systems with promoters and/or inhibitors are not universal and have subregular characterizations, their power is sufficient to simulate R systems.

All constructions presented in this paper (except those in previous section) simulate R systems (in their standard derivation mode) by P systems (in *their* standard derivation mode), with the slowdown by a factor of constant, where the descriptional complexity of the simulating P system is linear with respect to the descriptional complexity of the simulating P system. The proportionality constants vary depending on whether R systems are defined as triples of symbols or as triples of sets of symbols, and on whether promoters, inhibitors or both are used in P systems. All constructions are deterministic: while the multiset of rules to be applied to a given configuration may not be unique, the next configuration is unique. Indeed, in all these constructions, if two rules have the same left side, then either their applicability is mutually exclusive (one is being promoted and the other one is being inhibited by the same symbol), or also the right side is the same (and thus, if there are multiple choices which object would promote or inhibit the rule, such choice would not influence the result).

Seventeen constructions are presented, see Table 1: 0)(general and simple in particular) R systems using single higher-weight promoters together with multiple atomic inhibitors, 1)simple R systems using promoters and inhibitors, 2)simple R systems using promoters and inhibitors and resetting multiplicities to one, 3)general R systems using promoters and inhibitors, 4)general R systems using promoters and inhibitors and resetting multiplicities to one, 5)simple R systems using promoters, 6)simple R systems using promoters and resetting multiplicities to one, 7)general R systems using promoters, 8)general R systems using promoters and resetting multiplicities to one, 9)simple R systems using inhibitors, 10)simple

P	R	mult	features	steps	$\|O\|$	$\|R_1\|$
Π_0	s	L	$(ncoo, pro_{1,1} + inh_{1,1})$	1	$n+k+1$	$3n+k+1$
Π_0	G	L	$(ncoo, pro_{1,*} + inh_{*,1})$	1	$n+k+1$	$2n+k+1+T'$
Π_1	s	M	$(ncoo, pro_{1,1}, inh_{1,1})$	2	$n+k+k'$	$2n+k+k'$
Π_2	s	1	$(ncoo, pro_{1,1}, inh_{1,1})$	3	$n+3k+k'+3$	$2n+4k+k'+3$
Π_3	G	L	$(ncoo, pro_{1,1}, inh_{1,1})$	3	$n+k+3$	$n+k+3+T$
Π_4	G	1	$(ncoo, pro_{1,1}, inh_{1,1})$	4	$n+3k+4$	$n+4k+4+T$
Π_5	s	M	$(ncoo, pro_{1,1})$	3	$n+k+k'+3$	$2n+k+k'+3$
Π_6	s	1	$(ncoo, pro_{1,1})$	4	$n+3k+k'+4$	$2n+4k+k'+4$
Π_7	G	L	$(ncoo, pro_{1,1})$	3	$n+2k+3$	$n+3k+3+T$
Π_8	G	1	$(ncoo, pro_{1,1})$	4	$n+4k+4$	$n+6k+4+T$
Π_9	s	M	$(ncoo, inh_{1,1})$	2	$n+k+k'+2$	$2n+k+k'+2$
Π_{10}	s	1	$(ncoo, inh_{1,1})$	4	$n+3k+k'+4$	$2n+4k+k'+4$
Π_{11}	G	L	$(ncoo, inh_{1,1})$	3	$n+3k+4$	$n+4k+4+T$
Π_{12}	G	1	$(ncoo, inh_{1,1})$	5	$n+5k+6$	$n+6k+6+T$
Π_{13}	s	M	$(ncoo, antim/pri)$	2	$2n+k$	$3n+k$
Π_{14}	s	1	$(ncoo, antim/pri)$	4	$2n+5k+4$	$3n+5k+4$
Π_{15}	G	L	$(ncoo, antim/pri)$	3	$2n+4k+3$	$3n+3k+3+T$
Π_{16}	G	1	$(ncoo, antim/pri)$	5	$2n+8k+5$	$3n+7k+5+T$

Table 1. Comparative table of simulation of R systems by P systems

R systems using inhibitors and resetting multiplicities to one, 11)general R systems using inhibitors, 12)general R systems using inhibitors and resetting multiplicities to one, 13)simple R systems using antimatter, 14)simple R systems using antimatter and resetting multiplicities to one, 15)general R systems using antimatter, 16)general R systems using antimatter and resetting multiplicities to one. The table below shows the number of steps of simulating P system to simulate one step of R system, alphabet size and the number of rules in these simulations (n is the number of rules in S, k is the number of symbols in S, k' is the number of symbols that do not appear in the left side of any rule of the simulated system; by T we denote $\sum_{1 \leq i \leq k}(|A_i| + |B_i|)$ and by T' we denote $\sum_{1 \leq i \leq k} |B_i|$). Column R describes the type of simulated system, where s stands for simple (rules with triples of symbols) and G stands for general (rules with triples of sets). Column mult describes the multiplicities of symbols in the simulating P system, where M stands for multiplicative effect, L stands for last multiplicity, and 1 stands for multiplicities 0 and 1. Column features describes the kinds of rules used.

Note: in Π_6, Π_8 and Π_9, intermediate objects are removed one step later, in parallel with the first step of simulation of the next step of evolution of the simulated R system, but not interfering with it.

Finally, in the previous section we discussed how (qualitatively and quantitatively) adopting some aspects of R systems (such as sets instead of multisets, auto-replication or no-persistence) into the working model of P systems simplifies simulation of R systems.

References

1. A. Alhazov. P systems without multiplicities of symbol-objects. *Information Processing Letters*, 100(3):124–129, 2006.
2. A. Alhazov, B. Aman, and R. Freund. P systems with anti-matter. In M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, and C. 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 66–85. Springer, 2014.
3. A. Alhazov, S. Cojocaru, A. Colesnicov, L. Malahova, and M. Petic. A P system for annotation of Romanian affixes. In A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers*, volume 8340 of *Lecture Notes in Computer Science*, pages 80–87. Springer, 2013.
4. A. Alhazov and R. Freund. Asynchronous and maximally parallel deterministic controlled non-cooperative P systems characterize NFIN and coNFIN. In E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, and Gy. 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.
5. R. Brijder, A. Ehrenfeucht, M. G. Main, and G. Rozenberg. A tour of reaction systems. *International Journal of Foundations of Computer Science*, 22(7):1499–1517, 2011.
6. A. Ehrenfeucht and G. Rozenberg. Reaction systems. *Fundamenta Informaticae*, 75(1):263–280, 2007.
7. M. Margenstern, Yu. Rogozhin, and S. Verlan. Time-varying distributed H systems with parallel computations: The problem is solved. In J. Chen and J. H. Reif, editors, *DNA Computing, 9th International Workshop on DNA Based Computers, DNA9, Madison, WI, USA, June 1-3, 2003, revised Papers*, volume 2943 of *Lecture Notes in Computer Science*, pages 48–53. Springer, 2003.
8. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
9. Gh. Păun. DNA computing based on splicing: universality results. *Theoretical Computer Science*, 231(2):275–296, 2000.
10. Gh. Păun and M. J. Pérez-Jiménez. Towards bridging two cell-inspired models: P systems and R systems. *Theoretical Computer Science*, 429:258–264, 2012.
11. Gh. Păun, G. Rozenberg, and A. Salomaa. Computing by splicing. *Theoretical Computer Science*, 168(2):321–336, 1996.
12. Gh. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing - New Computing Paradigms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1998.
13. Gh. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
14. D. Sburlan. Further results on P systems with promoters/inhibitors. *International Journal of Foundations of Computer Science*, 17(1):205–221, 2006.
15. The P Systems Website. `http://ppage.psystems.eu`.

Purely Catalytic P Systems over Integers and Their Generative Power

Artiom Alhazov[1], Omar Belingheri[2], Rudolf Freund[3], Sergiu Ivanov[4], Antonio E. Porreca[2], and Claudio Zandron[2]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md
[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
E-mail: {o.belingheri@campus,porreca@disco,zandron@disco}.unimib.it
[3] Faculty of Informatics, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria
E-mail: rudi@emcc.at
[4] Université Paris Est, France
E-mail: sergiu.ivanov@u-pec.fr

Summary. We further investigate the computing power of the recently introduced P systems with $\mathbb{Z}$-multisets (also known as hybrid sets) as generative devices. These systems apply catalytic rules in the maximally parallel way, even consuming absent non-catalysts, effectively generating vectors of arbitrary (not just non-negative) integers. The rules may be made inapplicable only by dissolution rules. However, this releases the catalysts into the immediately outer region, where new rules might become applicable to them. We discuss the generative power of this model. Finally, we consider the variant with mobile catalysts.

1 Introduction

Membrane systems (cell-like, with symbol-objects) have traditionally been viewed as collections of hierarchically arranged multiset processors [12]. In the list of open problems disseminated in 2015 [11], Gheorghe Păun suggested going beyond the traditional setting where symbol multiplicities in multisets are restricted to non-negative integers. One suggested approach [6] defines generalized multisets as taking multiplicities from arbitrary finitely generated, totally ordered commutative groups.

In work [3], a different approach is taken: only catalytic rules are allowed, and the applicability of a rule only depends on presence of the corresponding catalyst

in the given region. Consuming an absent non-catalyst makes its multiplicity negative. While in [3] it was already established that such model is not universal, we found it interesting to investigate its generative power more precisely.

Since the number of catalysts remains finite and does not change throughout the computation, this induces a finite set of "rule teams" which can be applied in parallel in one step. The virtual absence of applicability conditions and the finiteness of the "teams" hints at the possibility of seeing them as integer vectors; in this case the P system itself can be seen as evolving by sequentially adding such vectors (possibly having negative components) to the contents of its membranes. Paper [2] compares this general model to vector addition systems [5, 9] (adapted to allow negative vector components [8]) and blind register machines [7].

Here we return to the particular model from [3], discussing the lower bound of its generative power and giving some results on the variant with target indications.

2 Preliminaries

The reader is assumed to be familiar with the basic notions of formal languages and membrane computing; see [13] for a comprehensive introduction to both. We only remark that, as common in membrane computing, multisets in $O^\circ = \mathbb{N}^O$ are represented by strings in O^*, keeping in mind that the order of symbols is not relevant.

2.1 Extending Multisets

To represent also negative multiplicities, multisets must be extended. A $\mathbb{Z}$-multiset, allowing integer multiplicities (called a *hybrid set* in [4]) would be from $\mathbb{Z}^O$; it can be represented by a string in $(O \cup O^-)^*$, where $O^- = \{a^- \mid a \in O\}$ is a set of symbols that represents objects in multiplicity "negative one". Note that, as opposed to P systems with matter-antimatter [1], symbol a^- here is not an actual object, but simply a convenient way to represent a deficit of a, and the actual multiplicity of a represented by a string w is $|w|_a - |w|_{a^-}$. We also do not distinguish between notations a^{-k} and $(a^-)^k$. The superscript $^-$ can be used as a morphism, producing a multiset with opposite multiplicities, e.g., $(a^k)^-$ represents the same $\mathbb{Z}$-multiset as the one in the previous sentence. As the strings here are only used to represent [$\mathbb{Z}$-] multisets, we may write an equality sign between the strings representing the same [$\mathbb{Z}$-] multiset. For conciseness, let us use the notation $O^\bullet = (O \cup O^-)^*$. Finally, since it will be always clear from the context, we may call an element of $O^\bullet$ "multiset", omitting the word "representing". Assuming an order is fixed on O, for $u \in O^\bullet$, vector $(|u|_a - |u|_{a^-})_{a \in O}$ is denoted by $\psi_O(u)$; the subscript O may be omitted when it is clear from the context. This vector is called the *Parikh image* of u.

2.2 Linear Sets

The *linear* set generated by a set of vectors $A = \{\mathbf{a}_i \mid 1 \leq i \leq d\} \subset \mathbb{Z}^n$ and an offset $\mathbf{a}_0 \in \mathbb{Z}^n$ is defined as follows:

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

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

We use the notation $\mathbb{Z}^n LIN_{\mathbb{N}} = \left\{ \langle A, \mathbf{a}_0 \rangle_{\mathbb{N}} \mid A \in (\mathbb{Z}^n)^d,\ \mathbf{a}_0 \in \mathbb{Z},\ m \in \mathbb{N} \right\}$, to refer to the class of all linear sets. Semilinear sets are defined as finite unions of linear sets. We use the notations $\mathbb{Z}^n SLIN_{\mathbb{N}}$ to refer to the classes of semilinear 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:

$$\begin{aligned} &\mathbb{Z}^n SLIN_{\mathbb{N}}^U = \left\{ \bigcup\nolimits_{\mathbf{b} \in B} \langle A, \mathbf{b} \rangle_{\mathbb{N}} \mid A \in (\mathbb{Z}^n)^d,\ B \in (\mathbb{Z}^n)^k,\ d, k \in \mathbb{N} \right\} \\ &= \left\{ \left\{ \mathbf{b} + \sum\nolimits_{i=1}^{d} k_i \mathbf{a}_i \mid k_i \in \mathbb{N},\ 1 \leq i \leq d \right\} \mid A \in (\mathbb{Z}^n)^d,\ B \in (\mathbb{Z}^n)^k,\ d, k \in \mathbb{N} \right\}. \end{aligned}$$

Let us denote these sets by $\langle A, B \rangle_{\mathbb{N}}$.

3 Purely Catalytic P Systems over Integers

In purely catalytic P systems over integers the set of objects is a disjoint union of catalysts C and the regular objects O. The regular objects are allowed to have any integer multiplicity, while the catalysts are only allowed to appear in a non-negative number of copies.

The rules can be of the two following types:

- *catalytic* rules: $cu \to cv$, where $c \in C$ and $u, v \in O^*$;
- catalytic rules with *dissolution*: $cu \to cv\delta$, where $c \in C$, $u, v \in O^*$, and $\delta \notin C \cup O$ is the symbol indicating membrane dissolution.

The rules applied in parallel cannot involve more catalysts than available in the system; the multiplicities of regular objects, on the other hand, do not influence the applicability of rules. An application of a rule $cu \to cv$ in a region containing cw ($c \in C$, $u, v \in O^*$, $w \in O^\bullet$ produces $cw(cu)^- cv = cwv(u^-)$, or, in terms of vectors, ignoring the catalyst, vector $\psi(w) + \psi(v) - \psi(u)$ is represented by the contents of that region after the rule has been applied. An application of a rule $cu \to cv\delta$ produces the same effect, and then dissolves the enclosing membrane, moving the contents of the dissolved membrane into the parent membrane.

Purely catalytic P systems over integers evolve under the maximally parallel semantics, so each catalyst enters exactly one rule (non-deterministically chosen), unless the given region has no rules associated with this catalyst. By

$Z^d O_{\mathbb{Z}} P_m(pcat_k, \delta)$ we denote the family of sets of d-dimensional vectors of integers generated by purely catalytic P systems over integers with dissolution, at most m membranes and at most k catalysts. If any of parameters d, m, k is unbounded, it is replaced by $*$ in the notation.

We also use notations for extended features (listed in parentheses in the notation of the sets of $\mathbb{Z}$-vectors generated by the corresponding families of P systems). Target indications, denoted by tar, allow the non-catalysts to be sent to a different membrane. In the right side of the rules, sending object a is written by (a, tar), where $tar \in \{out\} \cup \{in_j \mid 1 \leq j \leq m\}$; j here is a label of immediately inner membrane. In this paper, we may write tar_n in the notation of a set of $\mathbb{Z}$-vectors generated by a family of P systems; this generalization reflects the possibility to assign targets even to negative multiplicites of objects.

Another feature is *mobile catalysts* [10], i.e., targets may also be associated to the catalysts, and thus the catalysts move across the membrane structure; we denote this feature by $mpcat_k$ since the systems we consider are purely catalytic. We use the plus sign between the features of catalytic mobility and dissolution when it is allowed for the *same* rule to move a catalyst and to dissolve the membrane currently containing it.

4 Results

4.1 Simplifications and Observations

First, we would like to explicitly allow rules of the form $c \to cx$, ($c \in C$, $x \in O^\bullet$), i.e., the multiset of regular objects in the left side being empty. This does not change the model, since any $\mathbb{Z}$-multiset x can be written as $u(v^-)$, $u, v \in O^*$, and, fixing some $a \in O$, $c \to cx$ is equivalent to $cau \to av$. Moreover, any rule $cu \to cv$ is equivalent to $c \to cu(v^-)$, so it suffices to only consider rules of types $c \to cx$ and $c \to cx\delta$ ($c \in C$, $x \in O^\bullet$).

Second, notice that it is enough to start with a single catalyst in any region, because it can perform the role of any number of catalysts, and if multiple catalysts are initially in the same region, they will always stay in the same region (possibly, merged with others). Indeed, take an arbitrary region of an arbitrary purely catalytic P system over integers, say, it has catalysts c_i, $1 \leq i \leq d$, and each catalyst c_i has associated rules $c_i \to c_i x_{i,j}$, $1 \leq j \leq n_i$, where $x_{i,j} \in O^\bullet \cup O^\bullet\delta$. Note that if none of the catalysts has associated rules, then they are equivalent to a single catalyst with no associated rules, so in the following we assume the contrary. If some catalyst c_i has no associated rules, it is then equivalent to it having associated a single rule $c_i \to c_i$, i.e., $x_{i,1} = \lambda$ and $n_i = 1$, so in the following we assume $n_i \geq 1$ for $1 \leq i \leq d$. We can now replace all these catalysts by a single catalyst c having associated the following set of rules:

$$\{c \to cx_{1,j_1} \cdots x_{d,j_d} \mid 1 \leq j_i \leq n_i,\ 1 \leq i \leq d\}.$$

On the other side, no catalyst in some region is equivalent to one catalyst with no associated rules. Therefore, without restricting the generality, in the following we assume that in the initial configuration of an arbitrary purely catalytic P system over integers, each membrane region i, $1 \leq i \leq m$, contains precisely one catalyst, and we can call it c_i.

Third, notice that no information enters membranes, so the outer regions cannot affect the inner regions in any way. Hence, if the output region i_0 is not the skin, then only the membrane substructure inside i_0, including i_0 is relevant for the result, and other membranes are irrelevant and may be removed without affecting the result, making i_0 the skin (unless some rule in some removed membrane had applicable rules, but could never be dissolved, in which case the generated set of vectors is empty, which is a degenerate case). So in the following, we assume that the output region is always the skin.

Fourth, every elementary membrane having no rules associated to the catalysts available there may be removed from the system without affecting the result (unless it is the output membrane, in which case a singleton is generated, which is a degenerate case), so in the following we assume that each elementary membrane has some applicable rules. Clearly, the P system will not reach the halting until this membrane is dissolved.

Consider this reasoning starting from the elementary membranes outside, by induction. Take any non-elementary membrane i which becomes elementary during a computation. Assume i is not dissolved (i.e., it has no rules associated to any of the catalysts that were placed within the membrane substructure inside i, including i), but it is not the output membrane. Then all the computation in the membrane substructure inside i, including i, does not contribute to the result, and can be removed from the system without affecting the result.

As a summary of the fourth observation, without restricting the generality (except, possibly the degenerate cases generating the empty set or some singleton), we may assume that any purely catalytic P system over integers has applicable rules associated to all elementary membranes, and all membranes except the skin must be dissolved at some moment during the computation.

Finally, for every region except the skin, a catalyst c_i without associated rules is equivalent to a catalyst with a rule $c_i \to c_i$. Hence, without restricting the generality, we may assume that the catalysts are *never* idle before the halting is reached. Clearly, (excluding the degenerate case generating the empty set), the skin should have no rules associated to any catalyst of the system.

We would like to note that even without pruning the membrane structure by removing membrane substructures not contributing to the result, the membrane structure obtained at halting (if at all reachable) is unique.

We recall that in [2], the following generalization approach is taken: There is a finite number of reachable membrane structures. These could be used as states of a sequential P system, which may be obtained, separately for each membrane structure, by combining the behavior of all catalysts in all regions of the P system. Indeed, having fixed a reachable membrane structure, we know which membranes

have been dissolved, and thus the resulting location of each catalyst. Then, for each catalyst, associated rules in its current location are considered and combined, similarly to the second observation above, but globally. Having obtained a sequential system, the catalyst is no longer needed. Then, in [2] it was shown that such a generalization is nothing else but a sequential blind vector addition system with states, and it was claimed that it characterizes precisely the family of all semilinear vectors of integers.

Indeed, in this way any purely catalytic P system over integers can be substituted by a sequential blind vector addition system with states, so the upper bound of the family of all semilinear sets of vectors of integers, or, equivalently, the family of all integer vector sets, generated by blind register machines, holds. However, the reverse is not necessarily true, i.e., it does not follow that for any sequential blind vector addition system with states there would exist an equivalent purely catalytic P system over integers.

Another result in [2] has been obtained for integer vector addition P systems, namely Theorem 5. That model has been shown to characterize exactly the uniform semilinear sets. However, since in the model of integer vector addition P systems, as opposed to purely catalytic P systems over integers, there is no concept of a catalyst, dissolving a membrane only disables rules of that region, without enabling rules that, in purely catalytic P systems over integers, are contained in the parent region and associated to the catalysts that were in the dissolved region. Hence, the characterization from Theorem 5 of [2] has no direct implication on the power of purely catalytic P systems over integers.

Therefore, at this point in the present paper we would like to definitely deviate into the particularities of how dissolution affects the computation, and the lower bounds.

4.2 Generative Power

We recall that we discuss the family of integer vector sets generated by purely catalytic P systems over integers, with the usual halting condition.

Since the output region cannot be dissolved by definition and any other applicable rule can never be stopped, single-membrane purely catalytic P systems over integers are degenerate:

$$Z^d O_{\mathbb{Z}} P_1(pcat_*, \delta) = \{\emptyset\} \cup \{\{v\} \mid v \in \mathbb{Z}^d\}.$$

For simplicity, we will not mention these degenerate cases while considering multiple membranes.

With two membranes, a characterization is still straightforward:

$$Z^d O_{\mathbb{Z}} P_2(pcat_*, \delta) = \mathbb{Z}^d SLIN_{\mathbb{N}}^U.$$

Indeed, let A be the finite set of vectors corresponding to the non-dissolving rules in the elementary membranes, and let B be the finite set of sums of two vectors:

the one corresponding to the initial configuration and vectors corresponding to the dissolving rules in the elementary membrane; the skin should have no rules. If the catalyst in the elementary membrane is c_2, then the correspondence mentioned above is $c_2 \to c_2 x \leftrightarrow \psi(x)$, and similarly with dissolution. An arbitrary computation of a P system consists of an arbitrary number of applications of non-dissolving rules and one application of a dissolving rule. Hence, the resulting vector sums up from the "initial" vector, one arbitrary "dissolving" vector, and an arbitrary linear combination of "non-dissolving" vectors.

It is worth noting that, by a similar reasoning, for a P system with multiple membranes, if the chronological order of dissolving membranes is fixed, the result is still $\mathbb{Z}^d SLIN_{\mathbb{N}}^U$. Indeed, each combination of rules (one for each catalyst) yields one vector, so all such possible combinations of non-dissolving rules yield a finite set of vectors, and multiple non-dissolving steps yield a linear set generated by these vectors. Thus, over the whole computation the result sums up from the initial configuration, a finite number of dissolution vectors, and a finite number of linear sets corresponding to the membrane structures reached during that computation. Since the total number of chronological orders of dissolving membranes is bounded, the known result already follows:

$$Z^d O_{\mathbb{Z}} P_*(pcat_*, \delta) \subseteq \mathbb{Z}^d SLIN_{\mathbb{N}}.$$

Even with three membranes, in case two of them are elementary, the power of such purely catalytic P systems over integers is still $\mathbb{Z}^d SLIN_{\mathbb{N}}^U$, but for a different reason: each elementary membrane contributes with its uniform semilinear set, and a sum of two uniform semilinear sets is still uniform semilinear.

Let us now examine a P system with three nested membranes – the minimal number to obtain a set which is not in $\mathbb{Z}^d SLIN_{\mathbb{N}}^U$. Let the vector obtained by joining the initial contents of all membranes be $\mathbf{a}$, the set of non-dissolving vectors of the elementary membrane be A_3, the set of dissolving vectors of the elementary membrane be B_3, the sets of non-dissolving and dissolving vectors in the middle membrane associated to catalyst c_2 are A_2 and B_2, and the similar sets associated to catalyst c_3 (which will arrive from the elementary membrane) are A and B. Let us see what the resulting vector set is built from, besides $\mathbf{a}$.

A non-dissolving computation in three membranes adds at each step (an element of) A_3 to the elementary membrane and (an element of) A_2 to the middle membrane. Eventually all objects will arrive to the skin, so the three-membrane phase of the computation will contribute by (an arbitrary element of) $\langle A_2 + A_3 \rangle_{\mathbb{N}}$.

Then there are two possibilities. If membrane 2 is dissolved first, then the system continues computing by only applying the rules in membrane 3, and eventually dissolving membrane 3, yielding $B_2 + \langle A_3 \rangle_{\mathbb{N}} + B_3$. However, if membrane 3 is dissolved first, then both catalysts are active in membrane 2, eventually dissolving it, yielding $B_3 + \langle A_2 + A \rangle_{\mathbb{N}} + (B_2 + A \cup A_2 + B \cup A + B)$. The expression in parentheses corresponds to applying at least one dissolving rule. Therefore, the set of integer vectors generated by such a purely catalytic P system over integers with three nested membranes is

$$M = \mathbf{a}+B_3+\langle A_2+A_3\rangle_{\mathbb{N}}+\Big(B_2+\langle A_3\rangle_{\mathbb{N}} \cup \langle A_2+A\rangle_{\mathbb{N}}+\big(B_2+A \cup A_2+B \cup B_2+B\big)\Big),$$

and the power of all three-membrane purely catalytic P systems over integers, noting that the power of the nested case subsumes the power of the case with two elementary membranes, is

$$Z^dO_{\mathbb{Z}}P_1(pcat_*,\delta) = \{M \mid \mathbf{a} \in \mathbb{Z}^d,\ A_2, A_3, B_2, B_3, A, B \in FIN(\mathbb{Z}^d)\},$$

where M is the expression above. Unfortunately, it is not obvious what can be simplified in it, except B_3 can subsume $\mathbf{a}$. So we try to analyze it in details, possibly going into particular cases.

All terms in the expression M are bounded except three: $\langle A_3 + A_2\rangle_{\mathbb{N}}$, $\langle A_3\rangle_{\mathbb{N}}$ and $\langle A + A_2\rangle_{\mathbb{N}}$. These terms are not independent, even though A_2, A_3 and A are three independent finite sets of vectors. It is, however, possible to separate them in a particular case when $|A_3| = 1$, choosing $A_2 = -A_3$ and $A = C - A_2$. Since A_3 is a singleton, the identity $A_3 - A_3 = \{\mathbf{0}\}$ holds, so the three unbounded terms become $\langle\{\mathbf{0}\}\rangle_{\mathbb{N}}$, $\langle A_3\rangle_{\mathbb{N}}$ and $\langle C\rangle_{\mathbb{N}}$, so we are getting close to obtaining a union of two particular linear (or even uniform semilinear) sets with different base vectors.

Indeed, if we choose $\mathbf{a} = \mathbf{0}$, $B_3 = \{\mathbf{0}\}$, $B_2 = \{\mathbf{0}\}$, $B = \{\mathbf{0}\}$ and $A_3 = \{\mathbf{e}\}$, expression M simplifies to $\langle\{\mathbf{e}\}\rangle_{\mathbb{N}} \cup \langle C\rangle_{\mathbb{N}} + (C + \{\mathbf{e}\} \cup \{\mathbf{0}\})$, which can be rewritten as $\langle\{\mathbf{e}\}\rangle_{\mathbb{N}} \cup \langle C\rangle_{\mathbb{N}} \cup \{\mathbf{e}\}\langle C\rangle_{\mathbb{N}}$.

Alternatively, to avoid dealing with the union of three cases when membrane 2 is divided last, if we choose $B_2 = A_2$ and $B = A$, then the last parenthesis in the general expression of set M becomes simply $A_2 + A = C$. Choosing $\mathbf{a} = \mathbf{0}$, $B_3 = \{\mathbf{0}\}$, and $A_3 = \{\mathbf{e}\}$, expression M simplifies to $\langle\{\mathbf{e}\}\rangle_{\mathbb{N}} - \{\mathbf{e}\} \cup \langle C\rangle_{\mathbb{N}} + C$. Since $\mathbf{0} \in \langle\{\mathbf{e}\}\rangle_{\mathbb{N}} - \{\mathbf{e}\}$ and $\langle C\rangle_{\mathbb{N}} + C \cup \{\mathbf{0}\} = \langle C\rangle_{\mathbb{N}}$, in this case we can rewrite M to

$$-\{\mathbf{e}\} \cup \langle\{\mathbf{e}\}\rangle_{\mathbb{N}} \cup \langle C\rangle_{\mathbb{N}},$$

which is a union of any two homogeneous linear sets, such that the first one has only one generator, united with the opposite vector of that generator. Hence,

$$Z^dO_{\mathbb{Z}}P_n(cat,\delta) \supsetneq Z^dSLIN^U_{\mathbb{N}},\ n \geq 3.$$

What if $B = \emptyset$, i.e., catalyst c_3 has no associated dissolution rules in region 2? Then the general expression of set M is immediately simplified to

$$M = \mathbf{a} + B_2 + B_3 + \langle A_2 + A_3\rangle_{\mathbb{N}} + (\langle A_3\rangle_{\mathbb{N}} \cup \langle A_2 + A\rangle_{\mathbb{N}} + A),$$

and in our case of $A_3 = \{\mathbf{e}\}$, $A_2 = -\{\mathbf{e}\}$ and $A = C + \{\mathbf{e}\}$, M becomes

$$\mathbf{a} + B_2 + B_3 + (\langle\{\mathbf{e}\}\rangle_{\mathbb{N}} \cup \langle C\rangle_{\mathbb{N}} + C + \{\mathbf{e}\}),$$

and choosing $\mathbf{a} + B_2 + B_3 = \{-\mathbf{e}\}$, and noticing that C 0 times is covered by $\mathbf{e}$ 0 times and $\langle C\rangle_{\mathbb{N}} + C \cup \{\mathbf{0}\} = \langle C\rangle_{\mathbb{N}}$, we simplify M to $\{-\mathbf{e}\} \cup \langle\{\mathbf{e}\}\rangle_{\mathbb{N}} \cup \langle C\rangle_{\mathbb{N}}$, i.e., an "almost clean union" we already obtained before. Finally, we notice that we can equivalently write it as

$$\langle\{\mathbf{e}\}, -\mathbf{e}\rangle_{\mathbb{N}} \cup \langle C\rangle_{\mathbb{N}}.$$

Continuing the current approach with more membranes would only result in more cases.

4.3 Communication

We would like to remark that adding target indications to the regular objects should not increase the power of purely catalytic P systems over integers. Indeed, looking at a purely catalytic P system over integers, it is easily decidable which membranes will eventually be dissolved. Hence, the only question is whether the contents of a region specified by target, after possible dissolutions, will be in the output. There is no need to examine the future of a moved regular object, since the resources in purely catalytic P systems over integers are unbounded, and we can view this copy of a moved object as staying in that region until the end of the computation.

However, if also the catalysts are allowed to have target indications associated, it does make a difference. We claim the following characterizations.

$$Z^dO_{\mathbb{Z}}P_*(mpcat_k, tar_n) = \mathbb{Z}^dSLIN_{\mathbb{N}},\ k \geq 1,$$
$$Z^dO_{\mathbb{Z}}P_*(mpcat_k + \delta) = \mathbb{Z}^dSLIN_{\mathbb{N}},\ k \geq 1,$$
$$Z^dO_{\mathbb{Z}}P_*(mpcat_*, \delta) = \mathbb{Z}^dSLIN_{\mathbb{N}},$$

The upper bound in either case is easy to see because the number of possible arrangements of catalysts across the given membrane structure (and any possible structures obtained from it by membrane dissolutions) is bounded. Hence, purely catalytic P systems over integers with mobile catalysts are still not more powerful than blind vector-addition systems with states, which characterize $\mathbb{Z}^*SLIN_{\mathbb{N}}$, see [2]. We now proceed to $\supseteq$ inclusions.

Consider an arbitrary semilinear set $\bigcup_{1\leq i\leq m}\langle A_i, b_i\rangle_{\mathbb{N}}$, where for each i, $1 \leq i \leq m$, A_i is a finite set, $A_i \cup \{b_i\} \subseteq \mathbb{Z}^d$. We construct the following purely catalytic P system over integers

$$\begin{aligned}
\Pi_1 &= (O, C, \mu, w_1, \cdots, w_{2m+1}, R_1, \cdots, R_{2m+1}, i_0 = 1) \text{ where}\\
O &= \{a_i \mid 1 \leq i \leq d\},\ C = \{c\},\\
\mu &= [\,[\,[\,\,]_{m+2}\,]_2 \cdots [\,[\,\,]_{2m+1}\,]_{m+1}\,]_1,\\
w_1 &= c,\ w_{i+1} = \lambda,\ 1 \geq i \geq 2m,\\
R_1 &= \{c \to (c, in_{i+1})v_i \mid 1 \leq i \leq m,\ \psi(v_i) = b_i\},\\
R_{i+1} &= \{c \to c(v, out) \mid \psi(v) \in A_i\} \cup \{c \to (c, in_{m+i+1})\},\ 1 \leq i \leq m,\\
R_{m+i+1} &= \emptyset,\ 1 \leq i \leq m.
\end{aligned}$$

The work of Π_1 consists of a non-deterministic choice of i-th linear set to generate, by moving catalyst c into membrane $i+1$ and producing b_i. After sending to the skin an arbitrary combination of vectors from A_i, the catalyst enters membrane $m+i+1$ and the system halts.

The system Π_2 is obtained from Π_1 by replacing the sets R_{i+1} of rules, $1 \leq i \leq m$, by

$$\{c \to cv \mid \psi(v) \in A_i\} \cup \{c \to (c, in_{m+i+1})\delta\}.$$

It works just as Π_1, with one difference, Here, instead of sending v out (possibly containing negative multiplicities), the linear combination of vectors from A_i is generated directly in membrane $i+1$, and is released into the skin upon dissolution of membrane $i+1$, simultaneously with sending the catalyst into the elementary membrane $m+i+1$. Now consider the following purely catalytic P system over integers.

$$
\begin{aligned}
\Pi_3 &= (O, C, \mu, w_1, \cdots, w_{3m+1}, R_1, \cdots, R_{3m+1}, i_0 = 1) \text{ where} \\
O &= \{a_i \mid 1 \le i \le d\},\ C = \{c_i \mid 1 \le i \le m+1\}, \\
\mu &= [\,[\,[\,[\,]_{2m+2}\,]_{m+2}\,]_2 \cdots [\,[\,[\,]_{3m+1}\,]_{2m+1}\,]_{m+1}\,]_1, \\
w_1 &= c_1,\ w_{i+1} = \lambda,\ 1 \le i \le 2m, \\
w_{2m+1+i} &= c_{1+i},\ 1 \ge i \ge m, \\
R_1 &= \{c_1 \to (c_1, in_{i+1})v_i \mid 1 \le i \le m,\ \psi(v_i) = b_i\}, \\
R_{i+1} &= \{c_1 \to c_1 v \mid \psi(v) \in A_i\} \\
&\quad \cup \{c_1 \to (c_1, in_{m+i+1}),\ c_i \to c_i\delta\},\ 1 \le i \le m, \\
R_{m+i+1} &= \{c_1 \to (c_1, in_{2m+i+1}),\ c_i \to (c_i, out)\},\ 1 \le i \le m, \\
R_{2m+i+1} &= \{c_1 \to c_1\delta\},\ 1 \le i \le m.
\end{aligned}
$$

The basic idea is the same, but the implementation is a little longer. To each linear set i, $1 \le i \le n$, three nested membranes are associated ($i+1$, $m+i+1$ and $2m+i+1$). The beginning is just like in the case of Π_2, until catalyst c_1 is sent into membrane $m+i+1$, but membrane $i+1$ is not dissolved yet. Then, c_1 enters the elementary membrane $2m+i+1$ and dissolves it, releasing catalyst c_{i+1} into the surrounding membrane $m+i+1$. Clearly, c_1 cannot reenter membrane $2m+i+1$, which no longer exists, so it has no applicable associated rules. Catalyst c_i, however, is sent out to membrane $i+1$, and dissolves it, which releases all generated regular objects to the skin and halts the computation. This proves the characterizations.

5 Conclusions

We have reproved that the power of purely catalytic P systems over integers is contained in the family of all semilinear sets of vectors of integers. We then have shown that with one membrane purely catalytic P systems over integers give degenerate results, and with two membranes they are characterized exactly by the family of all uniform semilinear sets of vectors of integers. With more membranes, this equality becomes a strict inclusion, and a specific union of linear sets with different base vectors have been obtained. More specifically, for any vector $\mathbf{e} \in \mathbb{Z}^d$ and any finite set $C \subseteq \mathbb{Z}^d$, purely catalytic P systems over integers can generate

$$\langle \{\mathbf{e}\}, -\mathbf{e} \rangle \cup \langle C \rangle_{\mathbb{N}}.$$

The most interesting open question remaining is whether $Z^*O_{\mathbb{Z}}P_*(pcat_*, \delta)$ is closed under union. While in almost all cases in membrane computing closure under union is trivial, e.g., by making a non-deterministic choice in the first step of the computation, the current situation is rather surprising.

Finally, we have considered the variants with mobile catalysts, and showed a few combinations of features leading to characterizations of semilinear sets of $\mathbb{Z}$-vectors.

References

1. A. Alhazov, B. Aman, R. Freund, and Gh. Păun. Matter and anti-matter in membrane systems. In H. Jürgensen, J. Karhumäki, and A. Okhotin, editors, *Descriptional Complexity of Formal Systems: 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*, pages 65–76. Springer, 2014.
2. A. Alhazov, O. Belingheri, R. Freund, S. Ivanov, A. E. Porreca, and C. Zandron. Semilinear sets, register machines, and integer vector addition (P) systems. *This volume*, 2016.
3. O. Belingheri, A. E. Porreca, and C. Zandron. P systems with hybrid sets, 2016. Workshop on Membrane Computing, submitted.
4. J. Carette, A. P. Sexton, V. Sorge, and S. M. Watt. Symbolic domain decomposition. In S. Autexier, J. Calmet, D. Delahaye, P. D. F. Ion, L. Rideau, R. Rioboo, and A. P. Sexton, editors, *Intelligent Computer Mathematics, 10th International Conference, AISC 2010, 17th Symposium, Calculemus 2010, and 9th International Conference, MKM 2010, Paris, France, July 5-10, 2010. Proceedings*, volume 6167 of *Lecture Notes in Computer Science*, pages 172–188. Springer, 2010.
5. R. Freund, O. Ibarra, Gh. Păun, and H.-C. Yen. Matrix languages, register machines, vector addition systems. *Third Brainstorming Week on Membrane Computing*, pages 155–167, 2005.
6. R. Freund, S. Ivanov, and S. Verlan. P systems with generalized multisets over totally ordered abelian groups. In *Int. Conf. on Membrane Computing*, volume 9504 of *Lecture Notes in Computer Science*, pages 117–136. Springer, 2015.
7. S. A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3):311–324, 1978.
8. C. Haase and S. Halfon. Integer vector addition systems with states. In J. Ouaknine, I. Potapov, and J. Worrell, editors, *Reachability Problems: 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, pages 112–124. Springer, 2014.
9. J. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979.
10. S. N. Krishna and A. Păun. Results on catalytic and evolution-communication P systems. *New Generation Computing*, 22(4):377–394, 2004.
11. Gh. Păun. Some quick research topics. `http://www.gcn.us.es/files/OpenProblems_bwmc15.pdf`.
12. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
13. Gh. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.

Semilinear Sets, Register Machines, and Integer Vector Addition (P) Systems

Artiom Alhazov[1], Omar Belingheri[2], Rudolf Freund[3],
Sergiu Ivanov[4], Antonio E. Porreca[2], and Claudio Zandron[2]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md
[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
E-mail: {o.belingheri@campus,porreca@disco,zandron@disco}.unimib.it
[3] Faculty of Informatics, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria
E-mail: rudi@emcc.at
[4] Université Paris Est, France
E-mail: sergiu.ivanov@u-pec.fr

Summary. In this paper we consider P systems working with multisets with integer multiplicities. We focus on a model in which rule applicability is not influenced by the contents of the membrane. We show that this variant is closely related to blind register machines and integer vector addition systems. Furthermore, we describe the computational power of these models in terms of linear and semilinear sets of integer vectors.

1 Introduction

P systems have been traditionally viewed as hierarchical processors of multisets [11]. In the list of open problems disseminated in 2015 [10], Gheorghe Păun suggested going beyond the traditional setting and considering multisets in which objects would not be restricted to having natural multiplicities. Several possible approaches have been suggested since then, including the one from [3], which defines generalised multisets as taking multiplicities from finitely generated, totally ordered Abelian groups.

The work [1] takes a different approach — the objects of the P system are partitioned into two classes: regular objects, which may have any integer multiplicity, and "catalysts", which may only appear in a bounded number of copies and cannot be consumed without being immediately reproduced. Thus, the regular objects cannot influence the applicability of rules, while the always bounded

catalysts induce a finite set of "rule teams" which can be applied in parallel in one step. The virtual absence of applicability conditions and the finiteness of the "teams" hints at the possibility of seeing them as integer vectors; in this case the P system itself can be seen as evolving by sequentially adding such vectors to the contents of its membranes.

Even though this vision is quite reminiscent of the folklore vector addition systems (VAS), the latter model is actually limited to having natural vectors as configurations [2, 8]. On the other hand, P systems manipulating integer multisets allow symbols with negative multiplicities to appear. It turns out that vector addition systems *without* the limitation of having natural configurations (integer VAS) have received relatively little attention in the literature [7].

Another related model which has received notoriously little attention are the blind register machines, whose registers are allowed to range over the whole set of integers. Blind counter automata have been introduced and studied as string recogniser devices by Sheila Greibach in [6]; their adaptation to recognising vectors of integer numbers seems quite relevant to the study of multisets with integer multiplicities.

In the present work we bring together the three models — P systems over integer multisets as defined in [1], integer vector addition systems, and blind register machines — and formally show the connections between their different variants. We also give detailed characterisations of their computing power in terms of linear and semilinear sets of natural and integer vectors.

The article is structured as follows. Section 2 recalls some notions used throughout the paper, in particular semilinear sets and vector addition systems. Section 3 gives a general definition of a register machine over a set A, and then defines blind, partially blind, and conventional register machines within this general framework. Section 4 defines the model of integer vector addition P systems and gives some details as to their semantics. Section 5 investigates the power of blind register machines and gives characterisations in terms of semilinear sets of vectors. Finally, Section 6 studies the power integer vector addition systems with and without membranes, and compares different variants of the models between themselves and with blind register machines.

2 Preliminaries

The reader is assumed to be familiar with the basic notions of formal languages and membrane computing; see [12] for a comprehensive introduction to both.

2.1 Linear Sets

The $\mathbb{N}$-*linear* set of $\mathbb{Z}$-vectors (or just *linear set* of $\mathbb{Z}$-vectors) generated by a set of vectors $A = \{\mathbf{a}_1, \cdots, \mathbf{a}_d\} \subseteq \mathbb{Z}^n$ and an offset $\mathbf{a}_0 \in \mathbb{Z}^n$ is defined as follows:

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

We underline that the vectors are over $\mathbb{Z}$, but the coefficients are from $\mathbb{N}$ (we will also consider the special case when $A \cup \{a_0\} \subseteq \mathbb{N}^n$; this would be an $\mathbb{N}$-*linear* set of $\mathbb{N}$-vectors, or just a linear set, a well-known concept from Formal Language Theory).

A $\mathbb{Z}$-linear set of $\mathbb{Z}$-vectors

$$\langle A, \mathbf{a}_0 \rangle_{\mathbb{Z}} = \left\{ \mathbf{a}_0 + \sum_{i=1}^{d} k_i \mathbf{a}_i \mid k_i \in \mathbb{N},\ 1 \leq i \leq d \right\}$$

can be considered, too. It corresponds precisely to the *linear vector space* notion from the classic course of Linear Algebra. However, it is also a particular case. Indeed, it is easy to see that $\langle A, \mathbf{a}_0 \rangle_{\mathbb{Z}} = \langle B, \mathbf{a}_0 \rangle_{\mathbb{N}}$ for

$$B = \{\mathbf{a}_1, \cdots, \mathbf{a}_d, -\mathbf{a}_1, \cdots, -\mathbf{a}_d\}.$$

If the offset $\mathbf{a}_0$ is the zero vector, we call the corresponding linear set *homogeneous*.

A *positive-restricted* $\mathbb{N}$-linear set of $\mathbb{Z}$-vectors generated by A and an offset $\mathbf{a}_0$ is defined to be the $\mathbb{N}$-linear set of $\mathbb{Z}$-vectors generated by A, restricted to non-negative vectors only:

$$\langle A, a_0 \rangle_{\mathbb{Z}}^{+} = \{\mathbf{x} \in \langle A, a_0 \rangle_{\mathbb{Z}} \mid \mathbf{x} \geq 0\},$$

where $\mathbf{x} \geq 0$ means that every component of $\mathbf{x}$ is non-negative.

We will use the notations $\mathbb{Z}^n LIN_{\mathbb{N}}$, $\mathbb{N}^n LIN_{\mathbb{N}}$, $\mathbb{Z}^n LIN_{\mathbb{Z}}$, and $\mathbb{Z}_+^n LIN_{\mathbb{N}}$ to refer to the classes of all $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors, $\mathbb{N}$-linear sets of $\mathbb{N}$-vectors, $\mathbb{Z}$-linear sets of $\mathbb{Z}$-vectors, and positive restricted $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors of dimension n, correspondingly. Semilinear sets are defined as finite unions of the corresponding types of linear sets. We will use the notations $\mathbb{Z}^n SLIN_{\mathbb{N}}$, $\mathbb{N}^n SLIN_{\mathbb{N}}$, $\mathbb{Z}^n SLIN_{\mathbb{Z}}$ and $\mathbb{Z}_+^n SLIN_{\mathbb{N}}$ to refer to the families of $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors, $\mathbb{N}$-semilinear sets of $\mathbb{N}$-vectors, $\mathbb{Z}$-semilinear sets of $\mathbb{Z}$-vectors, a nd positive-restricted $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors of dimension n, respectively. In case no particular restriction is imposed on the dimension, n will be replaced by $*$. We may omit n if $n = 1$.

We recall the following general result from number theory known as *Bézout's identity*. Given a set of integers $A = \{a_1, \cdots, a_n\} \subseteq \mathbb{Z}$, there exist integers $x_1, \cdots, x_n \in \mathbb{Z}$ such that the following holds:

$$\sum_{i=1}^{n} x_i a_i = \gcd(a_1, \cdots, a_n),$$

where $\gcd(a_1, \ldots, a_n)$ in the greatest common divisor of the integers from A. Furthermore, the greatest common divisor is the smallest positive integer which can be obtained as a linear combination of the elements of A.

2.2 Vector Addition Systems

A *vector addition system* (VAS) of dimension $n \in \mathbb{N}$ is defined to be the pair $(\mathbf{w}_0, W)$, where $\mathbf{w}_0 \in \mathbb{N}^n$ is the start vector, and W is a finite set of vectors from $\mathbb{Z}^n$, called addition vectors. An addition vector $\mathbf{w} \in W$ is said to be enabled in a vector $\mathbf{x} \in \mathbb{N}^n$ if $\mathbf{x} + \mathbf{w} \in \mathbb{N}^n$, i.e. all the components of the vector $\mathbf{x} + \mathbf{w}$ are non-negative. A VAS evolves from the start vector $\mathbf{w}_0$ by sequentially iterating the addition of vectors from W.

A *vector addition system with states* (VASS) is a VAS equipped with a finite state control. Essentially, state labels are assigned to addition vectors and a graph of states is given which defines the possible sequences of application of addition vectors.

We will use the notation *VAS* and *VASS* to refer to the families of sets of natural vectors which can be generated by VAS and VASS, respectively.

It was shown in [8] that VASS are equivalent in expressive power to VAS (without states): any n-dimensional VASS can be simulated by an equivalent $(n+3)$-dimensional VAS.

A variation on the model of vector addition systems consists in lifting the restriction that the valid vectors must have non-negative components. This model has recently been defined in [7].

An *integer vector addition system* ($\mathbb{Z}$-VAS) of dimension $n \in \mathbb{N}$ is the pair $(\mathbf{w}_0, W)$, where $\mathbf{w}_0 \in \mathbb{Z}^n$ is the start vector, and $W \subseteq \mathbb{Z}^n$ is finite set of addition vectors. A $\mathbb{Z}$-VAS evolves from $\mathbf{w}_0$ by sequentially applying the addition vectors from W. The set of vectors generated by a $\mathbb{Z}$-VAS is defined to be the set of reachable vectors.

An *integer vector addition systems with states* ($\mathbb{Z}$-VASS) is a $\mathbb{Z}$-VAS equipped with a state control and is defined as a tuple $(\mathbf{w}_0, Q, q_0, q_h, p, \delta)$, where $\mathbf{w}_0 \in \mathbb{Z}^n$ is the start vector, Q is a finite set of state labels, $q_0 \in Q$ is the starting state, $q_h \in Q$ is the halting state, $p : Q \setminus \{q_h\} \to \mathbb{Z}^n$ is a function assigning a vector to every state from $Q \setminus \{q_h\}$, and $\delta : Q \to 2^Q$ is a state transition function assigning to each state the set of possible successor states.

A $\mathbb{Z}$-VASS starts in $\mathbf{w}_0$ and in state q_0, applies the addition vector $p(q_0)$, and non-deterministically moves into one of the states from $\delta(q_0)$. This process is iteratively repeated, until the halting state q_h is reached. The vector language generated by a $\mathbb{Z}$-VASS is defined as the set of all vectors which are reachable in the halting state q_h.

We will use the notations $\mathbb{Z}$-*VAS* and $\mathbb{Z}$-*VASS* to refer to the sets of integer vectors generated by $\mathbb{Z}$-VAS or $\mathbb{Z}$-VASS.

3 Register Machines

Definition 1. *A* register machine over the set A *is the tuple* $M_A = (n, A, Q, q_0, q_h, P)$*, where* $n \in \mathbb{N}$*,* A *is a (possibly infinite) register alphabet,* Q *is a finite set of state labels,* q_0 *is the initial state,* $q_h \in Q$ *is the halting state, and*

P is a mapping associating an instruction to every state of M_A. An instruction *is a function $p : A^n \to A^n \times 2^Q$ associating to every n-tuple of values from A another n-tuple of such values and a set of states from Q. A configuration $C \in Q \times A^n$ of M_A is a tuple combining a state and n values from A.*

M_A can be seen as storing values of type A in its n registers. A configuration of M_A therefore defines its current state and the values of its n registers. When in state $q \in Q$, M_A can execute the instruction $P(q)$, which will compute (1) new values for *all* registers of M_A and (2) a set of possible new states; M_A can non-deterministically transition into one of these states.

Definition 2. *A k-step (finite)* computation *of the register machine $M_A = (n, A, Q, q_0, q_h, P)$ is a finite sequence of configurations $(C_i)_{0 \leq i \leq k}$ such that,*

1. *$C_0 = (q_0, \mathbf{a}_0)$, where some of the components of $\mathbf{a}_0$ (registers) may contain input values;*
2. *$C_k = (q_h, \mathbf{a}_k)$, where some of the components of $\mathbf{a}_n$ (registers) may contain output values;*
3. *for every $0 \leq i < k$, $C_i = (q_i, \mathbf{a}_i)$, $C_j = (q_j, \mathbf{a}_j)$, $P(q_i)(a_i) = (\mathbf{a}_j, H)$, and $q_j \in H$.*

M_A therefore transitions from a configuration to another by sequentially applying its instructions. Whenever M_A is in state q_i, it retrieves the corresponding instruction $P(q_i)$ and applies it to the tuple describing the values of the registers. The result, $P(q_i)(a_i)$, gives the new values for the registers and a set of states H from which M_A picks q_j and moves into it. The last configuration C_h is habitually referred to as the *halting configuration*.

Often, in order to be able to express the instructions in a sensible way, one considers some kind of structure over the set A; one example of such a structure may be a finitely generated Abelian group. Classical definitions of register machines rely on (sub)sets of integers and on the associated structure of a linearly ordered finitely generated Abelian group.

In what follows, we describe the existing models of register machines using the abstract language we have just introduced, and we show that blind register machines actually represent the *least* restricted variant.

Definition 3. *A* blind register machine *is a register machine B over the finitely generated Abelian group $(\mathbb{Z}, +)$. The instructions of blind register machines can be of the following two types:*

1. *$(ADD(i), q, s)(a_1, \ldots, a_i, \ldots, a_n) = \big((a_1, \ldots, a_i + 1, \ldots, a_n), \{q, s\}\big)$, and*
2. *$(SUB^*(i), q)(a_1, \ldots, a_i, \ldots, a_n) = \big((a_1, \ldots, a_i - 1, \ldots, a_n), \{q\}\big)$.*

The computations of blind register machines are defined as a computation of the corresponding register machine over $(\mathbb{Z}, +)$.

Definition 4. *A blind register machine* accepts *an input vector by resetting all registers to zero in the halting configuration. A blind register machine* generates *(or* computes *from an input) a vector of numbers by resetting all registers not containing the output to zero.*

We will use the notation $Ps_{\mathbb{Z}}BRM$ (resp., $Ps_{\mathbb{N}}BRM$) to refer to the class of sets of vectors of integer (resp., natural) numbers *accepted* by blind register machines.

All other well known types of register machines can be defined as subtypes of blind register machines.

Definition 5. *A* partially blind register machine *is a blind register machine whose registers are only allowed to contain non-negative numbers: for any computation $(C_i)_{1\leq i\leq k}$ of a partially-blind register machine and for any $C_i = (q_i, \mathbf{a}_i)$, $1 \leq i \leq k$, every component of $\mathbf{a}_i$ is non-negative.*

Thus, if the partially blind register machine B' decides at some point to decrement a register whose value is already zero, it will produce an illegal configuration which will render the whole computation invalid. This means that B' still cannot check its registers for zero, but it knows that all of them are non-negative at any given time. The computations of partially blind machines therefore satisfy a condition which renders them strictly stronger than blind register machines [6]: the registers may never go below zero.

Definition 6. *A partially blind register machine* accepts *an input vector by resetting all registers to zero in the halting configuration. A partially blind register machine* generates *(or* computes *from an input) a vector of numbers by resetting all registers not containing the output to zero in the halting configuration.*

We will use the notation $PsPBRM$ to refer to the class of sets of vectors of natural numbers accepted by partially blind register machines.

We can now also define conventional register machines in our general framework.

Definition 7. *A (conventional)* register machine *is a register machine over $(\mathbb{Z}, +)$ with the following two types of instructions:*

1. $(ADD(i), q, s)(a_1, \ldots, a_i, \ldots, a_n) = \big((a_1, \ldots, a_i + 1, \ldots, a_n), \{q, s\}\big)$, *and*
2. $(SUB(i), q, z)(a_1, \ldots, a_i, \ldots, a_n) =$
$$\begin{cases} \big((a_1, \ldots, a_i - 1, \ldots, a_n), \{q\}\big), & \text{if } a_i > 0, \\ \big((a_1, \ldots, a_i, \ldots, a_n), \{z\}\big), & \text{if } a_i = 0. \end{cases}$$

Computations of conventional register machines are defined as computations of the corresponding register machines over $(\mathbb{Z}, +)$ with the restriction that, in the initial configuration, all registers must contain non-negative values.

It follows from the form of instructions allowed in conventional register machines that their registers contain non-negative values at any time. Therefore, one

can see such register machines as an even more powerful form of partially blind register machines (and thus a particular case of blind register machines), in which the machine is allowed to check whether any given register is zero.

We would like to remark that by considering other types of instructions or restrictions on the class of valid computations, one can characterise many other variants of register machines. For example, reversal-bounded counter automata are register machines in which one can only switch from incrementing to decrementing a register (and conversely) a bounded number of times [9].

4 Integer Vector Addition P Systems

In the article [10], Gheorghe Păun suggested exploring multisets with negative multiplicities. Several possible answers were suggested. In [3], the authors define generalised multisets as having multiplicities from totally ordered Abelian groups. The work [1] takes a different approach and partitions the alphabet of objects into two categories: the regular objects, which may have any integer multiplicity, and the so-called "catalysts", which are only allowed to appear in a bounded number of copies. Like in purely catalytic P systems, the "catalysts" in this model are used to guide the applicability of rules.

In this work, we generalise this model to the concept of integer vector addition P systems. Before defining this model, we define the following natural extension of multisets.

Definition 8. *A* $\mathbb{Z}$-multiset *over the (finite) alphabet* O *is a mapping* $w : O \to \mathbb{Z}$. *The value* $w(a)$ *is called the* multiplicity *of* a *in* w. *An object* $a \in O$ *is said to* apear in w *if* $w(a) \neq 0$. *A multiset* w *is said to be* empty *if no objects appear in it.*

Thus, $\mathbb{Z}$-multisets can also be seen as *vectors* of integers, indexed by elements of O. We will use the notation $\mathbb{Z}^O$ to refer to the set of all $\mathbb{Z}$-multisets over O.

Definition 9. *An* integer vector addition P system *($\mathbb{Z}$-VAPS) is the construct*

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

where O *is a finite alphabet of objects,* $T \subseteq O$ *is the set of terminal objects,* μ *is the membrane structure injectively labelled by the numbers from* $\{1, \ldots, n\}$ *and usually given by a sequence of correctly nested brackets,* w_i *are the* $\mathbb{Z}$*-multisets giving the initial contents of every membrane* i, $1 \leq i \leq n$, R *is a finite set of rules of the form* $r : \{1, \ldots, n\} \to \mathbb{Z}^O$, *and* h_i *and* h_o *are the labels of the input and the output membranes, respectively (*$1 \leq h_i \leq n$, $1 \leq h_o \leq n$*).*

Thus, integer vector addition P systems manipulate vectors of integers, indexed by the objects from O ($\mathbb{Z}$-multisets). A rule $r \in R$ assigns such a vector to every membrane of Π; applying r means adding (by componentwise addition) the vector $r(i)$ to the vector representing the contents of the membrane i, for every $1 \leq$

$i \leq n$. Therefore, one may see r as only having a right-hand side and as being unconditionally applicable. Such a form comes in naturally, since, as also pointed out in [3, 1], considering multiplicities over $\mathbb{Z}$ renders the usual rule applicability conditions irrelevant. We also remark that this way of defining the rules generalises naturally to a tissue-like membrane structure, i.e. a membrane structure which is not required to be a tree, but can be an arbitrary graph (cf. [5]).

We will use the tuple notation to describe rules of vector addition P systems — a rule r will be given by the set $\big\{(i, r(i)) \mid 1 \leq i \leq n, r(i) \text{ is not empty}\big\}$.

In [1], the authors use a special symbol δ to command the dissolution of the membrane in which it is produced. To allow for the same possibility in vector addition P systems, we will define the rules as functions of the form $\{1, \ldots, n\} \to \mathbb{Z}^{O \cup \{\delta\}}$, i.e. as functions assigning $\mathbb{Z}$-multisets over $O \cup \{\delta\}$ to each membrane. If $r(i)(\delta) = 1$ for the elementary membrane i, the application of r will dissolve this membrane after adding the multiplicities of symbols different from δ to its contents. We may even allow $r(i)(\delta) = k > 1$, in which case k successive membranes in the hierarchy will be dissolved, but only the contents of the innermost dissolved membrane will be copied into the corresponding parent membrane (the contents of the intermediary membranes will be lost).

Allowing dissolution makes it possible to introduce a rule applicability condition: r is applicable if every membrane i, for which $r(i)$ is not empty, is still present in the system.

The integer vector addition P system Π evolves by *sequentially* applying rules from R until a halting configuration is reached. Remark that, because of the use of $\mathbb{Z}$-multisets, the only way to use the classical halting condition is to dissolve all the membranes to which the rules of Π may contribute. This corresponds to the approach proposed in [1] which consists in dissolving all the working membranes until the result reaches a membrane without any rules. Thus, the classical halting condition becomes somewhat degenerate; it is therefore only natural to discuss other halting conditions, for example:

- *unconditional halting* — the system may halt at any moment, independently of rule applicability or contents of the membranes;
- *halting by zero* — the system halts when it reaches a configuration in which all multisets representing the contents of all membranes are empty.

One may see unconditional halting as corresponding to the way in which the language generated by a grammar is defined [4]: essentially, the contents of the output membrane of Π in any configuration Π can reach, projected on the terminal alphabet T, is part of the vector language generated by Π. On the other hand, halting by zero corresponds to the way in which blind register machines recognise input vectors.

We will use the symbols *uncond*, *zero*, and *inappl* to refer to unconditional halting, halting by zero, and halting by inapplicability of rules. Similarly, we will use the symbols *acc* and *gen* to refer to the accepting and generating modes. We will use the notation $Ps_{\mathbb{Z}}VAPS(m, h)$, $m \in \{acc, gen\}$, $h \in \{uncond, zero, inappl\}$,

to refer to the class of sets of vectors of integers accepted or generated by integer vector addition P systems working with the corresponding halting conditions. We will add the symbol δ to refer to the vector languages associated with $\mathbb{Z}$-VAPS with dissolution rules ($Ps_{\mathbb{Z}}VAPS(m,h,\delta)$) and the symbol δ^* to refer to the languages of $\mathbb{Z}$-VAPS which are allowed to dissolve multiple membranes at a time ($Ps_{\mathbb{Z}}VAPS(m,h,\delta^*)$). Finally, we will replace $\mathbb{Z}$ by $\mathbb{N}$ to refer to the languages of vectors of naturals (non-negative integers).

We immediately observe that $Ps_{\mathbb{Z}}VAPS(m,inappl) = \{\varnothing\}$, because if a $\mathbb{Z}$-VAPS has any rules at all, it can never halt by rule inapplicability.

5 On the Power of Blind Register Machines

In this section, we will focus on relating integer vector addition systems to blind register machines, as well as on expressing the power of both models in terms of semilinear vectors of numbers. We will show that blind register machines and $\mathbb{Z}$-VASS generate exactly $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors.

The work [2] also discusses the computational power of blind and partially blind register machines, but it uses a different definition of blindness: a blind register machine is defined as a partially blind register machine which may halt with any values in the registers. In the present paper we use a definition which is closer to Sheila Greibach's blind and partially register machines [6].

We will start by giving a proof of the quite intuitive result that blind register machines recognise exactly the same sets of integer vectors as integer vector addition systems with states generate.

Theorem 1. *$Ps_{\mathbb{Z}}BRM = \mathbb{Z}$-VASS.*

Proof. Take a blind register machine $B = (n, \mathbb{Z}, Q, q_0, q_h, P)$; we will construct a $\mathbb{Z}$-VASS $\Gamma = (w_0, S, s_0, s_h, p, \delta)$ with $w_0 = (0, \ldots, 0) \in \mathbb{Z}^n$, $S = Q$, $s_0 = q_h$, $s_h = q_0$. The set $\delta(p)$ will contains all the states of B from which p can be reached:

$$\begin{aligned}\delta(s) \;=\; \{q \in Q \mid P(q) = (SUB(i), s) \text{ or } P(q) = (ADD(i), s, s')\\ \text{or } P(q) = (ADD(i), s', s)\}.\end{aligned}$$

The vector $p(s)$ associated with a state $s \in S$ does the opposite effect of the instruction associated with the same state in B:

$$p(s) \;=\; \begin{cases} \mathbf{1}_i, & \text{if } P(s) = (SUB(i), q),\\ -\mathbf{1}_i, & \text{if } P(S) = (ADD(i), q, q'),\end{cases}$$

where $\mathbf{1}_i \in \mathbb{Z}^n$ is a vector whose only non-zero component is the i-th component.

It follows from the construction of Γ that, for every computation of B accepting an input vector $\mathbf{x}$, there exists a computation of Γ halting on the same vector, and conversely, which proves that $Ps_{\mathbb{Z}}BRM \subseteq \mathbb{Z}$-VASS.

To prove the converse inclusion, it suffices to take an arbitrary integer vector addition system and construct a blind register machine by reversing the arrows in the state control graph and by simulating the inverse effect of the addition vectors using multiple states.

The same construction can be used to show that partially blind register machine are equivalent in power to conventional vector addition systems with states. Taking into consideration the result on equivalence between (conventional) VAS and VAS with states from [8], we formulate the following characterisation of the power of partially blind register machines.

Theorem 2. $PsPBRM = VASS = VAS$.

We will now show that blind register machines do not recognise more than $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Lemma 1. $Ps_{\mathbb{Z}}BRM \subseteq \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Proof. Consider a blind n-register machine B. At every step, B can increment or decrement a register, *independently* of the contents of the registers. Consider the alphabet of actions of B: $\mathcal{A}_B = \{ADD(i), SUB(i) \mid 1 \leq i \leq n\}$; every computation of B can be represented as a string over this alphabet. Let $valid(A_B) \subseteq A_B^*$ be the strings over A_B^* which correspond to all computations of B. Pick such a string $w \in valid(A_B)$. Since the actions do not depend on the contents of the registers, any permutation of w which is in $valid(A_B)$ will have the same effect as w. In particular, B will halt with the same values in its registers. Therefore, the set of vectors B recognises can be described as follows:

$$N(B) = \left\{-(a_1, \ldots, a_n) \mid w \in valid(A_B), a_i = |w|_{ADD(i)} - |w|_{SUB(i)}\right\},$$

where $|w|_x$ is the number of copies of the symbol $x \in A_B$ in the string w.

Because B cannot read the values of its registers, the set $valid(A_B)$ is the regular language given by the state control of B. Therefore, the Parikh image $Ps\big(valid(A_B)\big)$ is an $\mathbb{N}$-semilinear set. This means that $N(B)$ is an $\mathbb{N}$-semilinear set of $\mathbb{Z}$-vectors, and so is the set of vectors including only the values of the output registers of B. Consequently, $Ps_{\mathbb{Z}}BRM \subseteq \mathbb{Z}^*SLIN_{\mathbb{N}}$, which is the statement of the lemma.

We will now show that blind register machines can recognise all $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Lemma 2. $Ps_{\mathbb{Z}}BRM \supseteq \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Proof. Consider an $\mathbb{N}$-semilinear set A of $\mathbb{Z}$-vectors. There exists a finite collection of sets of generators $A_i \subseteq \mathbb{Z}^n$ and offsets $\mathbf{a}_i \in \mathbb{Z}^n$ such that $A = \bigcup_i \langle A_i, \mathbf{a}_i \rangle_{\mathbb{Z}}$. Consider a blind register machine B which starts by non-deterministically choosing a set of generators A_i and the corresponding offset $\mathbf{a}_i$. B then repeats the following procedure until the set A_i is exhausted:

1. remove a generator $\mathbf{a}$ from A_i;
2. subtract $\mathbf{a}$ from the vector describing the registers of B a number of times chosen non-deterministically.

At the end, B subtracts the vector $\mathbf{a}_i$ from its registers. If B manages to reset all its registers using this procedure, then, by construction, the input vector belongs to $\langle A_i, \mathbf{a}_i \rangle_{\mathbb{Z}} \subseteq A$ (and the computation of the machine gives a way to construct this vector from A_i and $\mathbf{a}_i$). This implies the statement of the lemma.

It follows from Lemmas 1 and 2 that blind register machines recognise exactly the $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Theorem 3. $Ps_{\mathbb{Z}}BRM = \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Consequently, if one takes only the natural vectors recognised by blind register machines, one obtains positive-restricted $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Corollary 1. $Ps_{\mathbb{N}}BRM = \mathbb{Z}^*_+SLIN_{\mathbb{N}}$.

6 On the Power of $\mathbb{Z}$-VA(P)S

In this section we will describe the power of integer vector addition (P) systems in terms of semilinear sets of vectors and blind register machines. We will start by pointing out that $\mathbb{Z}$-VAPS without dissolution and with unconditional halting generate exactly the sets reachable by $\mathbb{Z}$-VAS.

Lemma 3. $Ps_{\mathbb{Z}}VAPS(gen, uncond) = \mathbb{Z}\text{-}VAS$.

Proof. The effect of rules of $\mathbb{Z}$-VAPS without dissolution does not depend on the contents of the membranes. Consider the set of rules R of such a P system Π; we will construct a $\mathbb{Z}$-VAS Γ whose starting vector is the initial contents of the output membrane h_o of Π, and whose addition vectors are given by the projection $\{r(h_o) \mid r \in R\}$. Since Π can halt at any moment, its output is exactly the reachable vectors of Γ. Therefore $Ps_{\mathbb{Z}}VAPS(gen, uncond) \subseteq \mathbb{Z}\text{-}VAS$.

The converse inclusions follows from the fact that any $\mathbb{Z}$-VAS can be seen as a one-membrane $\mathbb{Z}$-VAPS working with unconditional halting.

The same kind of reasoning allows us to characterise the power of $\mathbb{Z}$-VAPS working in recognising mode and halting by reaching zero vectors in all membranes.

Lemma 4. $Ps_{\mathbb{Z}}VAPS(acc, zero) = \mathbb{Z}\text{-}VAS$.

On the other hand, because of the direct equivalence between $\mathbb{Z}$-VAS and $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors, we can write the previous two results in the following way.

Theorem 4. $Ps_{\mathbb{Z}}VAPS(gen, uncond) = Ps_{\mathbb{Z}}VAPS(acc, zero) = Z^*LIN_{\mathbb{N}} = \mathbb{Z}\text{-}VAS$.

Allowing membrane dissolution together with unconditional halting allows generating at most unions of vector languages generated by families of $\mathbb{Z}$-VAS which may only differ in their start vectors. We will call such families *uniform* and will denote the class of vector languages generated by such families by $\mathbb{Z}$-$VAS_{\cup}$.

Lemma 5. $Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta) \subseteq \mathbb{Z}\text{-}VAS_{\cup}$.

Proof. Consider a $\mathbb{Z}$-VAPS Π with normal membrane dissolution and halting by inapplicability of rules. First of all, we remark that the contents of the output membrane h_o only depend on the evolution of the membranes located within. Furthermore, h_o must have no rules associated, otherwise the system will never halt (or will end up dissolving h_o if h_o is not the skin, in which case no output will be yielded either). Finally, only those inner membranes of h_o which are dissolved contribute to its final contents.

Consider one of the membranes h located somewhere within h_o. If it has no inner membranes, its evolution is described by a $\mathbb{Z}$-VAS. If h has one inner membrane h' which is elementary (it contains no other membranes), then we can distinguish two phases in the evolution of h: before and after the dissolution of h' (and before the dissolution of h itself). Given that the contents of h' must eventually be merged with those of h, we can just as well consider that, during the first phase, the rules contributing to h are extended by the corresponding additions carried out by the rules contributing to h'. Since the order in which the rules of $\mathbb{Z}$-VAPS are applied does not affect the result, we can consider that h contains *no inner membranes* at all, but possesses a double set of contributing rules instead: one which combines the original rules contributing to h and to h', and another which only includes the original rules contributing to h. Therefore, we can correctly describe the evolution of h by taking at least some of the vectors a $\mathbb{Z}$-VAS can reach.

We remark that the rule dissolving h' in this case may only be applied once, and its effect can be simulated by adding the vector it produces to the starting multiset of the containing membrane h.

The reasoning from the previous paragraphs can be applied to a membrane which contains multiple elementary membranes, as well as, inductively, to all inner membranes of the output membrane h_o: we can replace h_o by a new elementary membrane h'_o, and take some of the vectors generated in it into the output language. Remark now that the moment at which a membrane is dissolved is not correlated with its contents and only depends on whether all of its inner membranes have been dissolved already. This means that, if the depth of the membrane structure contained in h_o is d, d steps of evolution are necessary and suffice for dissolving all the inner membranes of h_o. Therefore, the contents of the membrane h_o in the halting configurations of Π are given by all the vectors that can be reached in membrane h'_o in at least d steps. These are the vectors which can be reached by the family of $\mathbb{Z}$-VAS $\{(\mathbf{w}_i, W) \mid \mathbf{w}_i \in W_d\}$, where W contains the addition vectors defined by the rules contributing to the new membrane h'_o, and W_d is the (finite) set of vectors which h'_o can reach in exactly d steps.

It turns out that this family of $\mathbb{Z}$-VAPS can generate all vector languages from $\mathbb{Z}$-$VAS_{\cup}$.

Lemma 6. $Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta) \supseteq \mathbb{Z}$-$VAS_{\cup}$.

Proof. Consider a finite family of $\mathbb{Z}$-VAS $F = \{(\mathbf{w}_i, W) \mid 1 \leq i \leq n\}$. We will define a $\mathbb{Z}$-VAPS Π generating the vectors reachable by the systems from F in the following way. Π will have two nested membranes and two groups of rules. The first group of rules will apply the vectors from W to the inner membrane. A rule of the second group will add one of the vectors $\mathbf{w}_i$ to the inner membrane and dissolve it immediately. By construction, the vectors appearing in the halting configurations of Π are exactly the vectors which can be reached by the $\mathbb{Z}$-VAS from F, which proves the lemma.

The following theorem summarises the two preceding lemmas.

Theorem 5. $Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta) = \mathbb{Z}$-$VAS_{\cup}$.

Interestingly, the class $\mathbb{Z}$-$VAS_{\cup}$ is strictly in between the classes $\mathbb{Z}$-VAS and $\mathbb{Z}$-$VASS$.

Lemma 7. $\mathbb{Z}$-$VAS \subsetneq \mathbb{Z}$-$VAS_{\cup}$.

Proof. The inclusion is trivial. Consider now two $\mathbb{Z}$-VAS having the axioms $(0, 0)$ and $(0, 1)$, and sharing the only addition vector $(1, 1)$. The language of vectors reachable by these two systems is $L = \{(a, a), (a, a + 1) \mid a \in \mathbb{N}\}$. Suppose there exists a $\mathbb{Z}$-VAS Γ generating the same vector language L. In order to generate all pairs of natural numbers (a, a), it must start with the axiom $(0, 0)$ and have an addition vector of the form $(1, 1)$. Then, in order to generate the pairs $(a, a + 1)$, Γ needs to have an addition vector of the form $(x, x + 1)$. However, applying this addition vector twice yields the vector $(2x, 2x + 2) \notin L$, which contradicts the supposition and proves that the inclusion from the statement of the lemma is strict.

The following lemma describes the relationship between $\mathbb{Z}$-$VAS_{\cup}$ and $\mathbb{Z}$-$VASS$.

Lemma 8. $\mathbb{Z}$-$VAS_{\cup} \subsetneq \mathbb{Z}$-$VASS$.

Proof. The work of a finite family F of $\mathbb{Z}$-VAS can be simulated by a $\mathbb{Z}$-VASS by non-deterministically choosing a state in which one of the start vectors of F will be added, and by subsequent direct simulation of the application of the shared addition vectors.

Consider now the $\mathbb{Z}$-VASS Γ with the starting vector $(0, 0)$, which applies the addition vector $(0, 0)$ in the starting state q_0 and the non-deterministically chooses between $q_{(1,0)}$ and $q_{(0,1)}$. In $q_{(1,0)}$, Γ may apply the addition vector $(1, 0)$ indefinitely, before transitioning into q_h. Similarly, in $q_{(0,1)}$, Γ may apply the addition vector $(0, 1)$ indefinitely, before moving into q_h. Thus, the vector language generated by Γ is $L = \{(a, 0), (0, a) \mid a \in \mathbb{N}\}$.

Suppose there exists a family of $\mathbb{Z}$-VAS which generate the same language. The shared addition vectors of this family must therefore include both $(1,0)$ and $(0,1)$. But then, this family must also generate vectors in which both components are non-zero and which therefore do not belong to L. This contradicts our supposition and proves that the inclusion in the statement of the lemma is strict.

The previous lemma also gives an example of a $\mathbb{Z}$-semilinear set which cannot be generated by uniform family of $\mathbb{Z}$-VAS systems, which implies the following result.

Corollary 2. $\mathbb{Z}\text{-}VAS_{\cup} \subsetneq \mathbb{Z}^*SLIN_{\mathbb{N}}$.

It follows from the Theorem 5, Lemmas 7 and 8, as well as from the characterisations from the previous section, that the languages recognised by $\mathbb{Z}$-VAPS with normal dissolution and conventional halting are situated strictly in between $\mathbb{N}$-linear sets of $\mathbb{Z}$-vectors and $\mathbb{N}$-semilinear sets of $\mathbb{Z}$-vectors.

Theorem 6. $\mathbb{Z}^*LIN_{\mathbb{N}} \subsetneq Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta) \subsetneq \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Finally, we show that allowing dissolution of *multiple* membranes by one rule allows generating all $\mathbb{Z}$-semilinear languages and therefore renders such $\mathbb{Z}$-VAPS equivalent in power to blind register machines.

Theorem 7. $Ps_{\mathbb{Z}}VAPS(gen, inappl, \delta^*) = \mathbb{Z}^*SLIN_{\mathbb{N}}$.

Proof (Sketch). Consider a family F of n $\mathbb{Z}$-VAS, each of which generates a $\mathbb{Z}$-linear set of vectors. One can construct an integer vector addition P system Π with multiple dissolution in the following way. Π will have $n+2$ membranes organised in a linear structure. The rules of Π will simulate the i-th $\mathbb{Z}$-VAS in the membrane at depth $i+1$ (the depth of the skin is 0); moreover, Π will have a rule producing the start vector $\mathbf{w}_i$ and introducing $n-i$ copies of δ into the membrane at depth $i+1$, for $1 \leq i \leq n$. These rule effectively finalise the simulation of the i-th $\mathbb{Z}$-VAS. Finally, Π will have rules introducing i copies of δ into the innermost membrane, for $1 \leq i \leq n$, which will "select" the membrane at depth $i+1$ and will allow it to eventually apply its dissolution rules and put the result into the skin. Thus, Π generates the semilinear language generated by the family F.

To prove the inverse inclusion, we will rely on Lemma 1. Consider a $\mathbb{Z}$-VAPS Π with multiple dissolution. We will construct a blind register machine B which recognises the vector language generated by Π in the following way. B will have a group of working register per membrane of Π which will represent the multiplicities of the symbols in this membrane. B will start with the vector $\mathbf{x}$ in its input registers, and will simulate the applications of rules of Π in its working registers. Whenever Π dissolves a membrane (or multiple membranes), B will nondeterministically guess the multiplicities of each symbol in the dissolved membrane and will copy the guessed values into the working registers representing the corresponding parent membrane. When all inner membranes of the output membranes have been dissolved (B can encode the information about the membrane structure

in its state), B will simultaneously decrement the working registers representing the contents of the skin and the input registers. If, earlier during the simulation, B had guessed the value of a register wrongly, or, at the end of the simulation, the values of the input registers and the working registers representing the skin did not match, some registers of B will be zero and the vector $\mathbf{x}$ will be rejected. It follows from the construction that B will accept exactly the vector generated by Π, which implies the statement of the theorem.

7 Conclusion

In this paper we continued the investigation of P systems with multisets with integer multiplicities, proposed in [10] and already studied in [3] and [1]. We focused on the model originally described in [1] and generalised it to integer vector addition P systems, in which the applicability of rules does not in any way depend on the contents of the membranes. Interestingly enough, this P system variant exhibits very strong connection with blind register machines and integer vector addition systems — two models which have received little to no attention in the scientific literature up to now.

We studied a number of working modes of and halting conditions for integer vector addition P systems and gave exact characterisations of the power of the corresponding variants in terms of linear and semilinear sets over $\mathbb{Z}$ and over $\mathbb{N}$. We also pointed out a number of relations between the classes of languages generated or accepted by the model.

Some non-trivial open questions are revealed by our research. One of them concerns the semantics of multiple dissolution. In P systems, dissolution typically concerns one membrane at a time; in the present paper we suggest considering the possibility of dissolving multiple containing membranes in one step. The semantics we propose discards the contents of the dissolved intermediary membranes, so only the multiset of the innermost dissolved membrane is transferred to the corresponding parent membrane. Other semantics of multiple dissolution may be possible and are certainly worth exploring.

A very interesting open question concerns the types of semilinear sets. In this paper we only deal with semilinear sets with generators and initial offsets from $\mathbb{N}^n$ and $\mathbb{Z}^n$, restricted to non-negative values or not. It is however possible to consider the generators, the offsets, and the *coefficients* to belong to $\mathbb{N}^n$ or $\mathbb{Z}^n$, alternatively. This yields 8 possibly different kinds of semilinear sets, not including restrictions to non-negative values. Exploring the relations between these kinds of semilinear sets may be useful in further refining certain characterisations.

Finally, we point out that classical halting by inapplicability of rules is not necessarily well adapted for dealing with generalisations of multisets to integers. We give examples of different halting conditions inspired by other models of computing, but our list is far from exhaustive and is definitely worth to be extended.

References

1. O. Belingheri, A. E. Porreca, and C. Zandron. P systems with hybrid sets, 2016. Workshop on Membrane Computing, submitted.
2. R. Freund, O. Ibarra, Gh. Păun, and H.-C. Yen. Matrix languages, register machines, vector addition systems. *Third Brainstorming Week on Membrane Computing*, pages 155–167, 2005.
3. R. Freund, S. Ivanov, and S. Verlan. P systems with generalized multisets over totally ordered abelian groups. In *Int. Conf. on Membrane Computing*, volume 9504 of *Lecture Notes in Computer Science*, pages 117–136. Springer, 2015.
4. R. Freund, M. Kogler, and M. Oswald. A general framework for regulated rewriting based on the applicability of rules. In J. Kelemen and A. Kelemenová, editors, *Computation, Cooperation, and Life*, volume 6610 of *Lecture Notes in Computer Science*, pages 35–53. Springer Berlin Heidelberg, 2011.
5. R. Freund and S. Verlan. A formal framework for static (tissue) P systems. In G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer Berlin Heidelberg, 2007.
6. S. A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3):311–324, 1978.
7. C. Haase and S. Halfon. Integer vector addition systems with states. In J. Ouaknine, I. Potapov, and J. Worrell, editors, *Reachability Problems: 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, pages 112–124. Springer, 2014.
8. J. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979.
9. O. H. Ibarra. Automata with reversal-bounded counters: A survey. In H. Jürgensen, J. Karhumäki, and A. Okhotin, editors, *Descriptional Complexity of Formal Systems: 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*, pages 5–22. Springer, 2014.
10. Gh. Păun. Some quick research topics. `http://www.gcn.us.es/files/OpenProblems_bwmc15.pdf`.
11. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
12. Gh. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.

Extended SNP Systems with States

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

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md
[2] Faculty of Informatics, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria
E-mail: rudi@emcc.at
[3] Université Paris Est, France
E-mail: sergiu.ivanov@u-pec.fr

Summary. We consider (extended) spiking neural P systems with states, where the applicability of rules in a neuron not only depends on the presence of sufficiently many spikes (yet in contrast to the standard definition, no regular checking sets are used), but also on the current state of the neuron. Moreover, a spiking rule not only sends spikes, but also state information to the connected neurons. We prove that this variant of the original model of extended spiking neural P systems can simulate register machines with only two states, even in the basic non-extended variant.

1 Introduction

In the area of P systems, the model of *spiking neural P systems* was introduced in [7]. Whereas the basic model of membrane systems, see [11], reflects hierarchical membrane structures, in spiking neural P systems the cells are arranged in a tissue-like manner, with the contents of a cell (neuron) consisting of a number of so-called *spikes*, i.e., of a multiset over a single object. The rules assigned to a neuron allow us to send information to other neurons in the form of electrical impulses (also called spikes) which are summed up at the target neuron; the application of the rules depends on the contents of the neuron and in the general case is described by regular sets. As inspired from biology, the neuron sending out spikes may be "closed" for a specific time period corresponding to the refraction period of a neuron; during this refraction period, the neuron is closed for new input and cannot get excited ("fire") for spiking again.

The length of the axon may also cause a time delay before a spike arrives at the target. Moreover, the spikes coming along different axons may cause effects of different magnitude. All these biologically motivated features were included in the model of extended spiking neural P systems considered in [3], the most

important theoretical feature being that neurons can send spikes along the axons with different magnitudes at different moments of time.

In this paper, we consider a variant of the model of extended spiking neural P systems which not only uses spikes to be sent to other neurons when some neuron spikes, but also allows for sending some additional information called "state" along the axons. All these state informations arriving in a neuron then determine the next state of the neuron. On the other hand, we do not use the regular checking sets for the current number of spikes in the neuron any more, which decreases the amount of information a spiking rule may use. Hence, the spiking rules now depend on the current states of the neurons and the availability of sufficiently many spikes. This variant of extended spiking neural P systems with states has been inspired by the variant of spiking neural P systems with polarizations, see [16], where the states are called polarizations, and the underlying model of extended spiking neural P systems was the basic one with a fixed connection structure, only extended by allowing more than one spike to be sent along the axons. There it was shown that computational completeness (i.e., simulation of register machines) can be obtained with only three polarizations. In this paper we now show that computational completeness can already be obtained with only two states, i.e., with two polarizations, even for the basic non-extended variant as considered in [16], which solves an open problem raised at the Brainstorming Week on Membrane Computing in Sevilla at the beginning of February 2016.

The rest of the paper is organized as follows: In the next section, we recall some preliminary notions and definitions from formal language theory, especially the definition and some well-known results for register machines. In Section 3 we recall the definitions of the extended model of spiking neural P systems as considered in [3] and then define the model of *spiking neural P systems with states* as considered in this paper. In Section 4, we prove our main result and show that spiking neural P systems with only two states (0 and 1) can simulate register machines; the complexity of the construction depends on the features we require the spiking neural P systems to have, but the result even holds true for the basic non-extended variant of spiking neural P systems with states, where the connection structure between the neurons is fixed and does not depend on the spiking rules applied in the neurons. Moreover, we can use a very simple global state composition function computing the new state of a neuron from the state information having arrived from the input neurons in the simplest way by going to the "activated state 1" if and only if at least one such activating signal 1 has come in the previous step. In Section 5, we show how small universal spiking neural P systems with states can be constructed based on the results obtained in this paper. A short summary of the results we obtained concludes the paper.

2 Preliminaries

In this section we recall the basic elements of formal language theory and especially the definitions and results for register machines; we also refer to the corresponding

section from [3] and [2]. For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [14]. We just list a few notions and notations:

V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; for any $w \in V^*$, $|w|$ denotes the number of symbols in w (the *length* of w). $\mathbb{N}_+$ denotes the set of positive integers (natural numbers), $\mathbb{N}$ is the set of non-negative integers, i.e., $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$.

2.1 Register Machines

The proofs of the results establishing computational completeness in the area of P systems are often based on the simulation of register machines; we refer to [9] for original definitions, and to [6] for the definitions we use in this paper:

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

- $p : (\texttt{ADD}(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq j \leq n$.
  Increases the value of register r by one, followed by a non-deterministic jump to instruction q or s. This instruction is usually called *increment*.
- $p : (\texttt{SUB}(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq j \leq n$.
  If the value of register r is zero then jump to instruction s; otherwise, the value of register r is decreased by one, followed by a jump to instruction q. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : halt$ (HALT instruction)
  Stop the machine. The final label l_h is only assigned to this instruction.

A (non-deterministic) register machine M is said to generate a vector $(s_1, \cdots, s_\beta)$ of natural numbers if, starting with the instruction with label l_0 and all registers containing the number 0, the machine stops (it reaches the instruction $l_h : halt$) with the first β registers containing the numbers $s_1, \cdots, s_\beta$ (and all other registers being empty).

The register machines are known to be computationally complete, equal in power to (non-deterministic) Turing machines: they generate or accept exactly the sets of vectors of non-negative integers which can be generated by Turing machines.

3 Extended Spiking Neural P Systems

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [10] and [12]; comprehensive information can be found on the P systems

web page [15]. Moreover, for the motivation and the biological background of spiking neural P systems we refer the reader to [7] as well as to the corresponding Chapter 13 in the Handbook of Membrane Computing [12]. For the definition of an *extended spiking neural P system* we refer to [3].

We now extend the model of *extended spiking neural P systems* to the new model of *extended spiking neural P systems with states*, i.e., the neurons can be in different states, and depending on the current state of a neuron, different spiking rules may be applicable.

Definition 1. *An* extended spiking neural P system with states *(of degree $m \geq 1$) (an* ESNPS system *for short) is a construct $\Pi = (N, S, I, R, f)$ where*

- *N is the set of* cells *(or* neurons*); the neurons may be uniquely identified by a number between* 1 *and m or by an alphabet of m symbols;*
- *S is the set of* states*;*
- *I describes the* initial configuration *by assigning an initial value (of spikes) and an* initial state *to each neuron; for the sake of simplicity, we assume that at the beginning of a computation we have no pending packages along the axons between the neurons;*
- *R is a finite set of* rules *of the form $\left(i, s_i : E/a^k \to P; d\right)$ such that $i \in \{1, \cdots, m\}$ (specifying that this rule is assigned to neuron i), $s_i \in S$ is the current state of neuron i, $E \subseteq REG(\{a\})$ is the* checking set *(the current number of spikes in the neuron has to be from E if this rule shall be executed), $k \in \mathbb{N}$ is the "number of spikes" (the energy) consumed by this rule, d is the* delay *(the "refraction time" when neuron i performs this rule), and P is a (possibly empty) set of* productions *of the form (l, w, s, t) where $l \in [1..m]$ (thus specifying the target neuron), $w \in \{a\}^*$ is the* weight *of the energy sent along the axon from neuron i to neuron l, $s \in S$ is the* state *sent along the axon from neuron i to neuron l, and t is the time needed before the information sent from neuron i arrives at neuron l (i.e., the* delay along the axon*);*
- *f is the state composition function, which for each neuron allows for computing the new state of a neuron from its current state and the state signals having arrived in the neuron in the previous step.*

Definition 2. *A* configuration *of the ESNPS system is described as follows:*

- *for each neuron, the actual number of spikes in the neuron is specified;*
- *in each neuron i, we may find an "activated rule" $\left(i, s_i, E/a^k \to P; d'\right)$ waiting to be executed where d' is the remaining time until the neuron spikes;*
- *in each axon to a neuron l, we may find pending packages of the form (l, w, s, t') where t' is the remaining time until $|w|$ spikes have to be added to neuron l provided it is not closed for input at the time this package arrives.*

A transition *from one configuration to another one now works as follows:*

- *for each neuron i, we first check whether we find an "activated rule" $\left(i, s_i, E/a^k \to P; d'\right)$ waiting to be executed; if $d' = 0$, then neuron i "spikes",*

i.e., for every production (l, w, s, t) *occurring in the set* P *we put the corresponding package* (l, w, s, t) *on the axon from neuron* i *to neuron* l*, and after that, we eliminate this "activated rule"* $\big(i, s_i, E/a^k \to P; 0\big)$*;*

- *for each neuron* l*, we now consider all packages* (l, w, t') *on axons leading to neuron* l*; provided the neuron is not closed, i.e., if it does not carry an activated rule* $\big(i, s_i, E/a^k \to P; d'\big)$ *with* $d' > 0$*, we then sum up all weights* w *in such packages where* $t' = 0$ *and add this sum of spikes to the corresponding number of spikes in neuron* l *as well as remember the corresponding state signals* s *for eventually computing the next state of the neuron; in any case, the packages with* $t' = 0$ *are eliminated from the axons, whereas for all packages with* $t' > 0$*, we decrement* t' *by one;*
- *for each neuron* i*, we now again check whether we find an "activated rule"* $\big(i, s_i, E/a^k \to P; d'\big)$ *(with* $d' > 0$*) or not; if we have not found an "activated rule", we now compute the new state of the neuron by using the state composition function for the underlying neuron* i*. Then we may apply any rule* $\big(i, s_i, E/a^k \to P; d\big)$ *from* R *for which neuron* i *is in state* s_i *and the current number of spikes in the neuron is in* E*, and then put a copy of this rule as "activated rule" for this neuron into the description of the current configuration; on the other hand, if there still has been an "activated rule"* $\big(i, s_i, E/a^k \to P; d'\big)$ *in the neuron with* $d' > 0$*, then we replace* d' *by* $d' - 1$ *and keep* $\big(i, s_i, E/a^k \to P; d' - 1\big)$ *as the "activated rule" in neuron* i *in the description of the configuration for the next step of the computation.*

After having executed all the substeps described above in the correct sequence, we obtain the description of the new configuration. A computation *is a sequence of configurations starting with the initial configuration given by* I*. A computation is called* successful *if it halts, i.e., if no pending package can be found along any axon, no neuron contains an activated rule, for no neuron, a rule can be activated, and no neuron would change its state in the next step (thus making other spiking rules applicable).*

In the original model introduced in [7], we have only one state, so we can omit the states in the description of spiking rules in this paragraph. In the productions (l, w, t) of a rule $\big(i, E/a^k \to \{(j, w_j, t_j) \mid j \in J\}; d\big)$, only $w = a$ (for *spiking rules*) or $w = \lambda$ (for *forgetting rules*) as well as $t = 0$ was allowed (and for forgetting rules, the checking set E had to be finite and disjoint from all other sets E in rules assigned to neuron i). Moreover, reflexive axons, i.e., leading from neuron i to neuron i, were not allowed, hence, for (l, w, t) being a production in a rule $\big(i, E/a^k \to P; d\big)$ for neuron i, $l \neq i$ was required. Yet the most important extension was that different rules for neuron i may affect different axons leaving from it whereas in the original model the structure of the axons (called synapses there) was fixed. In [3], the sequence of substeps leading from one configuration to the next one together with the interpretation of the rules from R was taken in such a way that the original model can be interpreted in a consistent way within the extended model introduced in that paper. As mentioned in [3], from a mathematical point

of view, another interpretation would have been even more suitable: whenever a rule $(i, E/a^k \to P; d)$ is activated, the packages induced by the productions (l, w, t) in the set P of a rule $(i, E/a^k \to P; d)$ activated in a computation step are immediately put on the axon from neuron i to neuron l, whereas the delay d only indicates the refraction time for neuron i itself, i.e., the time period this neuron will be closed. The delay t in productions (l, w, t) can be used to replace the delay in the neurons themselves in many of the constructions elaborated, for example, in [7], [13], and [4]. Yet as in (the proofs of computational completeness given in) [3], we shall not need any of the delay features in this paper, hence we need not go into the details of these variants of interpreting the delays in more details. Finally, we mention that as in [5], the notion of *extended* spiking neural P systems often is used only taking into account that more than one spike can be sent along all axons with one spiking rule.

Depending on the purpose the ESNP(S) system is to be used, some more features have to be specified: for generating k-dimensional vectors of non-negative integers, we have to designate k neurons as *output neurons*; the other neurons then will also be called *actor neurons*. As in [3], also in this paper, we take the number of spikes at the end of a successful computation in the neuron as the output value.

Definition 3. *Obviously, extending an already very powerful model with an additional feature (states) yields a model which is at least as powerful, even with respect to descriptional complexity. Hence, besides completely omitting delays, as in [16] we also omit the checking sets (in fact, this means taking all checking sets to be* $\{a\}^+$*). Thus, for a spiking rule* $(i, s_i : E/a^k \to P; d)$ *we write* $i : (s_i, a^k) \to P$*. Moreover, the set* P *will not be written as a set, but just by concatenating its elements of the form* (l, w, s)*, where* l *is the target neuron,* w *describes the number of spikes sent to* l *and* s *is the state signal sent to* l*.*

The following example illustrates the computational power of ESNPS systems with two states by showing how exponential number languages can be generated.

Example 1. We will construct the ESNPS system

$$\Pi_{4^n} = (\{\sigma_1, \sigma_1', \sigma_2, \sigma_2'\}, \{0, 1\}, I, R, f)$$

generating the multiset language $\left\{a^{4^n} \mid n > 1\right\}$ in the output neuron σ_1. Initially, σ_1 and σ_1' are in state 1 and contain one and two spikes, respectively. On the other hand, neurons σ_2 and σ_2' initially are in state 0 and are empty.

The state composition function f, for every neuron, is given as follows: If any state signal 1 has arrived in the previous step, the state of the neuron is 1, and 0 otherwise.

To illustrate the rules in the ESNPS system, we use the following graphical notation: Each rule is represented by an arrow with a single tail but with multiple

heads; the branching point is highlighted by a black bullet. The left-hand side of the rule is written on the segment preceding the bullet and each right-hand side on the corresponding arrow head. When writing the right-hand sides, we omit the names of the target neurons (because they are pointed at by the arrow heads).

Π_{4^n} works in a two-phase cycle. In the first phase, all the spikes from σ_1 are transferred in two copies into σ_2; this phase is controlled by σ'_1. The second phase is symmetric: the spikes from σ_2 are doubled and moved into σ_1, under the control of σ'_2.

The first phase is governed by the following rules in neurons σ_1 and σ'_1:

$$\begin{aligned}
&\sigma_1 : (1, a) \to \left(\sigma_2, a^2, 0\right) (\sigma'_1, a, 1)\,,\\
&\sigma'_1 : (1, a) \to (\sigma_1, \lambda, 1)\,,\\
&\sigma'_1 : (0, a) \to (\sigma_2, \lambda, 1) \left(\sigma'_2, a^2, 1\right).
\end{aligned}$$

The graphical representation of these rules is given in Figure 1.

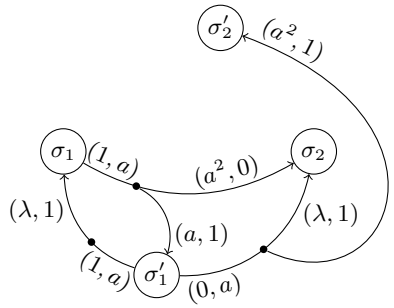


Fig. 1. Multiplication by 2 in ESNPS with two states.

The loop between σ_1 and σ'_1 ensures that, while there are still spikes in σ_1, both neurons stay in state 1. When there are no more spikes in σ_1, σ'_1 must pass into state 0 and will have to use its last spike (of the two it normally contains) to apply the rule $(0, a) \to (\sigma_2, \lambda, 1) \left(\sigma'_2, a^2, 1\right)$. This rule puts two spikes into the control neuron σ'_2 and switches both neurons σ_2 and σ'_2 to state 1, thereby starting the second phase of the cycle. The second phase is totally symmetric and is governed by the following rules in neurons σ_2 and σ'_2:

$$\begin{aligned}
&\sigma_2 : (1, a) \to \left(\sigma_1, a^2, 0\right) (\sigma'_2, a, 1)\,,\\
&\sigma'_2 : (1, a) \to (\sigma_2, \lambda, 1)\,,\\
&\sigma'_2 : (0, a) \to (\sigma_1, \lambda, 1) \left(\sigma'_1, a^2, 1\right).
\end{aligned}$$

Finally, to ensure that the system halts after the second phase of the cycle, we add the following rule to the control neuron σ'_2:

$$\sigma'_2 : (0, a) \to (\sigma_1, \lambda, 0)\,.$$

Thus, σ_2' may choose between restarting the loop by switching the state of σ_1 and σ_1', or just forgetting the last control spike, thus effectively bringing the system to a halt, with 4^n copies of a in σ_1, where n is the number of times the cycle has run.

4 Simulating Register Machines with Extended Spiking Neural P Systems with only Two States

We now consider an arbitrary *n-register machine* $M = (n, B, l_0, l_h, P)$ and provide a first simple proof how to simulate the computations of such a register machine by an extended spiking neural P system with only two states:

For all neurons, we use one global state composition function, as in the example, i.e., if any state signal 1 has arrived in the previous step, the state of the neuron is 1, and 0 otherwise.

$p : (\texttt{ADD}(r), q, s)$ is simulated by neuron p with the rules

p: $(0, a) \to (q, a, 0)(r, a, 0)$ and
p: $(0, a) \to (s, a, 0)(r, a, 0)$.

meaning that neuron p (always staying in state 0) consumes one spike and sends one spike and state 0 to neuron q or neuron s and to neuron r (the neuron representing register r). Figure 2 illustrates these rules graphically.

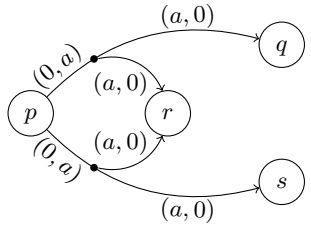


Fig. 2. Simulation of *ADD* relying on a flexible connection structure and on sending zero spikes.

The rule in register r allowing for simulating SUB-instructions is:
r: $(1, a) \to \prod_{l \in SUB(r)} (l'', \lambda, 1)$

In case the register is non-empty, in the activated state 1 one spike is eliminated and state 1 is sent to every neuron l'' for every label l of a SUB-instruction, yet no spike a is sent.

$p : (\texttt{SUB}(r), q, s)$ is simulated by neurons p, p', p'' with the rules

p: $(0, a) \to (p', a, 1)\,(r, \lambda, 1)$,
p': $(1, a) \to (p'', a, 0)$, as well as
p'': $(0, a) \to (s, a, 0)$ and

p'': $(1, a) \to (q, a, 0)$.

The rules are illustrated in Figure 3. The dashed arrow represents the family of right-hand sides broadcasting spikes from neuron r to neurons l'', $l \in SUB(r)$.

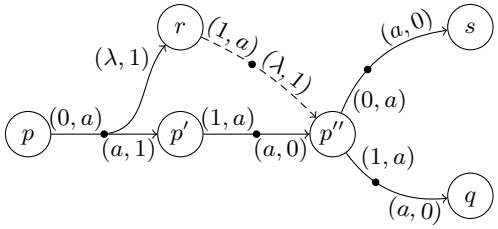


Fig. 3. Simulation of SUB relying on a flexible connection structure and on sending zero spikes.

The simple construction described above obeys to the following features:

- We only need two states!
- We do not use self-loops.
- We send the same state to all neurons in each rule!
- Exactly one spike is consumed by each rule!
- Wo do not use forgetting rules!
- But on the other hand, we allow to also send *zero* spikes to a neuron!
- The connection structure only depends on the state in case of deterministic register machines! Yet by using a more complicated construction for the simulation of non-deterministic ADD-instructions we can even obtain that feature in general:
  $p : (\texttt{ADD}(r), q, s)$ is simulated by neurons p, p', p'' with the rules
  p: $(0, a) \to (p', a, 0)\,(r, a, 0)$,
  p': $(0, a) \to (p'', a, 0)$ and
  p': $(0, a) \to (p'', a, 1)$, as well as
  p'': $(0, a) \to (q, a, 0)$ and
  p'': $(1, a) \to (s, a, 0)$.
  See Figure 4 for a graphical illustration of these rules.

For comparison with the model considered in [16] (where states are called *polarizations*) we have to ask the following question: *Can we have a completely static connection structure even not depending on the state of the neuron?*

We first show that a non-deterministic ADD-instruction can be simulated within a fixed connection structure, now using forgetting rules, yet also using the initial neuron p in the activated state 1:

$p : (\texttt{ADD}(r), q, s)$ is simulated by neurons p, p' with the rules

p: $(1, a) \to (p', a, 0)\,(r, a, 0)$,

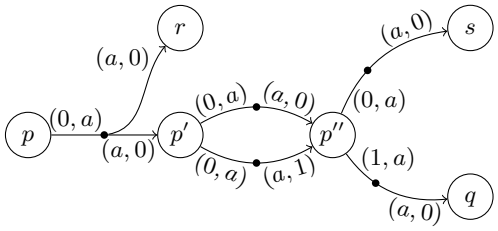


Fig. 4. Simulation of *ADD* using a connection structure which only depends on states.

p': $(0,a) \to \left(0''_q, a, 0\right)(1''_s, a, 0)$ and
p': $(0,a) \to \left(0''_q, a, 1\right)(1''_s, a, 1)$,
together with the following rules in the neurons $0''_l, 1''_l$, for every label $l \in B$:

$0''_l$: $(0,a) \to (l, a, 1)$ and
$0''_l$: $(1,a) \to \lambda$,
$1''_l$: $(1,a) \to (l, a, 1)$ and
$1''_l$: $(0,a) \to \lambda$.

Figure 5 gives a graphical illustration of these rules. Forgetting rules which only consume spikes without sending anything anywhere are depicted as dangling arrows.

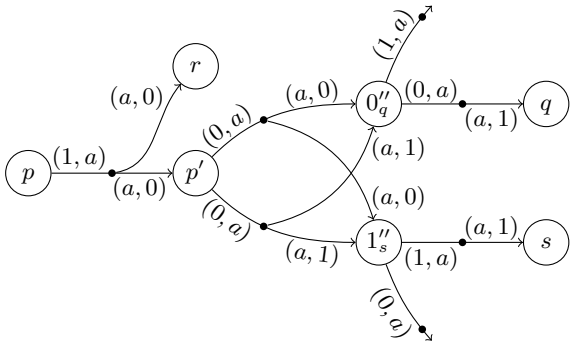


Fig. 5. Simulation of *ADD* using a fixed connection structure.

Instead of showing how SUB-instructions can also be simulated within a fixed connection structure, we also attack the last remaining non-standard feature at the same time, i.e.: *Can we avoid sending zero spikes?*

$p : (\texttt{SUB}(r), q, s)$ is simulated by the neurons $p, \tilde{p}, \tilde{p}', \hat{p}, \hat{p}', \hat{p}'', \bar{p}, \bar{p}'$ with the rules

p: $(1,a) \to (\tilde{p}, a, 1)(\hat{p}, a, 1)(r, a, 1)$,

$\tilde{p}$: $(1,a) \to (\tilde{p}',a,0)$,
$\tilde{p}'$: $\left(1,a^2\right) \to (q,a,1) \prod_{l \in SUB(r) \setminus \{p\}} \left(\hat{l}'',a,1\right)$,
$\tilde{p}'$: $(0,a) \to \lambda$ as well as

$\hat{p}$: $(1,a) \to (\hat{p}',a,1)$,
$\hat{p}'$: $(1,a) \to (\hat{p}'',a,0)$,
$\hat{p}''$: $(0,a) \to (r,a,1)\,(\bar{p},a,1)$,
$\hat{p}''$: $\left(1,a^2\right) \to \lambda$,
$\bar{p}$: $(1,a) \to (\bar{p}',a,1)$, and
$\bar{p}'$: $(1,a) \to (s,a,1) \prod_{l \in SUB(r)} \left(\hat{l}'',a,1\right)$

together with the following rules for the register neuron r and for the additional neuron r':

r: $\left(1,a^2\right) \to (r',a,1) \prod_{l \in SUB(r)} \left(\tilde{l}',a,1\right)$ and

r': $(1,a) \to \prod_{l \in SUB(r)} \left(\hat{l}'',a,1\right)$.

The rules are graphically illustrated in Figure 6. As before, dangling arrows represent rules sending nothing, while dashed arrows correspond to families of right-hand sides. For readability, we highlight neurons p, q, and s, which represent the states of the simulated machine. The dotted line going into neuron $\hat{p}''$ represents the family of right-hand sides which broadcast spikes and states to all neurons $\hat{l}''$, *except* neuron $\hat{p}''$ ($l \in SUB(r) \setminus \{p\}$). Finally, to avoid line intersections, the picture uses a "clone" of neuron r (the small grey r between $\hat{p}''$ and $\bar{p}$), which represents the same neuron r.

The main idea of this construction is to start both decrement and zero-check case in parallel and then, depending on the signal from r and r' take the necessary action, including to *reset* register r to the 0 if the additional spike sent there did not lead to a spiking action of neuron r in case the value stored in the register was zero. Moreover, all actor neurons affected by signals from neuron r not belonging to the current label p have to be reset without allowing them to act in a non-desired way:

For the neurons $\tilde{p}'$ this happens *automatically* as with only one spike a they cannot spike in state 1, yet after one step the state goes back to 0 and then allows the spike to be forgotten.

For the neurons $\hat{p}''$ this happens if the state signal 1 and the spike from neuron r' are accompanied by a second spike which allows for resetting the neuron by using the forgetting rule $\left(1,a^2\right) \to \lambda$.

5 Universal (Extended) Spiking Neural P Systems with Two States

We simulate the strongly universal register machine U_{22} of Korec, see [8]. Rather than performing a direct simulation which would yield $9 \times 1 + 8 \times 1 + 13 \times 4 = 69$

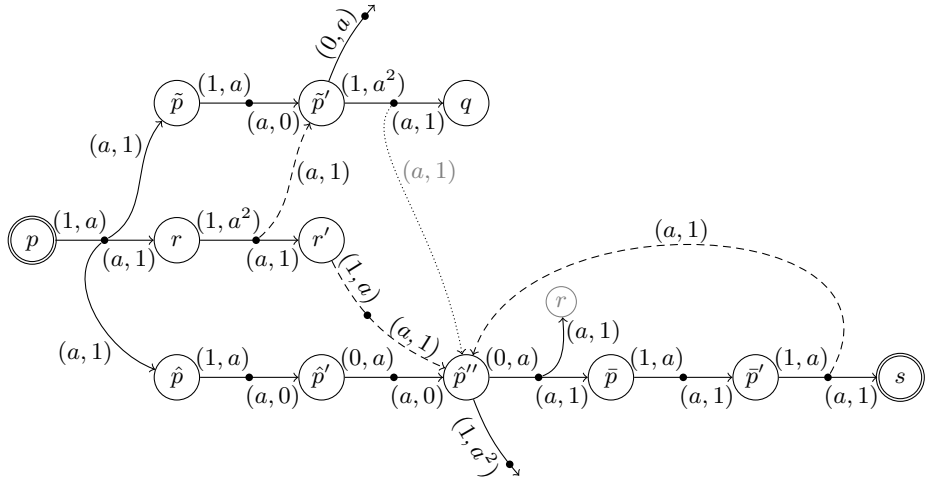


Fig. 6. Simulating SUB using a fixed connection structure and without sending zero spikes.

rules, we notice that simulation of ADD-instructions does not require separate rules, because it can be done as a part of the simulation of SUB-instructions. More exactly, increments are built into the transitions to q and s of $p : (\mathtt{SUB}(r), q, s)$. This has been formalized as *generalized register machine* (GRM for short), see [1]. The rules of U_{22} in the GRM form are given below.

$$
\begin{aligned}
q_1 &: (\mathtt{SUB}(1), \mathtt{ADD}(7)q_1, \mathtt{ADD}(6)q_4)\,,\\
q_4 &: (\mathtt{SUB}(5), \mathtt{ADD}(6)q_4, q_7)\,,\\
q_7 &: (\mathtt{SUB}(6), \mathtt{ADD}(5)q_{10}, q_4)\,,\\
q_{10} &: (\mathtt{SUB}(7), \mathtt{ADD}(1)q_7, q_{13})\,,\\
q_{13} &: (\mathtt{SUB}(6), \mathtt{ADD}(6)q_{14}, q_1)\,,\\
q_{14} &: (\mathtt{SUB}(4), q_1, q_{16})\,,\\
q_{16} &: (\mathtt{SUB}(5), q_{18}, q_{23})\,,\\
q_{18} &: (\mathtt{SUB}(5), q_{20}, q_{27})\,,\\
q_{20} &: (\mathtt{SUB}(5), \mathtt{ADD}(4)q_{16}, \mathtt{ADD}(2)\mathtt{ADD}(3)q_{32})\,,\\
q_{23} &: (\mathtt{SUB}(2), q_{32}, q_{25})\,,\\
q_{25} &: (\mathtt{SUB}(0), q_1, q_{32})\,,\\
q_{27} &: (\mathtt{SUB}(3), q_{32}, \mathtt{ADD}(0)q_1)\,,\\
q_{32} &: (\mathtt{SUB}(4), q_1, q_h)\,.
\end{aligned}
$$

We note that also the first step of the simulation of a generalized SUB-instruction can be embedded into the last step of the preceding simulation. Moreover, note that in this case we may start with one spike in neuron q''_{13}. It is easy to see that it suffices to have 3 rules per each of the 13 generalized conditional decrement instructions and 1 rule per each of the 8 registers, yielding a total

of only **47** rules, associated to register neurons, primed instruction neurons and double-primed instruction neurons.

Register neurons

$0 : (1, a) \to (q''_{25}, \lambda, 1)$,
$1 : (1, a) \to (q''_{1}, \lambda, 1)$,
$2 : (1, a) \to (q''_{23}, \lambda, 1)$,
$3 : (1, a) \to (q''_{27}, \lambda, 1)$,
$4 : (1, a) \to (q''_{14}, \lambda, 1)\,(q''_{32}, \lambda, 1)$,
$5 : (1, a) \to (q''_{4}, \lambda, 1)\,(q''_{16}, \lambda, 1)\,(q''_{18}, \lambda, 1)\,(q''_{20}, \lambda, 1)$,
$6 : (1, a) \to (q''_{7}, \lambda, 1)\,(q''_{13}, \lambda, 1)$,
$7 : (1, a) \to (q''_{10}, \lambda, 1)$.

The **primed instruction neurons** have the rules

$$q'_i : (1, a) \to (q''_i, a, 0) \text{ for } i \in \{1, 4, 7, 10, 13, 14, 16, 18, 20, 23, 25, 27, 32\}.$$

We give the rules of **double-primed instruction neurons** in the table below, the row representing the neuron, the column representing the left side of a rule, and their intersection containing the right side of that rule.

	$(0, a)$	$(1, a)$
q''_1	$(q'_4, a, 1)\,(5, \lambda, 1)(6, a, 0)$	$(q'_1, a, 1)\,(1, \lambda, 1)(7, a, 0)$,
q''_4	$(q'_7, a, 1)\,(6, \lambda, 1)$	$(q'_4, a, 1)\,(5, \lambda, 1)(6, a, 0)$,
q''_7	$(q'_4, a, 1)\,(5, \lambda, 1)$	$(q'_{10}, a, 1)\,(7, \lambda, 1)(5, a, 0)$,
q''_{10}	$(q'_{13}, a, 1)\,(6, \lambda, 1)$	$(q'_7, a, 1)\,(6, \lambda, 1)(1, a, 0)$,
q''_{13}	$(q'_1, a, 1)\,(1, \lambda, 1)$	$(q'_{14}, a, 1)\,(4, \lambda, 1)(6, a, 0)$,
q''_{14}	$(q'_{16}, a, 1)\,(5, \lambda, 1)$	$(q'_1, a, 1)\,(1, \lambda, 1)$,
q''_{16}	$(q'_{23}, a, 1)\,(2, \lambda, 1)$	$(q'_{18}, a, 1)\,(5, \lambda, 1)$,
q''_{18}	$(q'_{27}, a, 1)\,(3, \lambda, 1)$	$(q'_{20}, a, 1)\,(5, \lambda, 1)$,
q''_{20}	$(q'_{32}, a, 1)\,(4, \lambda, 1)(2, a, 0)(3, a, 0)$	$(q'_{16}, a, 1)\,(5, \lambda, 1)(4, a, 0)$,
q''_{23}	$(q'_{25}, a, 1)\,(0, \lambda, 1)$	$(q'_{32}, a, 1)\,(4, \lambda, 1)$,
q''_{25}	$(q'_{32}, a, 1)\,(4, \lambda, 1)$	$(q'_1, a, 1)\,(1, \lambda, 1)$,
q''_{27}	$(q'_1, a, 1)\,(1, \lambda, 1)(0, a, 0)$	$(q'_{32}, a, 1)\,(4, \lambda, 1)$,
q''_{32}	$(q_h, a, 0)$	$(q'_1, a, 1)\,(1, \lambda, 1)$.

This construction uses a total of $8 + 2 \times 13 + 1 =$ **35** neurons. The halting neuron q_h can be avoided, e.g., by changing the right side of the rule with $q''_{32} : (0, a)$ to, e.g., $(4, \lambda, 1)$. Indeed, the register machine halts with register 4 being empty, so the P system will halt after neuron 4 has reset its state to 0. We also remark that this construction does not respect the requirement of all states on the right side being equal. This requirement can be fulfilled by replacing $(r, a, 0)$ by $(r', a, 1)$ for $r \in \{0, 1, 4, 5, 6, 7\}$ and $(2, a, 0)(3, a, 0)$ by $(\langle 2, 3\rangle', a, 1)$ in the right sides of the rules above, and adding 7 additional neurons, each having one rule: $r' : (1, a) \to (r, a, 0)$ for $r \in \{0, 1, 4, 5, 6, 7\}$ and the neuron $\langle 2, 3\rangle$ with the rule $\langle 2, 3\rangle' : (1, a) \to (2, a, 0)(3, a, 0)$, yielding a total of **54** rules.

If we consider the most restricted variant of spiking neural P systems with states elaborated at the end of Section 4, which is comparable with the model considered in [16] using the notion of *polarizations* instead of the notion *states*, when again embedding the first step of the simulation of a generalized SUB-instruction into the last step of the preceding simulation, a straight-forward calculation yields two neurons and two rules per register as well as 7 neurons and 9 rules per generalized conditional decrement instruction, which yields a total of $16 + 7 \times 13 = \mathbf{107}$ neurons as well as $16 + 9 \times 13 = \mathbf{133}$ rules.

It only remains to argue the correctness of the construction. As before, embedded ADD-instructions translate exactly into sending additional spikes to register neurons by existing rules simulating the SUB-instructions. Hence, we only explain the latter, i.e., the decrement case and the zero-test case.

Suppose we simulate a rule $p : (\mathtt{SUB}(r), q, s)$, and the next instruction performs the $\mathtt{SUB}\,(r_q)$ or the $\mathtt{SUB}\,(r_s)$ command, respectively. The configuration of the P system consists of the description of (the state of and the number of spikes in) each of its neurons. Clearly, the neurons in state 0 and without spikes present no interest; we also omit the register neurons different from r. By l we denote any element of SUB_r other than r.

Furthermore, we underline the spikes which are to be removed from the firing neuron, and we will highlight in bold the neurons which are idle and whose state switches from 1 to 0.

Step	p	$\tilde{p}$	$\tilde{p}'$	$\hat{p}$	$\hat{p}'$	$\hat{p}''$	r	r'	$\tilde{l}'$	$\hat{l}''$
1	$(1, \underline{a})$						$(0, a^{n_r})$			
2		$(1, \underline{a})$		$(1, \underline{a})$			$(1, a^{n_r-1+\underline{2}})$			
3			$(1, \underline{a^2})$		$(1, \underline{a})$		$(0, a^{n_r-1})$	$(1, \underline{a})$	$(\mathbf{1}, \mathbf{a})$	
4						$(1, \underline{a^2})$	$(0, a^{n_r-1})$		$(0, \underline{a})$	$(1, \underline{a^2})$
5							$(0, a^{n_r-1})$			

Fig. 7. The trace of the **decrement case**.

The trace of the decrement case is presented in Figure 7. What is not reflected on this figure is that line 4 of the trace of instruction p is superposed with line 2 of instruction q. However, since the neurons are disjoint, no interference happens. Indeed, the last step of simulation of instruction p removes the spikes from neurons $\hat{p}''$, $\hat{l}''$, and $\tilde{l}'$, while the simulation of instruction q acts upon neurons $\tilde{q}$, $\hat{q}$ and r_q. We observe that (if $p \neq q$) a spike is removed from $\tilde{q}'$ simultaneously with a spike being sent to $\tilde{q}'$, which of course causes no interference, since the spikes do not meet. As mentioned before, the first step is omitted (by embedding it into the initial configuration and into every preceding step), so the simulation starts at line 2, taking just 2 steps to produce line 2 for the next instruction, and one more step to remove the superfluous spikes.

Step	p	$\tilde{p}$	$\tilde{p}'$	$\hat{p}$	$\hat{p}'$	$\hat{p}''$	$\bar{p}$	$\bar{p}'$	r	r'	$\tilde{l}'$	$\hat{l}''$
1	$(1, \underline{a})$											
2		$(1, \underline{a})$		$(1, \underline{a})$					$(\mathbf{1}, \mathbf{a})$			
3			$(\mathbf{1}, \mathbf{a})$		$(1, \underline{a})$				$(0, a)$			
4			$(0, \underline{a})$			$(0, \underline{a})$			$(0, a)$			
5							$(1, \underline{a})$		$(1, \underline{a^2})$			
6			$(\mathbf{1}, \mathbf{a})$					$(1, \underline{a})$		$(1, \underline{a})$	$(\mathbf{1}, \mathbf{a})$	
7			$(0, \underline{a})$			$(1, \underline{a^2})$					$(0, \underline{a})$	$(1, \underline{a^2})$
8												

Fig. 8. The trace of the **zero-test case**.

The trace of the zero-test case is presented in Figure 8. Similarly to the decrement case, it does not reflect that line 7 of the trace of instruction p is superposed with line 2 of instruction s. Hence, the arguments about the correctness of the decrement case also hold for the zero-test case, with the following differences. The last step of simulation of instruction p also removes two spikes from neuron $\tilde{p}'$. As mentioned before, the first step is omitted, so the simulation starts at line 2, taking 5 steps to produce line 2 for the next instruction, and one more step to remove the superfluous spikes. This completes the explanation of the correctness.

It is easy to see that forgetting rules may be replaced by sending a spike to one additional dummy neuron.

6 Conclusion

We have shown that only two states (or polarizations as they are called in [16]) are needed for obtaining computational completeness with (extended) spiking neural P systems with states, thus solving an open problem raised at the Brainstorming Week on Membrane Computing in Sevilla at the beginning of February 2016.

References

1. A. Alhazov and R. Freund. Variants of small universal P systems with catalysts. *Fundamenta Informaticae*, 138(1-2):227–250, 2015.
2. A. Alhazov, R. Freund, S. Ivanov, M. Oswald, and S. Verlan. Extended spiking neural P systems with white holes. In L. Macías-Ramos, Gh. Păun, A. Riscos-Núñez, and L. Valencia-Cabrera, editors, *Proceedings of the 13th Brainstorming Week on Membrane Computing*, pages 45–62. Fénix Editora, Sevilla, 2015.
3. A. Alhazov, R. Freund, M. Oswald, and M. Slavkovik. Extended spiking neural P systems. In *Workshop on Membrane Computing*, volume 4361 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2006.

4. H. Chen, R. Freund, M. Ionescu, Gh. Păun, and M. Pérez-Jiménez. On string languages generated by spiking neural P systems. In *Prooceedings of the Fourth Brainstorming Week on Membrane Computing*, volume 1, pages 169–194. Fénix Editora, Sevilla, 2006.
5. H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, and M. J. Pérez-Jiménez. Spiking neural P systems with extended rules: universality and languages. *Natural Computing*, 7(2):147–166, 2007.
6. R. Freund and M. Oswald. A short note on analysing P systems. *Bulletin of the EATCS*, 78:231–236, 2002.
7. M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2,3):279–308, 2006.
8. I. Korec. Small universal register machines. *Theoretical Computer Science*, 168:267–301, 1996.
9. M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
10. Gh. Păun. *Membrane Computing: An Introduction*. Natural Computing Series Natural Computing. Springer, 2002.
11. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000.
12. Gh. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
13. Gh. Păun, Y. Sakakibara, and T. Yokomori. P systems on graphs of restricted forms. *Publicationes Mathematicae Debrecen*, 60:635–660, 2006.
14. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 1-3. Springer, 1997.
15. The P Systems Website.
16. T. Wu, A. Păun, Z. Zhang, and L. Pan. Spiking neural P systems with polarizations. *submitted*, 2015.

Computational Completeness of P Systems Using Maximal Variants of the Set Derivation Mode

Artiom Alhazov[1], Rudolf Freund[2], and Sergey Verlan[3]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău, MD-2028, Moldova
E-mail: artiom@math.md
[2] Faculty of Informatics, TU Wien
Favoritenstraße 9-11, 1040 Wien, Austria
E-mail: rudi@emcc.at
[3] LACL, Université Paris Est – Créteil Val de Marne
61, av. Général de Gaulle, 94010, Créteil, France
Email: verlan@u-pec.fr

Summary. We consider P systems only allowing rules to be used in at most one copy in each derivation step, especially the variant of the maximally parallel derivation mode where each rule may only be used at most once. Moreover, we also consider the derivation mode where from those sets of rules only those are taken which have the maximal number of rules. We check the computational completeness proofs of several variants of P systems and show that some of them even literally still hold true for the for these two new set derivation modes. Moreover, we establish two new results for P systems using target selection for the rules to be chosen together with these two new set derivation modes.

1 Introduction

Membrane systems with symbol objects are a theoretical framework of parallel distributed multiset processing. Usually, multisets of rules are applied in parallel to the objects in the underlying configuration; for example, in the maximally parallel derivation mode (abbreviated max), a non-extendable multiset of rules is applied to the current configuration. In this paper we now consider variants of these derivation modes, where each rule is only used in at most one copy, i.e., we consider sets of rules to be applied in parallel, for example, in the *set-maximally parallel derivation mode* (abbreviated $smax$) we apply non-extendable *sets* of rules, and in another derivation mode we apply sets of rules which contain a maximal number of applicable rules (abbreviated max_{rule}).

Taking sets of rules instead of multisets is a quite natural restriction and it arises from different motivations, e.g., firing a maximal set of transitions in Petri Nets [5, 8] or optimizing an implementation of FPGA simulators [13]. A natural question arises concerning the power of set-based modes in contrast to multiset-based ones. The first attempt to go into this direction was done in [10] where it was shown that in some cases the computational completeness results established for the max-mode also hold for the $smax$-mode.

In this paper we continue this line of research and we show that for several variants of P systems the proofs for computational completeness for max can be taken over even literally for $smax$ and eventually even for max_{rule}, but on the other hand there are also variants of P systems where the derivation modes $smax$ and max_{rule} yield even stronger results than the max-mode.

2 Variants of P Systems

In this section we recall the well-known definitions of several variants of P systems as well as some variants of derivation modes and also introduce the variants of set derivation modes considered in the following.

A (cell-like) P system is a construct

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f_O, f_I) \text{ where}$$

- O is the alphabet of objects,
- $C \subset O$ is the set of catalysts,
- μ is the membrane structure (with m membranes),
- $w_1, \ldots, w_m$ are multisets of objects present in the m regions of μ at the beginning of a computation,
- $R_1, \ldots, R_m$ are finite sets of rules, associated with the regions of μ,
- f_O is the label of the membrane region from which the outputs are taken (in the generative case)
- f_I is the label of the membrane region where the inputs are put at the beginning of a computation (in the accepting case).

$f_O = 0/f_I = 0$ indicates that the output/input is taken from the environment.

If a rule $u \to v$ has at least two objects in u, then it is called *cooperative*, otherwise it is called *non-cooperative*. *Catalytic rules* are of the form $ca \to cv$, where $c \in C$ is a special object which never evolves and never passes through a membrane, it just assists object a to evolve to the multiset v.

In *catalytic P systems* we use non-cooperative as well as catalytic rules. In a *purely catalytic P system* we only allow catalytic rules.

In the *maximally parallel derivation mode* (abbreviated by max), in any computation step of Π we choose a multiset of rules from $\mathcal{R}$, defined as the union of the sets $R_1, \ldots, R_m$, in such a way that no further rule can be added to it so that the obtained multiset would still be applicable to the existing objects in the regions $1, \ldots, m$.

2.1 Set Derivation Modes

The basic set derivation mode is defined as the derivation mode where in each derivation step at must one copy of each rule may be applied in parallel with the other rules; this variant of a basic derivation mode corresponds to the asynchronous mode with the restriction that only those multisets of rules are applicable which contain at most one copy of each rule, i.e., we consider *sets* of rules:

$$Appl(\Pi, C, set) = \{R \in Appl(\Pi, C, asyn) \mid |R|_r \leq 1 \text{ for each } r \in \mathcal{R}\}$$

In the *set-maximally parallel derivation mode* (this derivation mode is abbreviated by $smax$ for short), in any computation step of Π we choose a non-extendable multiset R of rules from $Appl(\Pi, C, set)$; following the notations elaborated in [7], we define the mode $smax$ as follows:

$$Appl(\Pi, C, smax) = \{R \in Appl(\Pi, C, set) \mid \text{there is no } R' \in Appl(\Pi, C, set) \text{ such that } R' \supset R\}$$

The $smax$-derivation mode corresponds to the min_1-mode with the discrete partitioning of rules (each rule forms its own partition), see [7].

The derivation mode $max_{rule}smax$ is a special variant where only a maximal set of rules is allowed to be applied. But it can also seen as the variant of the basic set mode where we just take a set of applicable rules with the maximal number of rules in in it, hence, we will also call it the max_{rule} derivation mode. Formally we have:

$$Appl(\Pi, C, max_{rule}) = \{R \in Appl(\Pi, C, set) \mid \text{there is no } R' \in Appl(\Pi, C, set) \text{ such that } |R'| > |R|\}$$

As usual, with all these variants of derivation modes as defined above, we consider halting computations. We may generate or accept or even computing functions or relations. The inputs/outputs may be multisets or strings, defined in the well-known way.

2.2 The History of the *smax*-Derivation Mode

In [13], a paper on fast P systems simulators using FPGA, the problem of the unbounded max-mode was considered as too difficult to be parallelized on this hardware. In the quest for an efficient solution, the authors proposed to restrict to the case of the maximal parallelism where each rule can be applied at most once. The most important advantage of this variant was that the multiset of applicable rules could be represented as a binary string, i.e., an encoding as a number. Moreover, the paper showed that in many interesting cases it is possible to represent the language of corresponding binary strings at each step by an automaton. Then the problem of the simulation of a P system could be solved as follows:

- Find the size S of the set of multisets of applicable rules (the size of the language of binary strings).
- Take a random number $k \in \{1..S\}$ and chose the string representing k.

This algorithm allowed for obtain a speed-up of magnitude 10^5.

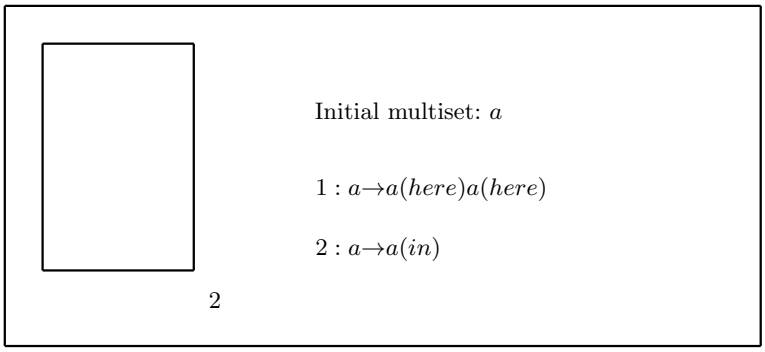
The advantages of the set-maximally parallel derivation mode over the unbounded maximally parallel derivation mode are:

- A compact representation of the applicable multisets of rules as binary strings/numbers is obtained.
- Most of the computational completeness results still hold.
- Simpler analysis of the behavior is possible.
- Only a bounded number of (multi)sets of rules has to be computed for each derivation step.

In [10], the *set-maximally parallel derivation mode* was called *flat maximal parallel derivation mode*, and, for example, P systems with promoters are shown to be computationally complete using this flat maximal parallel derivation mode with non-cooperative rules.

2.3 Examples

In the maximally parallel mode, we in addition need target or rule or label agreement to obtain $\{a^{2^n} \mid n \geq 0\}$, otherwise only $\{a^n \mid n \geq 1\}$ can be obtained.



target/ rule/ label agreement:
the same rule is used for all symbols a

Fig. 1. Example of a P system.

In the set-maximally parallel mode $smax$, we in addition need target or rule or label agreement to obtain $\{a^n \mid n \geq 1\}$, otherwise only $\{a\}$ can be obtained, because:

- If $2 : a \to a(in)$ is used in the first step, then a is obtained.
- If $1 : a \to a(here)a(here)$ is applied at least once, then from the second step on it has to be applied infinitely often, as only one copy of a can be sent into membrane 2 by the second rule $2 : a \to a(in)$.

The same arguments hold for the derivation mode max_{rule}.

3 Symport/Antiport P Systems

A *symport/antiport P system* is a construct

$$\Pi = (O, E, \mu, w_0, w_1, \ldots, w_m, R_1, \ldots, R_m, f_O, f_I) \text{ where}$$

- O is the alphabet of objects,
- $E \subseteq O$ is the set of objects being available in the environment in an unbounded number,
- μ is the membrane structure (with m membranes),
- w_0 is the finite multiset of objects over $O \setminus E$ present in the environment at the beginning of a computation,
- $w_1, \ldots, w_m$ are the multisets of objects present in the m regions of μ at the beginning of a computation,
- $R_1, \ldots, R_m$ are finite sets of symport and/or antiport rules, associated with the membranes of μ,
- f_O, f_I is the label of the membrane region from which the outputs are taken/the inputs are put in.

Every rule is of the form $(u, out; v, in)$ with $u, v \in O^*$ and $uv \neq \lambda$; if $u = \lambda$ or $v = \lambda$ then this rule is called a *symport rule*, otherwise it is called an *antiport rule*. The application of a rule $(u, out; v, in) \in R_i$ means sending out u from region i and taking v into it from the surrounding region.

For $(u, out; v, in)$, $\max\{|u|, |v|\}$ is called its *weight* and $|uv|$ is called its *size*; obviously, for symport rules weight and size are the same.

The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and $\gamma \in \{sequ, asyn, max, smax, max_{rule}, \ldots\}$, computed by symport/antiport P systems with at most m membranes, symport rules with maximal weight r as well as antiport rules with maximal weight w and maximal size s are denoted by $Y_{\gamma,\delta}OP_m(sym_r, anti_{w,s})$.

3.1 Accepting Antiport P Systems

Theorem 1. *For $Y \in \{N, Ps\}$, $\beta \in \{max, smax, max_{rule}\}$,*

$$Y_{\beta,acc}DOP_m(anti_{2,3}) = YRE.$$

Proof. Let $M = (m, B, l_0, l_h, P)$ be an arbitrary deterministic register machine. We now construct an antiport P system simulating M. The number in register r is represented by the corresponding number of symbol objects o_r.

- An ADD-instruction $p : (ADD(r), q)$ is simulated by the rule $(p, out; o_r q, in)$.
- A SUB-instruction $p : (SUB(r), q, s)$ is simulated by the following rules
  1. $(p, out; p'p'', in)$;
  2. $(p', out; \tilde{p}, in)$ as well as $(p''o_r, out; \bar{p}, in)$ which is executed in parallel if and only if the register is not empty;
  3. $(\tilde{p}p'', out; s, in)$ (if register was empty),
     $(\tilde{p}\bar{p}, out; q, in)$ (if register was not empty).

As can be seen immediately, in each step only different rules can be applied, each of them only once. Hence, the proof elaborated for the max-mode literally also works for the derivation modes $smax$ and max_{rule} without any restrictions as well. □

4 P Systems with Anti-Matter

For any object *a (matter)*, we consider its anti-object *(anti-matter)* a^- and the corresponding (cooperative) *annihilation rule* $aa^- \to \lambda$. This rule is assumed to exist in all membranes.

In the following, we assume these annihilation rules to have (weak) priority over all other rules, i.e., other rules may only be applied if objects cannot be bound by an annihilation rule any more.

This type of rules is abbreviated by $antim/pri$, indicating matter/anti-matter annihilation rules having weak priority. For further results we refer to [1].

4.1 Matter/Anti-Matter Annihilation Rules Having Priority

The matter/anti-matter annihilation rules are so powerful that we only need the minimum number of catalysts, i.e., zero ($cat(0) = ncoo$).

Theorem 2. *[1] For any* $n \geq 1$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, $Z \in \{Fun, Rel\}$, *and* $\beta \in \{max, smax, max_{rule}\}$,

$$Y_{\beta,\delta}OP_n\,(ncoo, antim/pri) = YRE \text{ and}$$
$$ZY_{\beta,\alpha}OP_n\,(ncoo, antim/pri) = ZYRE.$$

4.2 Deterministic Matter/Anti-Matter Accepting P Systems

In the accepting case, we can even simulate the actions of a deterministic register machine in a deterministic way, i.e., for each configuration of the system, there can be at most one multiset of rules applicable to it. Yet the proof exhibited in [1], even fulfills the condition that every rule is only applied at most once.

Theorem 3. *For any* $n \geq 1$,$Y \in \{N, Ps\}$, *and* $\beta \in \{max, smax, max_{rule}\}$,

$$Y_{\beta,detacc}OP_n\,(ncoo, antim/pri) = YRE \text{ and}$$
$$FunY_{\beta,detacc}OP_n\,(ncoo, antim/pri) = FunYRE.$$

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

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

$$\tilde{M}_r = \left\{\tilde{l} \mid l : (SUB\,(r)\,, l', l'') \in P\right\},$$
$$\tilde{M}_r{}^- = \left\{\tilde{l}^- \mid l : (SUB\,(r)\,, l', l'') \in P\right\},$$
$$\hat{M}_r = \left\{\hat{l} \mid l : (SUB\,(r)\,, l', l'') \in P\right\},$$
$$\hat{M}_r{}^- = \left\{\hat{l}^- \mid l : (SUB\,(r)\,, l', l'') \in P\right\}.$$

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

$$l_1 \to \bar{l}_1 a_r{}^-,$$
$$\bar{l}_1 \to \hat{l}_1{}^-(\tilde{M}_r \setminus \{\tilde{l}_1\}),$$
$$\hat{l}_1{}^- \to l_2(\tilde{M}_r{}^- \setminus \{\tilde{l}_1{}^-\}), \text{ and}$$
$$\tilde{l}_1{}^- \to l_3(\hat{M}_r{}^- \setminus \{\hat{l}_1{}^-\}).$$

The symbol $\hat{l}_1{}^-$ generated by the second rule is eliminated again and replaced by $\tilde{l}_1{}^-$ if $a_r{}^-$ is not annihilated.

Again, the proof elaborated for the max-mode literally also works for the derivation modes $smax$ and max_{rule} without any restrictions as well. □

5 Catalytic and Purely Catalytic P Systems

We now investigate proofs elaborated for catalytic and purely catalytic P systems working in the max-mode for the $smax$-mode.

5.1 Computational Completeness of Catalytic P Systems

We first check the construction for simulating a register machine $M = (d, B, l_0, l_h, R)$ by a catalytic P system Π, with $m \leq d$ being the number of decrementable registers, elaborated in [3] for the max-mode, and argue why it works for the smax-mode, too.

For all d registers, n_i copies of the symbol o_i are used to represent the value n_i in register i, $1 \leq i \leq d$. For each of the m decrementable registers, we take

a catalyst c_i and two specific symbols d_i, e_i, $1 \leq i \leq m$, for simulating SUB-instructions on these registers. For every $l \in B$, we use p_l, and also its variants $\bar{p}_l, \hat{p}_l, \tilde{p}_l$ for $l \in B_{SUB}$, where B_{SUB} denotes the set of labels of SUB-instructions.

$$
\begin{aligned}
\Pi &= (O, C, \mu = [\ ]_1, w_1 = c_1 \dots c_m d_1 \dots d_m p_1 w_0, R_1, f = 1),\\
O &= C \cup D \cup E \cup \Sigma\\
&\quad \cup \{\#\} \cup \{p_l \mid l \in B\} \cup \{\bar{p}_l, \hat{p}_l, \tilde{p}_l \mid l \in B_{SUB}\},\\
C &= \{c_i \mid 1 \leq i \leq m\},\\
D &= \{d_i \mid 1 \leq i \leq m\},\\
E &= \{e_i \mid 1 \leq i \leq m\},\\
\Sigma &= \{o_i \mid 1 \leq i \leq d\},\\
R_1 &= \{p_j \to o_r p_k D_m, p_j \to o_r p_l D_m \mid j : (ADD(r), k, l) \in R\}\\
&\quad \cup \{p_j \to \hat{p}_j e_r D_{m,r}, p_j \to \bar{p}_j D_{m,r}, \hat{p}_j \to \tilde{p}_j D'_{m,r},\\
&\qquad \bar{p}_j \to p_k D_m, \tilde{p}_j \to p_k D_m \mid j : (SUB(r), k, l) \in R\}\\
&\quad \cup \{c_r o_r \to c_r d_r, c_r d_r \to c_r, c_{r\oplus_m 1} e_r \to c_{r\oplus_m 1} \mid 1 \leq r \leq m\},\\
&\quad \cup \{d_r \to \#, c_r e_r \to c_r \# \mid 1 \leq r \leq m\}\\
&\quad \cup \{\# \to \#\}.
\end{aligned}
$$

Here $r \oplus_m 1$ for $r < m$ simply is $r + 1$, whereas for $r = m$ we define $m \oplus_m 1 = 1$; w_0 stands for additional input present at the beginning.

Usually, every catalyst c_i, $i \in \{1, \dots, m\}$, is kept busy with the symbol d_i using the rule $c_i d_i \to c_i$, as otherwise the symbols d_i would have to be trapped by the rule $d_i \to \#$, and the trap rule $\# \to \#$ then enforces an infinite non-halting computation.

In the *smax*-derivation mode only one trap rule $\# \to \#$ will be carried out, but this is the only difference!

Only during the simulation of SUB-instructions on register r the corresponding catalyst c_r is left free for decrementing or for zero-checking in the second step of the simulation, and in the decrement case both c_r and its "coupled" catalyst $c_{r\oplus_m 1}$ are needed to be free for specific actions in the third step of the simulation.

For the simulation of instructions, we use:

$$
\begin{aligned}
D_m &= \prod_{i \in [1..m]} d_i,\\
D_{m,r} &= \prod_{i \in [1..m] \setminus \{r\}} d_i,\\
D'_{m,r} &= \prod_{i \in [1..m] \setminus \{r, r\oplus_m 1\}} d_i.
\end{aligned}
$$

The HALT-instruction labeled l_h is simply simulated by not introducing the corresponding state symbol p_{l_h}, i.e., replacing it by λ, in all rules defined in R_1.

Each ADD-instruction $j : (ADD(r), k, l)$, for $r \in \{1, \dots, d\}$, can easily be simulated by the rules $p_j \to o_r p_k D_m$ and $p_j \to o_r p_l D_m$; in parallel, the rules $c_i d_i \to c_i$, $1 \leq i \leq m$, have to be carried out, as otherwise the symbols d_i would have to be trapped by the rules $d_i \to \#$.

Each SUB-instruction $j : (SUB(r), k, l)$, is simulated as shown in the table listed below (the rules in brackets [and] are those to be carried out in case of a wrong choice):

Simulation of the SUB-instruction $j : (SUB(r), k, l)$ if	
register r is not empty	register r is empty
$p_j \to \hat{p}_j e_r D_{m,r}$	$p_j \to \bar{p}_j D_{m,r}$
$c_r o_r \to c_r d_r \ [c_r e_r \to c_r \#]$	c_r should stay idle
$\hat{p}_j \to \tilde{p}_j D'_{m,r}$	$\bar{p}_j \to p_k D_m$
$c_r d_r \to c_r \ [d_r \to \#]$	$[d_r \to \#]$
$\tilde{p}_j \to p_k D_m$	
$c_{r\oplus_m 1} e_r \to c_{r\oplus_m 1}$	

In the first step of the simulation of each instruction (ADD-instruction, SUB-instruction, and even HALT-instruction) due to the introduction of D_m in the previous step (we also start with that in the initial configuration) every catalyst c_r is kept busy by the corresponding symbol d_r, $1 \leq r \leq m$.

Based on the construction elaborated in [3] and recalled above in sum we have obtained the following result:

Theorem 4. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a catalytic P system*

$$\Pi = (O, C, \mu = [\ \ ]_1, w_1, R_1, f = 1)$$

working in the max- or the smax-derivation mode and simulating the computations of M such that

$$|R_1| \leq ADD^1(R) + 2 \times ADD^2(R) + 5 \times SUB(R) + 5 \times m + 1,$$

where $ADD^1(R)$ denotes the number of deterministic ADD-instructions in R, $ADD^2(R)$ denotes the number of non-deterministic ADD-instructions in R, and $SUB(R)$ denotes the number of SUB-instructions in R.

5.2 Computational Completeness of Purely Catalytic P Systems

For the purely catalytic case, one additional catalyst c_{m+1} is needed to be used with all the non-cooperative rules. Unfortunately, in this case a slightly more complicated simulation of SUB-instructions is needed, a result established in [12], where for catalytic P systems

$$|R_1| \leq 2 \times ADD^1(R) + 3 \times ADD^2(R) + 6 \times SUB(R) + 5 \times m + 1,$$

and for purely for catalytic P systems

$$|R_1| \leq 2 \times ADD^1(R) + 3 \times ADD^2(R) + 6 \times SUB(R) + 6 \times m + 1,$$

is shown. Yet also this proof literally works for the *smax*-derivation mode as well, with the only exception that the trap rule $\# \to \#$ is carried out at most once.

6 Computational Completeness of (Purely) Catalytic P Systems with Additional Control Mechanisms

In this section we consider (purely) catalytic P systems with additional control mechanisms, in that way reaching computational completeness with only one (two) catalyst(s).

6.1 P Systems with Label Selection

For all the variants of P systems of type X, 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

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

where $\Pi' = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f)$ is a P system as defined above, 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 we first select a set of labels $U \in W$ and then apply a non-empty multiset R of rules such that all the labels of these rules in R are in U in the maximally parallel way, i.e., the set R cannot be extended by any further rule with a label from U so that the obtained multiset of rules would still be applicable to the existing objects in the membrane regions $1, \ldots, m$. The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and $\gamma \in \{sequ, asyn, max, smax, max_{rule}, \ldots\}$, computed by P systems with label selection with at most m membranes and rules of type X is denoted by $Y_{\gamma,\delta}OP_m(X, ls)$.

The proof of the following theorem is based on the proof given in [6] for the maximally parallel mode max; the proof can be taken over for the mode $smax$ word by word; the only difference is that in non-successful computations where more than one trap symbol # has been generated, the trap rule $\# \to \#$ is only applied once.

Theorem 5. *$Y_{\gamma,\delta}OP_1(cat_1, ls) = Y_{\gamma\delta}OP_1(pcat_2, ls) = YRE$ for any $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and $\gamma \in \{max, smax\}$.*

Proof. We only prove the inclusion $PsRE \subseteq Ps_{smax,gen}OP_1(cat_1, ls)$. Let us consider a register machine $M = (n+2, B, l_0, l_h, I)$ with only the first and the second register ever being decremented, and let $A = \{a_1, \ldots, a_{n+2}\}$ be the set of objects for representing the contents of the registers 1 to $n+2$ of M. We construct the following P system:

$$\begin{aligned}
&\Pi = (O, \{c\}, [\ ]_1, cdl_0, R_1, H, W, 0),\\
&O = A \cup B \cup \{d, \#\},\\
&H = \{l, l' \mid l \in B\} \cup \left\{l_{\langle x \rangle} \mid x \in \{1, 2, 1', 2', d, \#\}\right\},
\end{aligned}$$

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

A. Let $l_i : (\texttt{ADD}(r), l_j, l_k)$ be an ADD instruction in I. If $r > 2$, then the (labeled) rules

$$l_i : l_i \to l_j\,(a_r, out)\,, \quad l_i' : l_i \to l_k\,(a_r, out)\,,$$

are used, and for $r \in \{1, 2\}$, we take the rules

$$l_i : l_i \to l_j a_r, \quad l_i' : l_i \to l_k a_r.$$

In both cases, we define $\{l_i, l_i'\}$ to be the corresponding set of labels in W. The contents of each register r, $r \in \{1, 2\}$, is represented by the number of objects a_r present in the skin membrane; any object a_r with $r \geq 3$ is immediately sent out into the environment.

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

For the case that the register r, $r \in \{1, 2\}$, is not empty we take the (labeled) rules

$$l_i : l_i \to l_j, \quad l_{\langle r \rangle} : ca_r \to c, \quad l_{\langle d \rangle} : cd \to c\#,$$

(if no symbol a_r is present, i.e., if the register r is empty, then the trap symbol $\#$ is introduced by the rule $l_{\langle d \rangle} : cd \to c\#$).

For the case that the register r is empty, we take the (labeled) rules

$$l_i' : l_i \to l_k, \quad l_{\langle r' \rangle} : ca_r \to c\#$$

(if at least one symbol a_r is present, i.e., if the register r is not empty, then the trap symbol $\#$ is introduced by the rule $l_{\langle r' \rangle} : ca_r \to c\#$).

The corresponding sets of labels to be taken into W are $\{l_i, l_{\langle r \rangle}, l_{\langle d \rangle}\}$ and $\{l_i', l_{\langle r' \rangle}\}$, respectively. In both cases, the simulation of the SUB instruction works correctly if we have made the right choice.

C. As soon as the final label l_h is reached, we apply the rules

$$l_h : l_h \to \lambda, \quad l_h' : cd \to c$$

according to the set of labels $\{l_h, l_h'\}$ in W. In fact, neglecting the single catalyst c, we could even obtain a clean result in the skin membrane when leaving the result objects in the skin membrane instead of sending them out.

D. We also add the labeled rule $l_{\langle \# \rangle} : \# \to \#$ to R_1 and the set $\{l_{\langle \# \rangle}\}$ to W, hence, the computation cannot halt once the trap symbol $\#$ has been generated.

In sum, we observe that each computation step in M is simulated by exactly one computation step in Π; moreover, such a simulating computation in Π halts if and only if the corresponding computation in M halts (as soon as the label l_h appears, only the set of rules $\{l_h, l_h'\}$ can be applied, and afterwards no rule can be applied anymore in Π, of course, provided that no trap symbol is present).

If at some moment we make the wrong choice when trying to simulate a SUB instruction and have to generate the trap symbol #, the computation will never halt. Hence, we have shown $Ps(M) = Ps(\Pi)$, which completes the proof for the catalytic case.

For the purely catalytic case, all the non-cooperative rules are associated with the second catalyst, which immediately yields the corresponding purely catalytic P system with two catalysts. □

6.2 Controlled P Systems and Time-Varying P Systems

Another method to control the application of the labeled rules is to use control languages (see [9] and [4]). A *controlled P system* is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, H, L, f)$$

where $\Pi' = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f)$ is a P system as defined above, H is a set of labels for the rules in the sets $R_1, \ldots, R_m$, and L is a string language over 2^H (each subset of H represents an element of the alphabet for L) from a family FL. Every successful computation in Π has to follow a control word $U_1 \ldots U_n \in L$: in transition step i, only rules with labels in U_i are allowed to be applied (but again in the maximally parallel way, i.e., we have to apply a multiset R of rules with labels in U_i which cannot be extended by any rule with a label in U_i such that the resulting multiset would still be applicable), and after the n-th transition, the computation halts; we may relax this end condition, i.e., we may stop after the i-th transition for any $i \leq n$, and then we speak of *weakly controlled P systems*. If $L = (U_1 \ldots U_p)^*$, Π is called a *(weakly) time-varying P system*: in the computation step $pn + i$, $n \geq 0$, rules from the set U_i have to be applied; p is called the *period*. The family of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, computed by (weakly) controlled P systems and (weakly) time-varying P systems with period p, with at most m membranes and rules of type X as well as control languages in FL is denoted by $Y_{\gamma,\delta}OP_m(X, C(FL))$ ($Y_{\gamma,\delta}OP_m(X, wC(FL))$) and $Y_{\gamma,\delta}OP_m(X, TV_p)$ ($Y_{\gamma,\delta}OP_m(X, wTV_p)$), respectively, for $\delta \in \{gen, acc\}$ and $\gamma \in \{sequ, asyn, max, smax, max_{rule}, \ldots\}$.

The proof of the following theorem again is taken over for the mode *smax* word by word as given in [6] for the maximally parallel mode *max*.

Theorem 6. *$Y_{\gamma,\delta}OP_1(cat_1, \alpha TV_6) = Y_{\gamma,\delta}OP_1(pcat_2, \alpha TV_6) = YRE$, for any $\alpha \in \{\lambda, w\}$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and $\gamma \in \{max, smax\}$.*

Proof. We only prove the inclusion $PsRE \subseteq Ps_{smax,gen}OP_1(cat_1, TV_6)$. Let us consider a register machine $M = (n+2, B, l_0, l_h, I)$ with only the first and the second register ever being decremented. Again, we define $A = \{a_1, \ldots, a_{n+2}\}$ and divide the set of labels $B \setminus \{l_h\}$ into three disjoint subsets:

$$B_+ = \{l_i \mid l_i : (\texttt{ADD}(r), l_j, l_k) \in I\},$$
$$B_{-r} = \{l_i \mid l_i : (\texttt{SUB}(r), l_j, l_k) \in I\},\ r \in \{1,2\};$$

moreover, we define $B_- = B_{-1} \cup B_{-2}$ as well as

$$B' = \left\{l, \tilde{l}, \hat{l} \mid l \in B \setminus \{l_h\}\right\} \cup \left\{l^-, l^0, \bar{l}^-, \bar{l}^0, \mid l \in B_-\right\}.$$

The main challenge in the construction for the time-varying P system Π is that the catalyst has to fulfill its task to erase an object a_r, $r \in \{1,2\}$, for both objects in the same membrane where all other computations are carried out, too; hence, at a specific moment in the cycle of period six, parts of simulations of different instructions have to be coordinated in parallel. The basic components of the time-varying P system Π are defined as follows (we here do not distinguish between a rule and its label):

$$\Pi = (O, \{c\}, [\ ]_1, cl_0, R_1 \cup \cdots \cup R_6, R_1 \cup \cdots \cup R_6, (R_1 \ldots R_6)^*, 0),$$
$$O = A \cup \{a_1', a_2'\} \cup B' \cup \{c, h, l_h, \#\}.$$

We now list the rules in the sets of rules R_i to be applied in computation steps $6n+i$, $n \geq 0$, $1 \leq i \leq 6$:

$\mathbf{R}_1$: in this first step of the cycle, especially all the ADD instructions are simulated, i.e., for each $l_i : (\texttt{ADD}(r), l_j, l_k) \in I$ we take

$cl_i \to ca_r\tilde{l}_j$, $cl_i \to ca_r\tilde{l}_k$ for $r \in \{1,2\}$ as well as $cl_i \to c(a_r, out)\tilde{l}_j$, $cl_i \to c(a_r, out)\tilde{l}_k$ for $3 \leq r \leq n+2$ (in order to obtain the output in the environment, for $r \geq 3$ we have to take (a_r, out) instead of a_r); only in the sixth step of the cycle, from $\tilde{\tilde{l}}_j$ and $\tilde{l}_k$ the corresponding unmarked labels l_j and l_k will be generated;

$cl \to cl^-$, $cl \to cl^0$ initiate the simulation of a SUB instruction for register 1 labeled by $l \in B_{-1}$, i.e., we make a non-deterministic guess whether register r is empty (with introducing l^0) or not (with introducing l^-);

$cl \to c\hat{l}$ marks a label $l \in B_{-2}$ (the simulation of such a SUB instruction for register 2 will start in step 4 of the cycle);

$\# \to \#$ keeps the trap symbol $\#$ alive guaranteeing an infinite loop once $\#$ has been generated;

$h \to \lambda$ eliminates the auxiliary object h which eventually has been generated two steps before (h is needed for simulating the decrement case of SUB instructions).

$\mathbf{R}_2$: in the second and the third step, the SUB instructions on register 1 are simulated, i.e., for all $l \in B_{-1}$ we start with

$ca_1 \to ca_1'$ (if present, exactly one copy of a_1 can be primed, but only if a label l^- for some l from B_{-1} is present) and

$l^- \to \bar{l}^- h$, $l^0 \to \bar{l}^0$ for all $l \in B_{-1}$;

all other labels $\tilde{l}$ for $l \in B$ block the catalyst c from erasing a copy of a_1 by forcing the application of the corresponding rules $c\tilde{l} \to c\tilde{l}$ for c in order to avoid

the introduction of the trap symbol # by the enforced application of a rule $\tilde{l} \to \#$, i.e., we take

$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B$, and

$c\hat{l} \to c\hat{l}$, $\hat{l} \to \#$ for all $l \in B_{-2}$;

$\# \to \#$ keeps the computation alive once the trap symbol has been introduced.

$\mathbf{R}_3$: for all $l_i : (\mathtt{SUB}\,(1), l_j, l_k) \in I$ we take

$c\bar{l}_i^0 \to c\tilde{l}_k$, $a_1' \to \#$, $\bar{l}_i^0 \to \#$ (zero test; if a primed copy of a_1 is present, then the trap symbol # is generated);

$\bar{l}_i^- \to \tilde{l}_j$, $ca_1' \to c$, $ch \to c\#$ (decrement; the auxiliary symbol h is needed to keep the catalyst c busy with generating the trap symbol # if we have taken the wrong guess when assuming the register 1 to be non-empty);

$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B$ (with these labels, we just pass through this step);

$c\hat{l} \to c\hat{l}$, $\hat{l} \to \#$ for all $l \in B_{-2}$ (these labels pass through this step to become active in the next step);

$\# \to \#$.

$\mathbf{R}_4$: in the fourth step, the simulation of SUB instructions on register 2 is initiated by using

$c\hat{l} \to cl^-$, $c\hat{l} \to cl^0$ for all $l \in B_{-2}$, i.e., we make a non-deterministic guess whether register r is empty (with introducing l^0) or not (with introducing l^-);

$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B$ (with all other labels, we only pass through this step);

$\# \to \#$,

$h \to \lambda$ (if h has been introduced by $l^- \to \bar{l}^- h$ in the second step for some $l \in B_{-1}$, we now erase it).

$\mathbf{R}_5$: in the fifth and the sixth step, the SUB instructions on register 2 are simulated, i.e., for all $l \in B_{-2}$ we start with

$ca_2 \to ca_2'$ (if present, exactly one copy of a_2 can be primed) and

$l^- \to \bar{l}^- h$, $l^0 \to \bar{l}^0$ for all $l \in B_{-2}$;

$c_1\tilde{l} \to c_1\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B$;

$\# \to \#$.

$\mathbf{R}_6$: the simulation of SUB instructions $l_i : (\mathtt{SUB}\,(2), l_j, l_k) \in I$ on register 2 is finished by

$c\bar{l}_i^0 \to cl_k$, $a_2' \to \#$, $\bar{l}_i^0 \to \#$ (zero test; if a primed copy of a_2 is present, then the trap symbol # is generated);

$\bar{l}_i^- \to l_j$, $ca_2' \to c$, $ch \to c\#$ (decrement; the auxiliary symbol h is needed to keep the catalyst c busy with generating the trap symbol # if we have taken the wrong guess when assuming the register 2 to be non-empty; if it is not used, it can be erased in the next step by using $h \to \lambda$ in R_1);

$c\tilde{l} \to cl$, $\tilde{l} \to \#$ for all $l \in B$;

$\# \to \#$.

Without loss of generality, we may assume that the final label l_h in M is only reached by using a zero test on register 2; then, at the beginning of a new cycle,

after a correct simulation of a computation from M in the time-varying P system Π no rule will be applicable in R_1 (another possibility would be to take $c\bar{l}_i^0 \to c$ instead of $c\bar{l}_i^0 \to cl_h$ in R_6).

At the end of the cycle, in case all guesses have been correct, the requested instruction of M has been simulated and the label of the next instruction to be simulated is present in the skin membrane. Only in the case that M has reached the final label l_h, the computation in Π halts, too, but only if during the simulation of the computation of M in Π no trap symbol # has been generated; hence, we conclude $Ps(M) = Ps(\Pi)$.

For the purely catalytic case, all the non-cooperative rules are associated with the second catalyst, which immediately yields the corresponding purely catalytic P system with two catalysts. □

7 P Systems with Toxic Objects

In many variants of (catalytic) P systems, for proving computational completeness it is common to introduce a trap symbol # for the case that the derivation goes the wrong way as well as the rule $\# \to \#$ (or $c\# \to c\#$ with a catalyst c) guaranteeing that the derivation will never halt. Yet most of these rules can be avoided if we specify a specific subset of *toxic* objects O_{tox}.

The P system with toxic objects is only allowed to continue a computation from a configuration C by using an applicable multiset of rules covering all copies of objects from O_{tox} occurring in C; moreover, if there exists no multiset of applicable rules covering all toxic objects, the whole computation having yielded the configuration C is abandoned, i.e., no results can be obtained from this computation.

For any variant of P systems, we add the set of *toxic* objects O_{tox} and in the specification of the families of sets of (vectors of) numbers generated by P systems with toxic objects using rules of type X we add the subscript tox to O, thus obtaining the families $Y_{\gamma,gen}O_{tox}P_m(X)$, for any $m \geq 1$, $\gamma \in \{sequ, asyn, max, smax, max_{rule}\}$, and $Y \in \{N, Ps\}$.

The following theorem stated in [2] only for the max-mode obviously hols for the $smax$-mode, too.

Theorem 7. *For $\beta \in \{max, smax\}$,*

$$PsRE = Ps_{\beta,gen}O_{tox}P_1([p]cat_2).$$

In general, we can formulate the following "metatheorem":

Metatheorem: *Whenever a proof has been established for the derivation mode max and literally also holds true for the derivation mode smax, then omitting trap rules by using the concept of toxic objects works for both derivation modes in the same way.*

In the following sections, we now turn our attention to models of P systems where the derivation mode *smax* yields different, in fact, stronger results than the derivation mode *max*.

8 Atomic Promoters and Inhibtors

As shown in [11], P systems with non-cooperative rules and atomic inhibitors are not computationally complete when the maximally parallel derivation mode is used. P systems with non-cooperative rules and atomic promoters can at least generate $PsET0L$. On the other hand, already in [10], the computational completeness of P systems with non-cooperative rules and atomic promoters has been shown. In the following we will establish a new proof for the simulation of a register machine where the overall number of promoters only depends on the number of decrementable registers of the register machine. Moreover, we also show a new pretty surprising result, establishing computational completeness of P systems with non-cooperative rules and atomic inhibitors, and the number of inhibitors again only depends on the number of decrementable registers of the simulated register machine. Finally, in both cases, if the register machine is deterministic, then the P system is deterministic, too.

8.1 Atomic Promoters

We now establish our new proof for the computational completeness of P systems with non-cooperative rules and atomic promoters when using the derivation mode *smax*; the overall number of promoters only is $5m$ where m is the number of decrementable registers of the simulated register machine.

Theorem 8. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a P system with atomic inhibitors*

$$\Pi = (O, \mu = [\ \ ]_1, w_1 = l_0, R_1, f = 1)$$

working in the smax- or max_{rule}-derivation mode and simulating the computations of M such that

$$|R_1| \leq ADD^1(R) + 2 \times ADD^2(R) + 5 \times SUB(R) + 7 \times m,$$

where $ADD^1(R)$ denotes the number of deterministic ADD-instructions in R, $ADD^2(R)$ denotes the number of non-deterministic ADD-instructions in R, and $SUB(R)$ denotes the number of SUB-instructions in R; moreover, the number of atomic inhibitors is $5m$. Finally, if the register machine is deterministic, then the P system is deterministic, too.

Proof. The numbers of objects o_r represent the contents of the registers r, $1 \leq r \leq d$; moreover, we denote $B_{SUB} = \{p \mid p : (SUB(r), q, s) \in R\}$.

$$O = \{o_r \mid 1 \leq r \leq d\} \cup \{o'_r, c_r, c'_r, c''_r, c'''_r \mid 1 \leq r \leq m\}$$
$$\cup (B \setminus \{l_h\}) \cup \{p', p'', p''' \mid p \in B_{SUB}\}$$

The symbols from $\{o'_r, c_r, c'_r, c''_r, c'''_r \mid 1 \leq r \leq m\}$ are used as promoters.

An ADD-instruction $p : (ADD(r), q, s)$ is simulated by the two rules $p \to qo_r$ and $p \to so_r$.

A SUB-instruction $p : (SUB(r), q, s)$ is simulated in four steps as follows:

1. $p \to p'c_r$;
2. $p' \to p''c'_r$; $o_r \to o'_r \mid_{c_r}$, $c_r \to \lambda$;
3. $p'' \to p'''c'''_r$, $c'_r \to c''_r \mid_{o'_r}$, $o'_r \to \lambda$;
4. $p''' \to q \mid_{c''_r}$, $p''' \to s \mid_{c'_r}$, $c'_r \to \lambda \mid_{c'''_r}$, $c''_r \to \lambda$, $c'''_r \to \lambda$.

As final rule we could use $l_h \to \lambda$, yet we can omit this rule and replace every appearance of l_h in all rules as described above by λ. □

8.2 Atomic Inhibtors

We now show that even P systems with non-cooperative rules and atomic promoters using the derivation mode *smax* can simulate any register machine needing only $2m + 1$ inhibitors where m is the number of decrementable registers of the simulated register machine.

Theorem 9. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a P system with atomic inhibitors*

$$\Pi = (O, \mu = [\ ]_1, w_1 = l_0, R_1, f = 1)$$

a P system with atomic inhibitors $\Pi = (O, \mu = [\ ]_1, w_1 = l_0, R_1, f = 1)$ working in the smax- or max_{rule}-derivation mode and simulating the computations of M such that

$$|R_1| \leq ADD^1(R) + 2 \times ADD^2(R) + 5 \times SUB(R) + 3 \times m + 1,$$

where $ADD^1(R)$ denotes the number of deterministic ADD-instructions in R, $ADD^2(R)$ denotes the number of non-deterministic ADD-instructions in R, and $SUB(R)$ denotes the number of SUB-instructions in R; moreover, the number of atomic inhibitors is $2m + 1$. Finally, if the register machine is deterministic, then the P system is deterministic, too.

Proof. The numbers of objects o_r represent the contents of the registers r, $1 \leq r \leq d$. The symbols d_r prevent the register symbols o_r, $1 \leq r \leq m$, from evolving.

$$O = \{o_r \mid 1 \le r \le d\} \cup \{o'_r \mid 1 \le r \le m\} \cup \{d_r \mid 0 \le r \le m\}$$
$$\cup (B \setminus \{l_h\}) \cup \{p', p'', \tilde{p} \mid p \in B_{SUB}\}$$

We denote $D = \prod_{i=1}^{m} d_i$ and $D_r = \prod_{i=1, i \neq r}^{m} d_i$.

An ADD-instruction $p : (ADD(r), q, s)$ is simulated by the two rules $p \to qo_rD$ and $p \to so_rD$.

A SUB-instruction $p : (SUB(r), q, s)$ is simulated in four steps as follows:

1. $p \to p'D_r$;
2. $p' \to p''Dd_0$; in parallel, the following rules are used:
   $o_r \to o'_r \mid_{\neg d_r}$, $d_k \to \lambda$, $1 \le k \le m$;
3. $p'' \to \tilde{p}D \mid_{\neg o'_r}$; $o'_r \to \lambda$, $d_0 \to \lambda$;
   again, in parallel the rules $d_k \to \lambda$, $1 \le k \le m$, are used;
4. $p'' \to qD \mid_{\neg d_0}$, $\tilde{p} \to sD$.

As final rule we could use $l_h \to \lambda$, yet we can omit this rule and replace every appearance of l_h in all rules as described above by λ. □

9 P Systems with Target Selection

In P systems with target selection, all objects on the right-hand side of a rule must have the same target, and in each derivation step, for each region a (multi)set of rules – non-empty if possible – having the same target is chosen. We show that for P systems with target selection in the derivation mode *smax no* catalyst is needed any more, and with max_{rule}, we even obtain a deterministic simulation of deterministic register machines.

Theorem 10. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \le d$ being the number of decrementable registers, we can construct a P system with non-cooperative rules working in the smax-derivation mode and simulating the computations of M.*

Proof. As usual, we take an arbitrary register machine M with d registers satisfying the following conditions: the output registers are $m+1, \cdots, d$, and they are never decremented; moreover, registers $1, \cdots, m$ are empty in any reachable halting configuration. Clearly, these conditions do not restrict the generality. We construct the following P system Π simulating M.

The correct behavior of the object associated to the simulated instruction of M is the following. In the decrement case, we have $in_r + 2$, *out*, in_2, idle, *out*, in_2, *here*, *out*, *here* (9 steps in total), whereas in the zero-test case, we have the same as before, except that the fourth and the fifth steps are *out* and *here* instead of idle and *out*, respectively. In case of an increment instruction, we get *here*, *here*, *here*, *here*, in_2, *here*, *out*, *here* (8 steps in total). We remark that the first four steps are carried out in the skin, while the last four steps repeat the cases of zero-test and decrement.

The value of each register r is represented by the multiplicity of objects o_r in the skin. For every decrementable register r, there is a rule sending o_r into region $r+2$. However, this rule may only be applied safely in the first step of the simulation of the SUB instruction, as otherwise some other object will also enter the same region as # (either one of e, e', e'', $\hat{e}$, $\hat{e}'$, which we will in the following refer to as the *guards*, or an object associated to the label of the simulated instruction, which we will in the following call a *program symbol*) forcing an unproductive computation, see the rules in brackets in the tables below.

The "correct" target selection for the inner regions normally coincides with that of the program symbol (described above) and no rule is applied there if the program symbol is not there, with the following exceptions. In the first step of simulating an instruction, object e exits membrane 2, as it is the only rule applicable there in this step. In the last step of simulating an instruction, object $\bar{e}$ is rewritten into e in membrane 2, as it is the only rule applicable there in this step. In the fourth step of the decrement case, the program symbol is idle while object d is erased. The "correct" target selection for the skin coincides with that of the program symbol, and is *here* if the program symbol is missing in the skin.

$$
\begin{aligned}
\Pi &= (O, \mu, w_1, \cdots, w_{m+2}, R_1, \cdots, R_{m+2}) \text{ where}\\
O &= \{o_r \mid 1 \le r \le d\} \cup \{\bar{p}, p \mid p \in B\} \cup \{p', p''\hat{p} \mid p \in B_{ADD}\}\\
&\quad \cup \{p', p_-, p'_-, p_0, p'_0, p''_0 \mid p \in B_{SUB}\} \cup \{\bar{e}, e, e', e'', \hat{e}, \hat{e}', d, \#\},\\
\mu &= [\,[\;\;]_2 \cdots [\;\;]_{m+2}\,]_1,\\
w_1 &= l_0,\ w_2 = e,\ w_{r+2} = \lambda,\ 1 \le r \le m,\\
R_1 &= \bigcup_{i=1}^{m+2} (R_{1,i,s} \cup R_{1,i,\#}),\\
R_i &= R_{i,1,s} \cup R_{i,1,\#} \cup R_{i,i,s} \cup R_{i,i,\#},\ 2 \le j \le m+2,\\
R_{1,1,s} &= \{e \to e', e' \to e'', e'' \to \hat{e}, \hat{e} \to \hat{e}', e' \to \lambda\}\\
&\quad \cup \{p'_0 \to p''_0 \mid p \in B_{SUB}\} \cup \{\bar{p} \to p \mid p \in B\}\\
&\quad \cup \{p \to \tilde{p}o_r \mid p : (ADD(r), q, s) \in P\}\\
&\quad \cup \{\tilde{p} \to p', p' \to p'', p'' \to \hat{p} \mid p \in B_{ADD}\},\\
R_{1,2,s} &= \{p' \to (p_-, in_2), p' \to (p_0, in_2), p'_- \to (p'_-, in_2), p''_0 \to (p''_0, in_2)\\
&\quad \mid p \in B_{SUB}\} \cup \{\hat{p} \to (\hat{p}, in_2) \mid p \in B_{ADD}\} \cup \{d \to (d, in_2)\}\\
R_{1,r+2,s} &= \{o_r \to (o_r, in_{r+2})\} \cup \{p \to (p, in_{r+2})\\
&\quad \mid p : (SUB(r), q, s) \in P\},\ 1 \le r \le m,\\
R_{1,1,\#} &= \{p' \to \#, p''_0 \to \#, p'_- \to \# \mid p \in B_{SUB}\} \cup \{\hat{p} \to \# \mid p \in B_{ADD}\}\\
&\quad \cup \{\# \to \#\},\\
R_{1,2,\#} &= \{p'_0 \to (\#, in_2), e'' \to (\#, in_2) \mid p \in B_{SUB}\}\\
&\quad \cup \{\bar{p} \to (\#, in_2) \mid p \in B\},\\
R_{1,r+2,\#} &= \{x \to (\#, in_{r+2}\} \mid x \in \{e, e', e'', \hat{e}, \hat{e}'\}\\
&\quad \cup \{p'_0, p'_- \mid P \in B_{SUB}\} \cup \{\bar{p} \mid P \in B\}\}\\
&\quad \cup \{p \to (\#, in_{r+2}) \mid p : (SUB(i), q, s) \in P,\ i \ne r\}\\
&\quad \cup \{p' \to (\#, in_{r+2}) \mid p \in B_{SUB}\},\ 1 \le r \le m,\\
R_{2,1,s} &= \{e \to (e, out)\} \cup \{\bar{p} \to (\bar{p}, out) \mid p \in B\}\\
&\quad \cup \{p_0 \to (p'_0, out), p_- \to (p'_-, out) \mid p \in B_{SUB}\},\\
R_{2,2,s} &= \{d \to \lambda, \bar{e} \to e\} \cup \{\mid p \in B\}\\
&\quad \cup \{p''_0 \to \bar{s}\bar{e}, p'_- \to \bar{q}\bar{e} \mid p : (SUB(r), q, s) \in P\}\\
&\quad \cup \{\hat{p} \to \bar{q}\bar{e}, \hat{p} \to \bar{s}\bar{e} \mid p : (ADD(r), q, s) \in P\},\\
R_{2,1,\#} &= \{d \to (\#, out), \# \to (\#, out)\},\\
R_{2,2,\#} &= \{p_0 \to \# \mid p \in B_{SUB}\} \cup \{\bar{p} \to \# \mid p \in B\},\\
R_{r+2,1,s} &= \{p \to (p', out) \mid p \in B_{SUB}\} \cup \{o_r \to (d, out\},\ 1 \le r \le m\\
R_{r+2,r+2,\#} &= \{\# \to (\#, out)\},\ R_{r+1,r+1,s} = R_{r+1,r+1,\#} = \emptyset.
\end{aligned}
$$

Most trapping rules, given in brackets in the tables below and listed in rule groups $R_{i,j,\#}$ above, are only needed to force the "correct" target selection. The exception are some rules in steps 4 and 5 of the simulation of SUB instructions,

needed for verifying that the decrement and the zero test have been performed correctly (the guess is made at step 3 by the program symbol, and is reflected in its subscript). Indeed, if the zero-test is chosen while d is present (signifying that the register was decremented), causing a target conflict: either p_0 or d will be anyway rewritten into #. However, if the decrement is chosen while d is absent (signifying that the register was zero), then p_- will appear in the skin in step 4 instead of step 5, causing a target conflict: either p'_- or e'' will be anyway rewritten into #.

Below we present the tables describing the simulation of instructions of M. An application of one of the rules given in brackets leads to non-halting computations, not contributing to the result.

$(p : (SUB(r), q, s))$

	$r+2$	1	2
1	- -	$o_r \to (o_r, in_{r+2})$ $p \to (p, in_{r+2})$ $[p \to (\#, in_{i+2}),\ i \neq r]$	$e \to (e, out)$
2	$p \to (p', out)$ $o_r \to (d, out)$	$e \to e'$ $[e \to (\#, in_{i+2})]$	-
3	-	$p' \to (p_-, in_2)$ $p' \to (p_0, in_2)$ $d \to (d, in_2)$ $[p' \to \#]$ $[e' \to (\#, in_{i+2})]$	-

	1,-	1,0	2,-	2,0
4	$e' \to e''$		$d \to \lambda$ $[p_- \to (p'_-, out)]$	$p_0 \to (p'_0, out)$ $[d \to (\#, out)]$ $[p_0 \to \#]$
5	$e'' \to \hat{e}$ $[p'_- \to (p'_-, in_2)]$ $[p'_- \to \#]$ $[e'' \to (\#, in_t)]$ [for $t > 1$]	$p'_0 \to p''_0$ $e'' \to \hat{e}$ $[p'_0 \to (\#, in_t)]$ $[e'' \to (\#, in_t)]$ [for $t > 1$]	$p_- \to (p'_-, out)$	-
6	$p'_- \to (p'_-, in_2)$ $[p'_- \to \#]$ $[p'_- \to (\#, in_{i+2})]$	$p''_0 \to (p''_0, in_2)$ $[p''_0 \to \#]$ $[p''_0 \to (\#, in_{i+2})]$	-	
7	$\hat{e} \to \hat{e}'$ $[\hat{e} \to (\#, in_{i+2})]$		$p'_- \to \bar{q}\bar{e}$	$p''_0 \to \bar{s}\bar{e}$
8	$\hat{e}' \to \lambda$ $[\hat{e}' \to (\#, in_{i+2})]$		$\bar{q} \to (\bar{q}, out)$ $[\bar{q} \to \#]$	$\bar{s} \to (\bar{s}, out)$ $[\bar{s} \to \#]$
9	$\bar{q} \to q$ $[\bar{q} \to (\#, in_t)]$	$\bar{s} \to s$ $[\bar{s} \to (\#, in_t)]$	$\bar{e} \to e$	

$(p : (ADD(r), q, s))$

	1	2
1	$p \to \tilde{p}o_r$	$e \to (e, out)$
2	$\tilde{p} \to p'$ $e \to e'$	-
3	$p' \to p''$ $e' \to e''$	-
4	$p'' \to \hat{p}$ $e'' \to \hat{e}$	-
5	$\hat{p} \to (\hat{p}, in_2)$ $[\hat{p} \to \#]$	-
6	$\hat{e} \to \hat{e}'$	$\hat{p} \to \bar{x}\bar{e}$
7	$\hat{e}' \to \lambda$	$\bar{x} \to (\bar{x}, out)$ $[\bar{x} \to \#]$
8	$\bar{x} \to x$	$\bar{e} \to e$

Auxiliary rules

$r+2$	1	2
$[\# \to (\#, out)]$	$[\# \to \#]$	$[\# \to (\#, out)]$

Nearly half of the steps in the preceding constructions is needed for releasing the auxiliary symbol e in the first step of a simulation from membrane 2, yet in our construction, e and its derivatives are needed to control the correct target selection in the skin membrane, and especially to keep the register objects o_r from moving into membrane $r+2$. □

We now show that taking the maximal sets of rules which are applicable, the simulation of SUB-instructions can even be carried out in a deterministic way.

Theorem 11. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a P system with non-cooperative rules*

$$\Pi = (O, \mu = [\ [\ ]_2 \dots [\ ]_{2m+1}\ ]_1, w_1, \lambda, \dots, \lambda, R_1 \dots R_{2m+1}, f = 1)$$

working in the max_{rule}-derivation mode and simulating the computations of M such that

$$|R_1| \leq 1 \times ADD^1(R) + 2 \times ADD^2(R) + 4 \times SUB(R) + 10 \times m + 3,$$

where $ADD^1(R)$ denotes the number of deterministic ADD-instructions in R, $ADD^2(R)$ denotes the number of non-deterministic ADD-instructions in R, and $SUB(R)$ denotes the number of SUB-instructions in R.

Proof. The contents of the registers r, $1 \leq r \leq d$ is represented by the numbers of objects o_r, and for the decrementable registers we also use a copy of the symbol o'_r for each copy of the object o_r. This second copy o'_r is needed during the simulation of SUB-instructions to be able to distinguish between the decrement and the zero test case. For each r, the two objects o_r and o'_r can only be affected by the rules $o_r \to (\lambda, in_{r+1})$ and $o'_r \to (\lambda, in_{r+1})$ sending them into the membrane $r+1$ corresponding to membrane r (and at the same time erasing them; in fact, we could also leave them in the membrane unaffected forever as a garbage). These are already two rules, so any other combination of rules with different targets has to contain at least three rules.

One of the main ideas of the proof construction is that in the skin membrane the label p of an ADD-instruction is represented by the three objects p and e, e', and the label p of any SUB-instruction is represented by the eight objects $p, e, e', e'', d_r, d'_r, \tilde{d}_r, \tilde{d}_r{}'$. Hence, for each $p \in (B \setminus \{l_h\})$ we define $R(p) = pee'$ for $p \in B_{ADD}$ and $R(p) = pee'e''d_r d'_r \tilde{d}_r \tilde{d}_r{}'$ for $p \in B_{SUB}$ as well as $R(l_h) = \lambda$; as initial multiset w_1 in the skin membrane, we take $R(l_0)$.

$$O = \{o_r \mid 1 \leq r \leq d\} \cup \{o'_r \mid 1 \leq r \leq m\} \cup (B \setminus \{l_h\}) \\ \cup \left\{d_r, d'_r, \tilde{d}_r, \tilde{d}_r{}' \mid 1 \leq r \leq m\right\} \cup \{e, e', e''\}$$

An ADD-instruction $p : (ADD(r), q, s)$ is simulated by the rules $p \to R(q)o_r$ and $p \to R(s)o_r$ as well as the rules $e \to \lambda$ and $e' \to \lambda$. This combination of three rules supercedes any combination of rules $o_r \to (\lambda, in_{r+1})$ and $o'_r \to (\lambda, in_{r+1})$, for some $1 \leq r \leq m$.

A SUB-instruction $p : (SUB(r), q, s)$ is simulated in two steps as follows:

1. In R_1, for the first step we take one of the following tuple of rules
   $p \to (p, in_{r+1})$, $d_r \to (\lambda, in_{r+1})$, $d'_r \to (\lambda, in_{r+1})$, $\tilde{d}_r \to (\lambda, in_{r+1})$,
   $o_r \to (\lambda, in_{r+1})$, $o'_r \to (\lambda, in_{r+1})$;
   $p \to (p, in_{m+r+1})$, $d_r \to (\lambda, in_{m+r+1})$, $d'_r \to (\lambda, in_{m+r+1})$,
   $\tilde{d}_r \to (\lambda, in_{m+r+1})$, $\tilde{d}_r{}' \to (\lambda, in_{m+r+1})$;
   the application of the rules $o_r \to (\lambda, in_{r+1})$, $o'_r \to (\lambda, in_{r+1})$ in contrast to the application of the rule $\tilde{d}_r{}' \to (\lambda, in_{m+r+1})$ determines whether the first or the second tuple of rules has to be chosen. Here it becomes clear why we have to use the two register symbols o_r and o'_r, as we have to guarantee that the target $r+1$ cannot be chosen if none of these symbols is present, as in this case then only four rules could be chosen in contrast to the five rules for the zero test case. On the other hand, if some of these symbols o_r and o'_r are present, then six rules are applicable superceding the five rules which could be used for the zero test case.
2. In the second step, the following three or four rules, again superceding any combination of rules $o_r \to (\lambda, in_{r+1})$ and $o'_r \to (\lambda, in_{r+1})$ for some $1 \leq r \leq m$, are used in the skin membrane:
   $e \to \lambda$, $e' \to \lambda$, $e'' \to \lambda$, and in the decrement case also the rule $\tilde{d}_r{}' \to \lambda$.

In the second step, we either find the the symbol p in membrane $r+1$, if a symbol o_r together with its copy o'_r has been present for decrementing or in membrane $m+r+1$, if no symbol o_r has been present (zero test case).
In the decrement case, the following rule is used in R_{r+1}: $p \to (R(q), out)$.
In the zero test case, the following rule is used in R_{m+r+1}: $p \to (R(s), out)$.

We finally point out that the simulation of the SUB-instructions works deterministically, hence, although the P system itself is not deterministic syntacticly, it works in a deterministic way if the underlying register machine is deterministic. □

10 Conclusion and Future Work

It is not very surprising that the proofs we have checked in the preceding sections also work for the derivation mode *smax*, as many constructions elaborated for the derivation mode *max* just "break down" maximal parallelism to near sequentiality in order to work for the simulation of register machines. On the other hand, we also have shown that due to this fact some variants of P systems become even stronger with the modes *smax* and max_{rule}.

- There are many models of P systems for which the maximally parallel derivation mode has been used, especially for showing computational completeness.
- As we have seen by careful inspection of several proofs for computational completeness, many results established with using the maximally parallel derivation mode literally hold true as well for the derivation modes *smax* and max_{rule}.
- Many other constructions working in the maximally parallel derivation mode have to be checked carefully if they work for the derivation modes *smax* and max_{rule}, too.
- For some proofs having been established in the maximally parallel derivation mode we might need completely new proofs or proof techniques for the set-maximally parallel derivation mode; one such example is the proof for P systems with target selection.
- Some variants of P systems become even stronger with the mode *smax*; as already pointed out by Gheorghe Păun, P systems with non-cooperative rules and atomic promoters are computationally complete with the *smax*-mode, also see [10], and in this paper we have shown a new proof for this computational completeness result and even shown a similar result for P systems with non-cooperative rules and atomic inhibitors.
- On the other hand, eventually, some results stablished in the maximally parallel derivation mode are not valid any more for the set-maximally parallel derivation mode.

References

1. A. Alhazov, B. Aman, R. Freund, and Gh. Păun. Matter and anti-matter in membrane systems. In *Proceedings of the Twelfth Brainstorming Week on Membrane Computing*, pages 1–26, 2014.
2. A. Alhazov and R. Freund. P systems with toxic objects. In M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, and C. 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 99–125. Springer, 2014.
3. A. Alhazov and R. Freund. Small catalytic P systems. In M. J. Dinneen, editor, *Proceedings of the Workshop on Membrane Computing 2015 (WMC2015), (Satellite workshop of UCNC2015), August 2015*, volume CDMTCS-487 of *CDMTCS Research Report Series*. Centre for Discrete Mathematics and Theoretical Computer, ScienceDepartment of Computer Science University of Auckland, Auckland, New Zealand, 2015.
4. A. Alhazov, R. Freund, H. Heikenwälder, M. Oswald, Yu. Rogozhin, and S. Verlan. Sequential P systems with regular control. In E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, and Gy. 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 112–127. Springer, 2013.
5. H. Burkhard. Ordered firing in petri nets. *Elektronische Informationsverarbeitung und Kybernetik*, 17(2/3):71–86, 1981.
6. R. Freund and Gh. Păun. How to obtain computational completeness in P systems with one catalyst. In T. Neary and M. Cook, editors, *Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9-11, 2013*, volume 128 of *EPTCS*, pages 47–61, 2013.
7. R. Freund and S. Verlan. A formal framework for static (tissue) P systems. In G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing. 8th International Workshop, WMC 2007 Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer, 2007.
8. P. Frisco and G. Govan. P systems with active membranes operating under minimal parallelism. In M. Gheorghe, Gh. Păun, G. Rozenberg, A. Salomaa, and S. Verlan, editors, *Membrane Computing - 12th International Conference, CMC 2011, Fontainebleau, France, August 23-26, 2011, Revised Selected Papers*, volume 7184 of *Lecture Notes in Computer Science*, pages 165–181. Springer, 2011.
9. K. Krithivasan, Gh. Păun, and A. Ramanujan. On controlled P systems. In L. Valencia-Cabrera, M. García-Quismondo, L. Macías-Ramos, M. Martínez-del-Amor, Gh. Păun, and A. Riscos-Núñez, editors, *Proceedings 11th Brainstorming Week on Membrane Computing, Sevilla, 4–8 February 2013*, pages 137–151. Fénix Editora, Sevilla, 2013.
10. L. Pan, Gh. Păun, and B. Song. Flat maximal parallelism in P systems with promoters. *Theoretical Computer Science*, 2015, to appear.
11. D. Sburlan. Further results on P systems with promoters/inhibitors. *International Journal of Foundations of Computer Science*, 17(1):205–221, 2006.
12. P. Sosík and M. Langer. Small catalytic P systems simulating register machines. *Theoretical Computer Science*, accepted, 2015.

13. S. Verlan and J. Quiros. Fast hardware implementations of P systems. In E. Csuhaj-Varjú, M.Gheorghe, G. Rozenberg, A. Salomaa, and Gy. 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 404–423. Springer, 2013.

Verifying P Systems with Costs by Using Priced-Timed Maude

Bogdan Aman and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science, Iaşi, Romania
E-mail: bogdan.aman@gmail.com, gabriel@info.uaic.ro

Summary. We consider P systems that assigns storage costs per step to membranes, and execution costs to rules. We present an abstract syntax of the new class of membrane systems, and then deal with costs by extending the operational semantics of P systems with promoters, inhibitors and registers. We use Priced-Timed Maude to implement the P systems with costs. By using such a rewriting engine which corresponds to the semantics of membrane systems with costs, we are able to prove the operational correctness of this implementation. Based on such an operational correspondence, we can analyze properly the evolutions of the P systems with costs, and verify several reachability properties, including the cost of computations that reach a given membrane configuration. This approach opens the way to various optimization problems related to membrane systems, problems making sense in a bio-inspired model which now can be verified by using a complex software platform.

1 Introduction

Membrane computing is introduced in [10] and represents now a well known branch of natural computing that aims to abstract computing ideas and formal models from the structure and functioning of living cells, as well as from the organization of cells in tissues, organs or other higher order structures such as colonies of cells [11]. Membrane systems (known also as P systems) are parallel and distributed models working with multisets of symbols in cell-like compartmental architectures. The existing results in membrane computing refer mainly to the P systems characterization of Turing computability, providing also some polynomial solutions to NP-complete problems by using an exponential workspace created in a "biological way".

Time was introduced and studied in the framework of membrane systems [2]. However, time is not the only quantitative notion of interest; other quantities such as energy [8] or accumulated cost can be included in such systems. The notions of energy and cost are connected to (evolution) time, because the longer the system evolves, the higher the energy and costs are. For simplicity, in this paper we study

only the (evolution) costs in a membrane system. A membrane system with costs is essentially a simple membrane system in which object storage costs per evolution step are assigned to membranes, and execution costs are assigned to rules. Notice that here we consider the cost only as an external/observer variable, and thus whether a rule is applicable only depends on available resources (not cost value). In this paper we present an abstract syntax of the membrane systems with costs, and then define a structural operational semantics of P systems with costs. We use a rewriting engine called Priced-Timed Maude to implement these P systems. After proving an operational correctness of this implementation, we can analyze properly the evolutions of the P systems involving costs, We look at the cost of computations reaching a given membrane configuration. This paper represents a first step towards a more detailed analysis of various costs in the context of membrane systems.

This class of P systems with costs differs from energy-based P systems [8], a model of membrane systems whose computations occur by manipulating the energy associated to the objects, as well as the free energy units occurring inside the regions of the system. In [8] the energy units are used to transform objects, while in this paper the costs are used only to compute the evolution cost, and eventually to return an optimal evolution with respect to its cost.

2 Membrane Systems with Costs

Before describing in a formal way the evolution of a P system with costs, we present first an inductive definition of the membrane structure, the sets of configurations, and a definition for the corresponding transition systems.

Configurations are states of a transition system, and a computation consists of sequences of transitions between configurations terminating (if it terminates) in a final configuration. A sequence of transition steps represents a *computation*. A computation is successful if this sequence is finite, namely there is no rule applicable to the objects present in the last committed configuration. In a halting committed configuration, the result of a successful computation is the total number of objects present either in the membrane considered as the output membrane, or in the outer region.

In general,operational semantics provides a framework for defining a formal description of a computing system. It is intuitive and flexible, and it becomes more attractive during the years by the developments presented in [12] and [9]. In basic P systems, a computation is regarded as a sequence of parallel applications of rules in various membranes, followed by a communication step and a dissolving step. The operational semantics of the P systems emphasises the deductive nature of the membrane computing by describing the transition steps by using a set of inference rules [3]. The operational semantics of P systems is implemented by using the cost extension of the rewriting system called Maude [7]. The relationship between the operational semantics of P systems and Maude rewriting is given by certain operational correspondence results.

Let O be a finite alphabet of objects over which we consider the *free commutative monoid* O_c^*, whose elements are *multisets*. The empty multiset is denoted by *empty*. Objects can be enclosed in messages together with a target indication. We have *here* messages of typical form $(w, here)$, *out* messages (w, out), and *in* messages (w, in_L). For the sake of simplicity, hereinafter we consider that the messages with the same target indication merge into one message:
$\prod_{i\in I}(v_i, here) = (w, here)$, $\prod_{i\in I}(v_i, in_L) = (w, in_L)$, $\prod_{i\in I}(v_i, out) = (w, out)$, with $w = \prod_{i\in I} v_i$, I a non-empty set, and $(v_i)_{i\in I}$ a family of multisets over O.

We use the mapping rules to associate to a membrane label the set of evolution rules: $\mathsf{rules}(L_i) = R_i$, and the projections L, w and c which return from a membrane its label, its current multiset, and its cost, respectively.

The set $\mathcal{M}(\Pi)$ *of membranes for a P system with costs* Π, and *the membrane structures* are inductively defined as follows:

- if L is a label, c is a cost and w is a multiset over $O \cup (O \times \{here\}) \cup (O \times \{out\}) \cup \{\delta\}$, then $\langle L; c | w \rangle \in \mathcal{M}(\Pi)$; $\langle L; c | w \rangle$ is called *simple (or elementary) membrane*, and it has the structure $\langle\rangle$;
- if L is a label, c is a cost and w is a multiset over $O \cup (O \times \{here\}) \cup (O \times \{in_{L(M_j)} | \, j \in [n]\}) \cup (O \times \{out\}) \cup \{\delta\}$, $M_1, \ldots, M_n \in \mathcal{M}(\Pi)$, $n \geq 1$, where each membrane M_i has the structure μ_i, then $\langle L; c | w; M_1, \ldots, M_n \rangle \in \mathcal{M}(\Pi)$; $\langle\, L\,;c\,|\,w\,;\, M_1, \ldots, M_n\,\rangle$ is called *a composite membrane* having the structure $\langle \mu_1, \ldots, \mu_n \rangle$.

We conventionally suppose the existence of a set of sibling membranes denoted by $NULL$ such that $M, NULL = M = NULL, M$ and $\langle L | w; NULL \rangle = \langle L | w \rangle$. The use of $NULL$ significantly simplifies several definitions and proofs. Let $\mathcal{M}^*(\Pi)$ be the free commutative monoid generated by $\mathcal{M}(\Pi)$ with the operation $(\_, \_)$ and the identity element $NULL$. We define $\mathcal{M}^+(\Pi)$ as the set of elements from $\mathcal{M}^*(\Pi)$ without the identity element. Let M_+, N_+ range over non-empty sets of sibling membranes, M_i over membranes, M_*, N_* range over possibly empty multisets of sibling membranes, and L over labels. The membranes preserve the initial labelling, cost and evolution rules in all subsequent configurations. Therefore in order to describe a membrane we consider its label, its cost and the current multiset of objects together with its structure.

A *configuration* for a P system with costs Π is a skin membrane which has no messages and no dissolving symbol δ, i.e., the multisets of all regions are elements in O_c^*. We denote by $\mathcal{C}(\Pi)$ the set of configurations for Π.

An *intermediate configuration* is an arbitrary skin membrane in which we may find messages or the dissolving symbol δ. We denote by $\mathcal{C}^{\#}(\Pi)$ the set of intermediate configurations. We have $\mathcal{C}(\Pi) \subseteq \mathcal{C}^{\#}(\Pi)$.

Each P system with costs has an *initial configuration* which is characterized by the initial multiset of objects for each membrane and the initial membrane structure of the system. For two configurations C_1 and C_2 of Π, we say that there is a *transition* from C_1 to C_2, and write $C_1 \Rightarrow C_2$, if the following *steps* are executed in the given order:

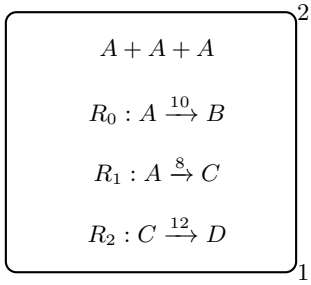
1. *maximal parallel rewriting step* as in [6];

2. *parallel communication of objects through membranes*;
3. *parallel membrane dissolving* of the membranes containing δ.

The last two steps take place only if there are messages or δ symbols resulting from the first step, respectively. If the first step is not possible, then neither are the other two steps; we say that the system has reached a *halting configuration*.

To illustrate these notions we give in Figure 1 a small example containing only of a single membrane labelled by 1 and with associated cost 2. This membrane contains three objects A and three rules with different assigned costs (rule R_0 with cost 10, rule R_1 with cost 8 and rule R_2 with cost 12). A possible evolution leads in just one step to a configuration consisting of three objects B and with the cost of evolution of 30 resulting from applying three times rule R_0.

Fig. 1. A Small P System with Costs



3 Implementing P Systems with Costs by using Maude

Reasoning about the accumulated cost (of energy usage, for instance) during behaviours is crucial in biological and embedded systems (e.g., wireless sensor networks) where minimizing overall consumed resources is critical. Generally, by using a rewriting engine called Maude, a formal specification of a system can be automatically transformed into an interpreter. Moreover, Maude provides an useful new extension called Priced-Timed Maude [4] supporting the formal specification and analysis of systems in which the cost of performing actions plays a significant role. The tool offers a search command, a semi-decision procedure for finding failures of safety properties, and also a model checker. Since the P systems with costs combine the power of parallel rewriting in various locations (compartments), the power of local and contextual evolution and the use of rewriting costs, it is natural to use a rewriting engine and a rewrite theory.

Roughly speaking, a rewrite theory is a triple $(\Sigma, E, \mathcal{R}, \mathcal{L})$, where (Σ, E) is an equational theory used for implementing the deterministic computation, therefore (Σ, E) should be terminating and Church-Rosser, $\mathcal{R}$ is a set of rewrite rules with costs used to implement nondeterministic and/or concurrent computations, and $\mathcal{L}$ is a set of tick rules which can model the time elapse in the system. Therefore we find rewriting logic suitable for implementing these membrane systems.

For simplicity, in this section we consider only costs added to the evolution rules. A P system consists of a maximal parallel application of the evolution rules, the (repeated) steps of internal evolution, communication, and dissolving. This sequence of steps uses a kind of synchronization. A P system has a tree like structure with the skin as its root, the composite membranes as its internal nodes and the elementary membranes as its leaves. The order of the children of a node is not important due to the associativity and commutativity properties of the concatenation operation $\_,\_$ of membranes.

In what follows we extend with costs the operational semantics of membrane systems with promoters, inhibitors [5] and registers [1]. We define an operational semantics of membrane systems by means of three sets of inference rules corresponding to maximal parallel rewriting, sending messages and dissolving. The notation $\mathcal{R} \vdash t \to t'$ is used to express that $t \to t'$ is provable in the theory $\mathcal{R}$ using the inference rules of rewriting logic. We use the syntax of the rewriting engine Maude extended for systems with costs [4] to describe a rewriting theory which corresponds faithfully to the semantics of membrane systems with costs.

In rewriting logic we describe a multiset of objects and messages as consisting of four "bags" of which three are multisets of objects (standing for objects which are actually in the membrane, objects with message *here*, objects with message *out*), and the fourth containing a multiset of pairs of objects and labels i which stand for objects with message in_i. This representation facilitates the rewriting logic specification because in this way there is no need for additional sorts with respect to messages. We first consider the following sorts:

```
sorts Obj ObjMultiset ObjAddressMultiset Label Rule RuleSet .
subsort Obj < ObjMultiset . subsort Rule < RuleSet .
```

By `emptyMO` and `emptyMAO` we denote the empty multiset of objects, respectively of objects with labels, and use `+` to denote the addition on both `ObjMultiset` and `ObjAddressMultiset`.

The multiset of objects with addresses is constructed through the operator
`op in : ObjMultiset Label -> ObjAddressMultiset`.
A rule is constructed through the operator

```
op _->_|_|_|_|_|_ : ObjMultiset ObjMultiset ObjMultiset
ObjAddressMultiset ObjMultiset ObjMultiset Cost -> Rule [ctor] .
```

The first slot is for the objects to be consumed (it is the left hand side of the rule); the second slot is for the objects produced with label "here"; the third slot is for the objects produced with label "out"; the fourth slot is for the objects produced with label "in_child"; the next two slots are for promoters respectively inhibitors. The last slot is used to give the cost of applying the rule. The operators which are used to manipulate the components of a rule are

```
ops lhs rhsHere rhsOut promoter inhibitor : Rule -> ObjMultiset .
op rhsIn : Rule -> ObjAddressMultiset .
op costOf : Rule -> Cost .
```

`rulesIn : Label -> RuleSet` is used to present the rules inside a membrane.

We work with register membranes even when implementing message passing and dissolving. This does not modify in any way the semantics. In what follows, all the membranes are register membranes, even when not explicitly stated.
A register membrane is constructed through the operator

```
op <_‘[_|_|_|_‘]_>_ : Label ObjMultiset ObjMultiset ObjMultiset
ObjAddressMultiset MembraneSet ObjMultiset -> Membrane [ctor] .
```

The first slot is for the label; the second slot is for the objects inside the membrane; the third slot is for the objects with label "here"; the fourth slot is for the objects with label "out"; the fifth slot is for the objects with label "in_child"; the sixth slot is for the set of children membranes; the last slot is for the register. The operators which are used to manipulate the components of a rule are

```
op labelOf : Membrane -> Label .
ops register here : Membrane -> ObjMultiset .
ops content out : MembraneSet -> ObjMultiset .
op inChildren : Membrane -> ObjAddressMultiset .
op children : Membrane -> MembraneSet .
```

Other operators are `_isIn_` which evaluates whether a multiset is contained in another multiset, `mprIrred`, `msgIrred`, `dissIrred`, `eraseDelta`, `emptyOut` and `emptyReg` whose names are self-explaining. We also use `labelsOf` to gather the membrane labels which appear in the right hand side of a rule, for the same purpose `membraneSetLabels` with respect to the membrane sets, and `subsetOf` to compare them. These last three functions are used only when evaluating whether a pair formed of a membrane M and a rule R is valid:

```
op valid : Membrane Rule -> Bool .
ceq valid(M, R) = true if lhs(R) isIn content(M) /\ promoter(R)
  isIn (content(M) + register(M)) /\ labelsOf(rhsIn(R)) subsetOf
  membraneSetLabels(children(M)) /\ if (inhibitor(R) =/= emptyMO)
  then (inhibitor(R) isIn (content(M) + register(M)) == false)
  else true fi .
eq valid(M, R) = false [owise] .
```

To separate the three stages of evolution of a membrane we use four tags:

```
sorts evolutionType State .
ops mpr msg diss end : -> evolutionType [ctor] .
op _;_ : MembraneSet evolutionType -> State [ctor] .
```

where `end` is used to stop the rewriting once the membrane has stopped evolving.

The maximal parallel rewriting of a membrane is given by the following rules, where the second one is executed with the cost of the corresponding rule:

```
crl [1] : M , MM ; mpr => M1 , MM ; mpr if
  MM =/= null /\ M ; mpr => M1 ; mpr /\ M =/= M1 .

crl [2] : < L [ W1 | W2 | W3 | A ] MM > W4 ; mpr =>
  < L [ W1 - lhs(R) | W2 + rhsHere(R) | W3 + rhsOut(R) | A
  + rhsIn(R) ] MM > (W4 + lhs(R) ) ; mpr with cost costOf(R)
```

```
  if mprIrred(MM) /\ R RR := rulesIn(L)
  /\ valid(< L [ W1 | W2 | W3 | A ] MM > W4, R) .

crl [3] : < L [ W1 | W2 | W3 | A ] MM > W4 ; mpr =>
  < L [ W1 | W2 | W3 | A ] MM1> W4 ; mpr if
  mprIrred(MM) == false /\ MM ; mpr => MM1 ; mpr /\ MM =/= MM1 .
```

These rules impose the following evolution: if in a membrane there is some mpr-reducible child membrane, then the membrane is replaced by a similar membrane which has that child rewritten (rules `crl [3]` and `crl [1]`); if a membrane has only mpr-irreducible children, all valid rules are applied one by one (rule `crl [2]`). When even the *skin* membrane is mpr-irreducible, the following rule is applied

```
crl [4] : M ; mpr => emptyReg(M) ; msg if labelOf(M) == 1
  /\ mprIrred(M) .
```

in order to empty the register and to begin the next evolution stage, that of the message sending.

This message sending stage is governed by the following rules:

```
crl [5] : M , MM ; msg => M1 , MM ; msg if
  MM =/= null /\ M ; msg => M1 ; msg /\ M =/= M1 .

crl [6] : < L [ W1 | W2 | W3 | A ] MM > W4 ; msg => if L == 1 then
  < L [ W1 + W2 + out(MM1) | emptyMO | emptyMO | emptyMAO ]
  emptyOut(sendIn(A, MM1)) > W4 ; msg else
  < L [ W1 + W2 + out(MM1) | emptyMO | W3 | emptyMAO ]
  emptyOut(sendIn(A, MM1)) > W4 ; msg fi
  if msgIrred(MM) == false /\ MM ; msg => MM1 ; msg /\ msgIrred(MM1) .

crl [7] : < L [ W1 | W2 | W3 | A ] MM > W4 ; msg => if L == 1 and
  W3 =/= emptyMO then < L [ W1 + W2 + out(MM) | emptyMO
  | emptyMO | emptyMAO ] emptyOut(sendIn(A, MM)) > W4;msg else
  < L [ W1 + W2 + out(MM) | emptyMO | W3 | emptyMAO ]
  emptyOut(sendIn(A, MM)) > W4 ; msg fi if msgIrred(MM)
  /\ (A =/= emptyMAO) or (W2 =/= emptyMO) or out(MM) =/= emptyMO .
```

In this stage a membrane evolves in a single rewriting step: if the set MM of children membranes is msg-reducible, then MM rewrites to a msg-irreducible $MM1$(rule `crl [5]`); the membrane M with objects $W1$ which contains MM is rewritten to the membrane $M1$ with objects $W1 + W2 + out(MM1)$ (i.e. the objects with messages of form $(a, here)$ are transformed in objects of form a, and the objects sent out by the set $MM1$ of membranes are added), and children $emptyOut(sendIn(A, MM1))$ (i.e. the objects of form (a, in_j) are sent into the membrane with label j and then the objects with messages of form (a, out) are erased from every child membrane). The result is msg-irreducible, because the only objects with messages are in the membrane $M1$, and they are of the form (a, out) (if $M1$ is the *skin* not even those objects remain). If the set MM of children membranes is msg-irreducible, then the same process takes place, except that instead of $MM1$ it is still MM(rule [7]).

Rules `crl [5]`, `crl [6]` and `crl [7]` correspond to inference rules $msg1$ and $msg2$. In defining the transition relation T_{msg} we treat the case of an elementary membrane separately, since we prefer to avoid extending T_{msg} to sets of membranes. Although rules `crl [6]` and `crl [7]` look almost identical, we cannot include them in a single rule with the conditional part `if MM ; msg => MM1 ; msg` because it would lead to an infinite loop of identical rewritings. This happens because $MM; msg \rightarrow MM; msg$ is provable in rewriting logic.

When the entire membrane system is msg-irreducible, the rule

```
crl [8] : M ; msg => M ; diss if labelOf(M) == 1 /\ msgIrred(M) .
```

is applied. This rule starts the next evolution stage, that of dissolving.

The rules for dissolving membranes are:

```
crl [9] : M , MM ; diss => M1 , MM ; diss if
  MM =/= null /\ M ; diss => M1 ; diss /\ M =/= M1 .

crl [10] : < L [ W1 | W2 | W3 | A ] MM > W4 ; diss =>
  < L [ W1 + eraseDelta(content(M)) | W2 | W3 | A ]
  children(M) , MM1 > W4 ; diss if dissIrred(MM) /\ M , MM2 := MM
  /\ delta isIn content(M) /\ MM1 := children(M) , MM2 .

crl [11] : < L [ W1 | W2 | W3 | A ] MM > W4 ; diss =>
  < L [ W1 | W2 | W3 | A ] MM1 > W4 ; diss
  if dissIrred(MM) == false /\ MM ; diss => MM1 ; diss
  /\ dissIrred(MM1) .
```

If the set MM of children membranes for a membrane M is diss-reducible and it rewrites to a diss-irreducible set of membranes $MM1$, then M is rewritten to the similar membrane $M1$ which has children membranes $MM1$ (rules `crl [9]` and `crl [11]`). When the set MM of children membranes is diss-irreducible and at least one of the membranes in MM contains the special symbol δ, then all the membranes from MM which contain δ are dissolved (rule `crl [10]`). Note that a top membrane M does not dissolve even when it does contain δ. This happens because the rewriting rules are given with the purpose of describing the evolution of the *skin* membrane, which can never dissolve. Rules `crl [9]`, `crl [10]` and `crl [11]` correspond to inference rules $msg1$ and $msg2$. Again, we have used the first rule in this group as a stepping stone towards the rewriting of a set of sibling membranes, while avoiding to include the rewriting of a set of sibling membranes in the transition relation T_{diss}.

When the *skin* membrane is diss-irreducible but is mpr-reducible, the rule

```
crl [12] : M ; diss => M ; mpr if labelOf(M) == 1
  /\ dissIrred(M) /\ mprIrred(M) == false .
```

is applied; it starts once more the maximal parallel rewriting stage of the evolution. However, if the *skin* membrane is also mpr-irreducible, rule

```
crl [13] : M ; diss => M ; end if   labelOf(M)==1
  /\dissIrred(M)/\mprIrred(M).
```

is applied; in this case it ends the rewriting. We do not need to evaluate the msg-irreducibility of the *skin* membrane, because the dissolving stage can only be reached by msg-irreducible membranes.

The correspondence between the operational semantics given by the transition relation $\Rightarrow$ on one hand, and the rewriting logic implementation on the other hand is is given by a mapping $\psi : \Pi \rightarrow \mathsf{State}$ defined by the natural encoding presented above. By $\mathcal{R}_\diamond$ we denote the rewrite theory defined by the rewrite rules `[1]` ... `[13]` together with the operators and equations defining them. The next theorem emphasizes the correspondence between the dynamics of the membrane systems with costs and the rewrite theory.

Theorem 1. $M \stackrel{c}{\Rightarrow} N$ *iff* $\mathcal{R}_D \vdash \psi(M) \Rightarrow^* \psi(N)$ *with cost* c.

4 Analyzing and Verifying P Systems With Costs

Using the previous operational correspondence provided by Theorem 1, the software experiments done in Priced-Timed Maude reflect exactly the evolution of the encoded membrane systems with costs. In this section we use a simple example of a membrane system with costs, example that is described in rewriting logic in the following form:

```
eq R0 = A -> B | emptyMO | emptyMAO | emptyMO | emptyMO | 10 .
eq R1 = A -> C | emptyMO | emptyMAO | emptyMO | emptyMO | 8 .
eq R2 = C -> D | emptyMO | emptyMAO | emptyMO | emptyMO | 12 .
eq Q = < 1 [ A + A + A | emptyMO | emptyMO | emptyMAO ] null > emptyMO .
eq rulesIn(1) = R0 R1 R2 .
eq S = Q ; mpr .
```

When entering the rewrite command

```
(ptfrew {S} in time <= 0 with cost <= 70 .)
```

Maude presents the following output:

```
Result PricedTimedSystem :
 {< 1[B + B + B | emptyMO | emptyMO | emptyMAO]null > emptyMO ; end}
    in time 0 with cost 30
```

We use priced-time Maude to check if certain configurations of a system can be reached (reachability problem).

```
(ptsearch {S} =>* {X:StateStop} with no limits .)
```

We use the *ptsearch* command to answer the question: starting from the initial membrane system S, what are the reachable final states (the ones containing the *end* tag)? This is done by searching for states which match a corresponding pattern. In this example, we use the $=> *$ symbol, meaning that we are searching for several steps. If one is interested in a bounded number of reachable final states, the command $ptsearch[n]$ can be used to obtain systems reachable in n steps. In our case, the output is

```
Priced-timed search in EXAMPLE
    {S} =>* {X:StateStop}
with no time or cost limit and with mode default time increase 10 :

Solution 1
TIME_ELAPSED:Time --> 0 ; TOTAL_COST_INCURRED:Cost --> 30 ;
  X:StateStop --> < 1[B + B + B | emptyMO | emptyMO | emptyMAO]null
  > emptyMO ; end

Solution 2
TIME_ELAPSED:Time --> 0 ; TOTAL_COST_INCURRED:Cost --> 40 ;
  X:StateStop --> < 1[B + B + D | emptyMO | emptyMO | emptyMAO]null
  > emptyMO ; end

Solution 3
TIME_ELAPSED:Time --> 0 ; TOTAL_COST_INCURRED:Cost --> 50
  X:StateStop --> < 1[B + D + D | emptyMO | emptyMO | emptyMAO]null
  > emptyMO ; end

Solution 4
TIME_ELAPSED:Time --> 0 ; TOTAL_COST_INCURRED:Cost --> 60 ;
  X:StateStop --> < 1[D + D + D | emptyMO | emptyMO | emptyMAO]null
  > emptyMO ; end

No more solutions
```

It should be noticed that after only two steps, the cost of the reachable configurations is very different depending on the rules applied.

In addition to these commands, Priced-Timed Maude allows to find optimal results such as the earliest state matching a pattern, as well as the cheapest evolution to reach a given configuration. In our case, the earliest reachable states containing the evolution type `end` can be found using the following command

```
(priced find earliest { Q ; mpr} =>* {X:MembraneSet ; end}
   with no cost limit .)
```

that returns the result:

```
Priced find earliest {X:MembraneSet ; end} in EXAMPLE such that
   {Q ; mpr} =>* {X:MembraneSet ; end}
with no cost limit with mode default time increase 10 :

Result: {< 1[B + B + B | emptyMO | emptyMO | emptyMAO]null >
   emptyMO ; end} in time 0 with cost 30
```

Using the command `find cheapest`, it is possible to detect the cheapest evolution (as cost) to reach a given configuration.

```
(find cheapest { Q ; diss} =>* {X:MembraneSet ; mpr}
   with no time limit . )
```

This command verifies that indeed reaching a configuration ready to apply maximal from a configuration ready to apply dissolution rules takes 10 time units with cost 0.

```
Find cheapest in EXAMPLE
   {Q ; diss} =>* {X:MembraneSet ; mpr}
with no time limit time and with mode default time increase 10 :

Solution
TIME_ELAPSED:Time --> 10 ; TOTAL_COST_INCURRED:Cost --> 0 ;
   X:MembraneSet
   --> < 1[A + A + A | emptyMO | emptyMO | emptyMAO]null > emptyMO
```

5 Conclusions and Future Work

We defined P systems with costs by assigning storage costs to membranes, as well as and execution costs to rules. We used the Priced-Timed Maude rewriting engine to implement these P systems with costs. By using such a rewriting engine corresponding to the semantics of membrane systems with costs, we proved the operational correctness of this implementation. Based on such an operational correspondence, we can analyze the P systems with costs and verified several interesting properties.

As a future work we plan to deal with Cost Problems in the framework of membrane systems by considering two variants of the cost problem, namely the Cost-Threshold Problem (can we obtain an evolution cost under a certain threshold value) and the Cost-Optimality Problem (compute the minimal evolution cost). We also intend to study how different evolution strategies influence the computed cost of reaching a desired configuration.

References

1. O. Agrigoroaiei, G. Ciobanu. Rewriting Logic Specification of Membrane Systems with Promoters and Inhibitors. *Electronic Notes in Theoretical Computer Science* **238**(3), 5–22 (2009).
2. B. Aman, G. Ciobanu. Time Delays in Membrane Systems and Petri Nets. *Electronic Proceeding in Theoretical Computer Science* **57**, 47–60 (2011).
3. O. Andrei, G. Ciobanu, D. Lucanu. A Rewriting Logic Framework for Operational Semantics of Membrane Systems. *Theoretical Computer Science* **373**, 163–181 (2007).
4. L. Bendiksen, P.C. Ölveczky. The Priced-Timed Maude Tool. *Lecture Notes in Computer Science* **5728**, 443–448 (2009).
5. P. Bottoni, C. Martín-Vide, Gh. Paun, G. Rozenberg. Membrane Systems With Promoters/Inhibitors. *Acta Informatica* **38**, 695–720 (2002).
6. G. Ciobanu, S. Marcus, Gh. Păun. New Strategies of Using the Rules of a P System in a Maximal Way: Power and Complexity. *Romanian Journal of Information Science and Technology* **12**(1), 157–173 (2009).
7. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J.F. Quesada. Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science* **285**, 187–243 (2002).
8. A. Leporati, C. Zandron, G. Mauri. Simulating the Fredkin Gate with Energy-based P Systems. *Journal of Universal Computer Science* **10**(5), 600–619 (2004).

9. R. Milner. Operational and Algebraic Semantics of Concurrent Processes. *Handbook of Theoretical Computer Science* **B**, 1201–1242, Elsevier (1990).
10. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences* **61**, 108–143 (2000).
11. Gh. Păun. *Membrane Computing. An Introduction.* Springer (2002).
12. G. Plotkin. Structural Operational Semantics. *Journal of Logic and Algebraic Programming* **60**, 17–140 (2004).

On the 14th BWMC

Maria Arazo

Universitat de Barcelona
Email: maria.arazo@gmail.com

Attending the *14th Brainstorming Week on Membrane Computing*[1] has been a challenge in many ways. A whole week in a place I had never been before, with a group of people I did not know, and learning about a topic which existence I completely ignored. With all that, given my initial hesitation, I am glad I decided to go, because it was worth it. I attended the 14th BWMC with six other Physics students from University of Barcelona who, like me, were interested in computational physics and were curious about the workshop, that took place in Seville from February 1st to 5th, 2016.

First of all, I could say that in some talks I got lost almost at the beginning; they were aimed for an audience with extensive knowledge on the topic of membrane computing and a solid background on mathematics and/or computer science. In other words, as a physicist and undergraduate student I felt a bit out of place. However, the tutorial sessions, that were meant to introduce membrane computing to those new to the field, were really useful and allowed me to follow the later presentations. It was also crucial and appreciated the patience of the workshop attendants; they answered any questions we asked them and even simplified some of the talks so students could follow them more easily.

Many specific topics and applications of membrane computing to other fields were presented during the talks. Some of them were more accessible for me than others, but in any way I found it interesting to listen to them, and to see how each attendee exposed about his/her research area. Despite the variety of topics that were treated, though, I would like to focus my memoir on a simple idea and a question: the basics of membrane computing, and how can it be applied to physics.

To summarise, membrane computing is a computational model inspired by nature, in which certain processes defined by rules take place in a system or cell that is hierarchically structured in compartments that are called membranes. This kind of systems, named P-systems after their creator Gheorghe Păun, are composed of multisets of objects, membranes delimiting the regions of the system, an environment, and rules that describe how a number of systems, also called machine (i.e.

[1] 14th BWMC website: http://www.gcn.us.es/14bwmc.

the cell), works according to its objects and membranes, and how they interact with each other. In the simplest case, it is considered that time is the same for all membranes, so a computational step comprises a series of transitions that occur regarding to the set of rules that is applied. Furthermore, in every computational step the maximal number of possible rules is applied in each membrane. The rules will be applied in each step until no more rules can be applied, in which case the computation halts.

An interesting characteristic element of those systems is that they present the possibility of adding a probabilistic factor to the rules, so a transition can follow different rules with the corresponding probabilities that have been defined. This introduces the concept of "fuzzy logic". Another main point of membrane computing is that it allows us to study a system with a very large number of initial objects as if all of them evolved independently and in a parallel way, like it happens on real biological systems, being the computational time proportional to the number of steps defined by the rules.

With this simple picture of the P-systems in mind, I think that it is impossible not to think in similar physical systems or in other problems that can be simplified in order to be modeled with membrane computing. In fact, motivated by the workshop attendants, which encouraged us to investigate how membrane computing could involve physics and vice versa, and moved also by our own curiosity, we thought about the improvements that membrane computing could bring into physics and in which cases could we apply it.

An important constraint we saw was that any system defined in the continuum needed to be discarded, because the set of objects that we consider is discrete. Nevertheless, membrane computing allows us to study certain magnitudes of a system with no need to define neither positions nor momenta.

The first case we decided to study was the Stern-Gerlach experiment of Quantum Mechanics. It is a simple example that can be modeled by membrane computing, where the magnitude under study is the third component of the spin of a very large number of incident particles that initially we define as positive and that after going through a Stern-Gerlach device can change or remain the same with probabilities that depend on the angle in which the Stern-Gerlach is oriented. By using a very high number of particles, the final count of positive and negative third components reproduces, respectively, the probabilities expected, and thus we show that by taking a measure, the result is altered.

The second example we considered is the uranium-238 decay chain, where we had to take several simplifications in order to apply what we had learnt from membrane computing. Initially, we start with n uranium nuclei, that naturally decay to form thorium-234 nuclei emitting α particles. While this decay takes place, since the resulting nuclei are also radioactive, they will decay in turn following the decay chain, until lead-206 is reached, which is a stable nucleus. The evident problem that this system entails is that, as we begin with n nuclei and not with a single nucleus, the number of disintegrations that compound the chain depend on the amount of parent nuclei left at every step of time, so we had to consider that

a new reaction could not begin until all the parent nuclei of the step before of the chain had decayed. With that unrealistic but useful approximation, the different disintegrations or reactions that form the decay chain are uncoupled to each other, and the resulting system can be easily modeled with membrane computing. A second simplification we had to take into consideration, which derives essentially from the first one, is that the time it takes for every reaction to take place must be constant and proportional to a certain number of time steps.

With those simplifications, all complexity and part of the interest of the system vanish and we are left with a rather simple problem, so we tried to study it, as in the first case, as a statistical problem: given that some nuclei can follow different decay modes that are weighed by some probabilities experimentally determined, membrane computing allows us to count at the end of the computation, once all reactions have taken place, the amount of resultant particles of each kind that have been emitted at every step. Since our nuclei decay mainly following α and β^- decay, we expected alpha particles or antineutrinos and electrons, and therefore we could see if the amounts of the different kinds of particles (that denoted the decay mode followed in each step) were the same for every initial U-238 nuclei or showed variations. Again, this is a very simple case and with a questionable utility, but it occurred to us that perhaps it could be described with another kind of system that allowed us to remove the simplifications and consider the real system and how it evolved in time. For the time being, we are working on it.

The attendees and organizers of the venue were, as I mentioned at the beginning, another remarkable element of the workshop. They not only helped us to enter into a world of which we knew little or nothing, but also made us feel like at home and encouraged us to participate more actively in the workshop. That at the end of the workshop we were presenting the few ideas we had been able to collect during the week, apart from putting us under pressure and keeping us busy even in the sparse free hours, was also a great motivation for asking and trying to understand more deeply.

Last but not least, I would like to emphasize how much I have learnt from my colleagues. Even though most of us had not met before, we managed to work as a group, first to help each other to understand what was explained in the talks, and later to make motivation alive and to work together in our little contribution to the workshop. We had very similar motivations, and that encouraged us to naturally build a team to achieve our common goal. And, at least in my case, I have participated much more in the workshop that what I would have participated had I gone alone.

To conclude, I think it has been a very rewarding experience, useful to learn about a new topic and to see how research about it was accomplished, to practice with team work, and to motivate me to improve in my studies and to head towards research.

Stern-Gerlach Experiment

Maria Arazo[1], Marc Barroso[1], Óscar De la Torre[1], Laura Moreno[1], Ariadna Ribes[1], Patricia Ribes[1], Ana Ventura[1], and David Orellana-Martín[2]

[1] Universitat de Barcelona
Email: {maria.arazo, marc.barroso4, oscar.delatorre.perez, 95morenolaura, aribesmetidieri, ribesmetidieri, a.venturabarroso}@gmail.com
[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
E-mail: dorellana@us.es

Summary. This work is about modelling an experiment composed by multiple Stern-Gerlach devices using *Membrane Computing*. We will study the behaviour of a set of independent particles passing through three linked Stern-Gerlach devices and discarting the spin down particles after passing through the first one, taking profit of the *Membrane Computing*'s ability of running parallel processing. Using a cell-like model to describe the system and testing it using the P-lingua framework we have obtained the theorically predicted results when the number of initial multisets is high enough.

1 Introduction

In 1998 Gheorghe Păun introduced an alternative computing science paradigm, *Membrane Computing* (*MC* from now on) [4]. P-systems appeared, and with it, an innovative way to interpret the natural world. Those models are based on the structure of living cells and how they process compounds within their membranes (cell-like system) or even how they interact one with the others (tissue-like system). To model the processes occurring inside them, they make use of rules, which represent the different reactions or exchanges of the objects inside or through the membranes. Those rules can be of several types: communication, rewriting, annihilation, etc. For years, they have been applied to study the evolution of biological systems, neuronal systems or even complex ecosystems. Nevertheless, the application of *Membrane Computing* paradigms in other fields has not been yet so promoted. Therefore, one of the aims of this project is to extend the *Membrane Computing* applicability to physics. Moreover, this article has been used as a excuse in order to learn about this new computing paradigm and to be able to seek for possible further applications.

1.1 Combining *MC* with Physics

When studying Physics the aim is not to know exactly what things are, but to understand how they behave, this is the reason why it is so important to model natural phenomena. The modelling process' goal is to attain a set of analytical expressions that describes the studied system reduced into a determined approximation.

Because of the complexity of those analytical expressions, it is commonly useful to solve them with numerical methods. When dealing with a great number of particles, even with $n \to \infty$, being n the number of particles, the computation time could become large enough that it would turn inefficient (to use certain numerical methods). Here is where *Membrane Computing* provides a really suitable framework, due to maximal parallelism, one of *MC*'s main features.

Membrane Computing is originally based in a model analogous to cells and tissues. Because of that, it could fit perfectly in a system of discrete particles. By making the analogy with the structure provided by *MC*, one could identify particles with the objects, whereas rules applied in each membrane enable us to model particles' behaviour. It is also a useful tool when working with problems involving non-deterministic processes, i.e. those that can be found in modern physics, such as quantum mechanics or nuclear physics, where probabilities play an important role.

1.2 Introduction to Quantum Mechanics

For many years, the world was ruled by Classical Mechanics, which considered that all processes occurred in a deterministic way, i.e. one could predict the position and momentum of any particle at the same time. It was not until 1900, when Max Planck published his paper [6], that new phenomena which did not coincide with classical physics stopped seeming unwarranted, then quantum physics was borned.

Quantum Mechanics is a fundamental branch of Physics that explains the behaviour of subatomic particles and it is grounded on the idea that measurable observables are discrete and quantified. In Quantum Mechanics, the mathematical formalism is based on the Hilbert space, hence the Quantum world is described by six postulates, whereas the evolution of the body's movement in classical mechanics is ruled by Newton's Law.

Postulate 1. On the representation of the state of a physical system.
The maximum possible information on a physical system at a given time t is its quantum state ψ, which is represented as a vector $|\psi\rangle$ of unitary module and arbitrary phase in a separable Hilbert space.

Postulate 2. On the representation of measurable magnitudes.
Every measurable magnitude of the system has associated a linear and autoadjoint operator defined on the vector space of the states. The totality of the eigenvalues is the spectrum, and the eigenvectors define a base on the Hilbert space.

Postulate 3. On the result of the measure.
The result of measuring the observable A is one of its eigenvalues a_i of the spectrum, the probability of obtaining the result a_i is given by:

$$P_{\psi}^{(a_i)} = |\langle a_i|\ \psi\rangle|^2$$

Postulate 4. On the collapse or reduction of the wave function.
Immediately after measuring the observable A with result a_i, the new state of the system is $|a_i\rangle$, i.e. the corresponding eigenvector.

Postulate 5. On the temporal evolution.
Between measures, the system evolves according to the Schrödinger equation:

$$i\hbar \frac{d}{dt} |\psi(t)\rangle = \mathcal{H}(t) |\psi(t)\rangle$$

Where $\mathcal{H}$ is the Hamiltonian of the system.

Postulate 6. On the Pauli exclusion principle.
The position and momentum operators for fermions satisfy commutative rules that are directly related to the Pauli exclusion principle, i.e. that two fermions cannot have the same quantum numbers.

2 Stern-Gerlach experiment

2.1 The basic Stern-Gerlach experiment

The Stern-Gerlach experiment [3] is used to illustrate that particles have intrinsic properties such as the spin, the orbital momentum, etc. The total momentum of a particle is the composition of the orbital momentum and the spin, being the last one the observable that is measured by the Stern-Gerlach device. In particular, we focus the study on the third component of the spin that it is discrete and quantifiable, and can only take the values (i.e. eigenvalues) $+\frac{\hbar}{2}$ (denoted as *up* to simplify notation) or $-\frac{\hbar}{2}$ (*down*).

The basic Stern-Gerlach experiment is composed by a magnet that creates a non-uniform magnetic field oriented towards a general direction $\hat{n}$ that is contained in the plain surface perpendicular to the particle's direction of propagation (the y axis in Figure 1). Once the particle has passed through the magnetic field, the third component of the spin may have changed.

In the particular case of a Stern-Gerlach (device) oriented towards the z axis (i.e. the magnetic field too), the third component of the spin is measured after the particles passes through. While in a more general case (where $\hat{n}$ is a general direction as mentioned above), the measured magnitude is the projection of the spin in that arbitrary direction.

2.2 The experiment modelled with *MC*

The modelled experiment consists on three Stern-Gerlach devices situated along the x axis, and a set of particles that go through the three of them and impact

on a screen. Initially, the incident particles have an undetermined state, i.e. the third component of the spin may be positive or negative and unknown unless it is explicitly measured. The first Stern-Gerlach, which is $\hat{z}$-oriented, defines the third component of the spin with a fifty percent of probability of being either positive (or *up*, to simplify notation), or negative (*down*). Then, once a particle with non-determined spin goes through the first Stern-Gerlach device, which in Figure 2 is labelled as $SG1$, the spin-state of the particle becomes determined. This is what we define as the *initial state* for the other two Stern-Gerlach devices. Since we are considering an arbitrary number n of incident particles, after this step we would obtain 50% particles with third component of the spin *up* and 50% *down* if n is large enough.

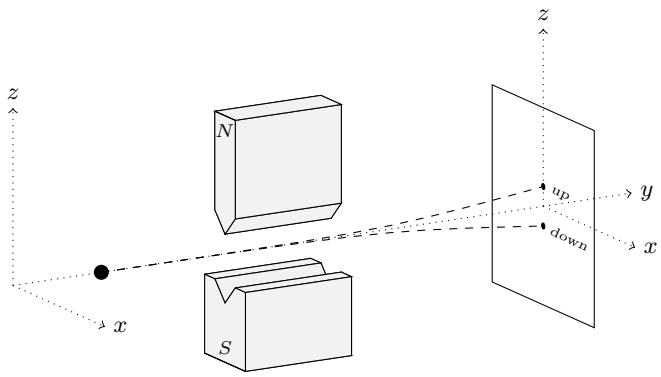


Fig. 1: Original Stern-Gerlach experiment, with the magnet parallel to the $\hat{z}$ axis so the z component of the spin is measured.

For the second Stern-Gerlach device, we discard the particles with initial spin *down* and make the *up*-particles go through the magnet (labelled as $SG2$), oriented with an arbitrary angle θ as shown in Figure 2. In this general case, the probabilities of obtaining spin *up* or spin *down* do not only depend on the initial state of the particle (now restrained to *up*), but also on the angle θ between the Stern-Gerlach device and the z axis. Those probabilities are derived on the following lines.

For $\hat{n} = \sin\theta\hat{\imath} + \cos\theta\hat{k}$ the direction of the Stern-Gerlach device and $\boldsymbol{\sigma} = \sigma_x\hat{\imath} + \sigma_y\hat{\jmath} + \sigma_z\hat{k}$ a general vector for the Pauli matrices[3], the associated matrix is

[3] The Pauli matrices are the most general hermitic matrices of dimension 2×2 with eigenvalues 1 and -1, and are defined as:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

With eigenvalues 1 and -1 and the corresponding eigenvectors:

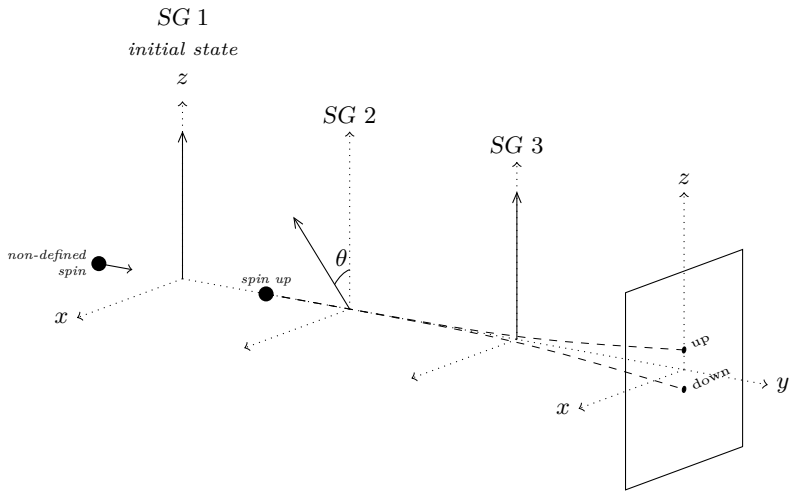


Fig. 2: Three Stern-Gerlach devices as proposed in this article

$$\hat{n}\boldsymbol{\sigma} = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}$$

Given that the eigenvalues are $+1$ and -1 as in the Pauli matrices, and imposing that the corresponding eigenvectors generalised as $(\alpha \;\; \beta)$ should be normal and therefore satisfy that $|\alpha|^2 + |\beta|^2 = 1$, the eigenvectors found are:

$$|\hat{n}\boldsymbol{\sigma} = +1\rangle = \begin{pmatrix} \cos\theta/2 \\ \sin\theta/2 \end{pmatrix}$$

$$|\hat{n}\boldsymbol{\sigma} = -1\rangle = \begin{pmatrix} -\sin\theta/2 \\ \cos\theta/2 \end{pmatrix}$$

And the probabilities[4] of obtaining spin *up* or *down* for a particle with initial *up* state[5] are defined by:

$$|\sigma_x = +1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad |\sigma_x = -1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$|\sigma_y = +1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ i \end{pmatrix} \qquad |\sigma_y = -1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -i \end{pmatrix}$$

$$|\sigma_z = +1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |\sigma_z = -1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

[4] More on notation: the first subscript refers to the initial state, and the second one to the final state, so $\mathcal{P}_{\uparrow\downarrow}$ is the probability of obtaining *down* spin given a particle with initial spin *up*.

[5] Notation for the initial states *up* or *down*:

$$\mathcal{P}_{\uparrow\uparrow} = |\langle \hat{n}\boldsymbol{\sigma} = +1| + \rangle|^2 = \left| \left(\cos\theta/2 \ \sin\theta/2\right) \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right|^2 = \cos^2\frac{\theta}{2}$$

$$\mathcal{P}_{\uparrow\downarrow} = |\langle \hat{n}\boldsymbol{\sigma} = -1| + \rangle|^2 = \left| \left(-\sin\theta/2 \ \cos\theta/2\right) \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right|^2 = \sin^2\frac{\theta}{2}$$

Following that same procedure, the probabilities for initial *down* state particles are:

$$\mathcal{P}_{\downarrow\uparrow} = |\langle \hat{n}\boldsymbol{\sigma} = +1| - \rangle|^2 = \left| \left(\cos\theta/2 \ \sin\theta/2\right) \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right|^2 = \sin^2\frac{\theta}{2}$$

$$\mathcal{P}_{\downarrow\downarrow} = |\langle \hat{n}\boldsymbol{\sigma} = -1| - \rangle|^2 = \left| \left(-\sin\theta/2 \ \cos\theta/2\right) \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right|^2 = \cos^2\frac{\theta}{2}$$

Finally, the particles will pass through the last SG device ($SG3$), which is oriented on the z axis and therefore allows us to measure the third component of the spin[6] and count how many particles have as final state spin *up* or spin *down*. This result, though redundant, shows one of the most important facts of Quantum Mechanics: that measure alters the system. As we can observe here, though we considered only the *up*-particles to go through the second Stern-Gerlach, on the final state (i.e. the screen) we have recovered the initial distribution of fifty percent of particles *up* and fifty percent *down*. See Figure 3 for an scheme of the proposed experiment.

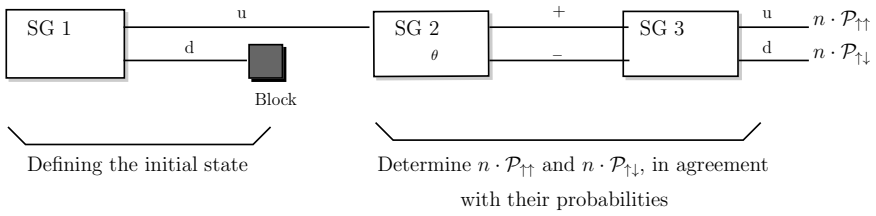


Fig. 3: Schematic description of the proposed experiment

3 P-system model

3.1 PDP systems

P systems [2] [5] are an abstraction of the membrane structure inside a cell, which delimite regions containing objects that can evolve acording to certain rules. In

up: $|\uparrow\rangle = |+\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ *down*: $|\downarrow\rangle = |-\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

[6] As a reminder, the Stern-Gerlach device basically measures the projection of the spin in the direction that it is oriented, i.e. a $\hat{z}$-SG measures the third component S_z and so on.

general, the dynamic of those systems is defined through a non-deterministic and synchronised mode. Population Dyanmics P systems (PDP) models are a complex variant of P systems, as they consider a collection of environments, each containing a cell (all of them with the same membrane structure and rules) connected among them as a network. Also, in those models, rules are associated with probabilistic functions and membranes with polarizations. All these ingredients make PDP systems a useful computational tool to model complex systems.

On this articles, we focus on a reduced version of PDP systems as no envrionment and polarizations are needed. What follows are the main aspects.

We define a probabilistic P system as a tuple

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, R, \{f_r | r \in R\})$$

where:

- Γ is a finite set, not empty, called *alphabet*, whose elements are named objects of Π. The whole of all finite multisets over Γ is denoted by $MF(\Gamma)$.
- μ is a tree structure, labelled by $\{i | 1 \leq i \leq q\}$, that describes the membranes' structure. The skin membrane (also named as *tree root*) is the only one labelled by 1.
- $\mathcal{M}_i \in MF(\Gamma)$, $1 \leq i \leq q$, is the initial multiset of objects associated to cell i.
- R is a finite set of *evolution rules* of the form: $u\,[v]_i \to u'\,[v']_i$, where $u, v, u', v' \in MF(\Gamma)$, $1 \leq i \leq q$, and $|u| + |v| \neq 0$. With $u\,[v]_i$ being the left-hand side of the rule.
- For each $r \in R$, $f_r \in [0, 1]$, describes the probability distribution over the rules with a same left-hand side. Then $\sum_r f_r = 1$ for all the the rules in R whose left-hand side is equal.

A rule $r \in R$ of the form $u\,[v]_i \to u'\,[v']_i$ can be applied within a membrane labelled i if it contains v and its parent membrane contains u. If a rule of this kind is applied, objects in v and u vanish from membrane i and its parent. Simultaneously, objects in v' and u' are included in membrane i and its parent, respectively.

A configuration for any unit time is a tuple that specifies the multisets of objects that can be found in each membrane. In every step of time, rules applied are chosen in a non-deterministic way depending on its left-hand side, taking into account the probability associated to each of them. A maximal number of rules are applied simultaneously. Computation is a succession of configurations such that the first one coincides with the initial configuration and every of the remaining are obtained from the former using the rules of the system as it has been described above.

3.2 Model

In order to test the designed system we have define it for the simulator given by the P-lingua framework, using a single cell with two inner membranes. Given n as the number of particles we want to do the experiment with, we put an object a (in

membrane 2) n times, i.e., a^n would be the input multiset.

Let $\Pi_{\mathrm{SG}} = (\Gamma, \mu, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, R, \{f_r | r \in R\})$, where:

- $\Gamma = \{a, u, d\}$ is the alphabet we use for this multiset.
- $\mu = [\,[\quad]_2\,[\quad]_3\,]_1$ is the structure of the membranes.
- $\mathcal{M}_i = \emptyset, i \in \{1, 3\}$.
- $\mathcal{M}_2 = a^n$.
- The set of rules R, each rule with its corresponding probability, is:
  (a) These rules take care of the initial state spin of the particles that is determined by the first Stern Gerlach device. The particles with spin *down* are blocked.
  $r_1 \equiv [a]_2 \xrightarrow{1/2} [d]_2$
  $r_2 \equiv [a]_2 \xrightarrow{1/2} u[\ ]_2$
  (b) Here, we simulate the particles passing through the second and third Stern Gerlach devices. The spin of the particles is determined according to a probability given by the angle of the magnetic field of the second Stern Gerlach with the $\hat{z}$ axis. The third Stern Gerlach is simulated by introducing the particles with definite spin within the membrane with label 3, where the results are collected.
  $r_3 \equiv u[\ ]_3 \xrightarrow{\cos^2(\theta/2)} [u]_3$,
  $r_4 \equiv u[\ ]_3 \xrightarrow{1-\cos^2(\theta/2)} [d]_3$,

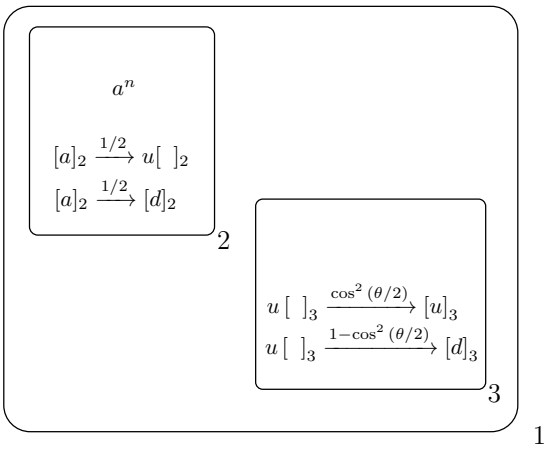
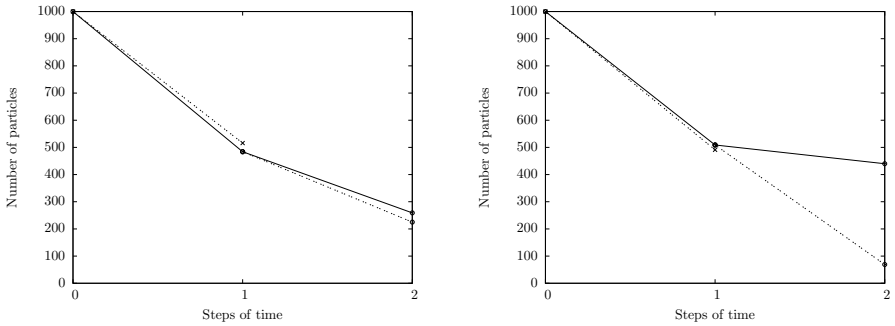


Fig. 4: Visual representation of our P-system

4 Code

```
@model<probabilistic>
def Sg(@cos(theta / 2))
{
  @mu = [[]'2 []'3]'1;
 /* Rules in membrane 2 */
 /* Here, we determine the initial state of the particle
 by blocking the particles with down spin, which remain
 in the region 2*/
 [a]'2 --> [d]'2 :: 0.5;
 [a]'2 --> u[]'2 :: 0.5;
 /* Rules in membrane 3 */
 /* Implementation of the second Stern Gerlach 2 and 3*/
 u[]'3 --> [u]'3 :: @cos(theta / 2) * @cos(theta / 2);
 u[]'3 --> [d]'3 :: 1-@cos(theta / 2) * @cos(theta / 2);
}
def main()
{
  call Sg(@cos(theta / 2));
  @ms(2) = a*1000;
} /* End of main module */
```

5 Results



(a) $\theta = \pi/2$. (b) $\theta = \pi/4$.

Fig. 5: Results obtained with $n = 10^3$ particles, for two different angles, $\theta = \pi/2$ and $\theta = \pi/4$. In both figures, it has been represented how from the inicial n particles we obtain a first selection of *up* and *down* particles, and from the *up* ones how we reach the final results. In dashed lines are represented the path to *down* particles, whereas in solid lines to the *up* ones. The cross at the end of the dashed line represents the block stopping the *down* particles in the end of the 1st step of time, as it is shown in Figure 3.

The results obtained:

		Number of initial objects, n							
		10^1		10^2		10^3		10^4	
		u	**d**	**u**	**d**	**u**	**d**	**u**	**d**
θ	0	4	0	49	0	489	0	4974	0
	$\pi/4$	5	0	42	1	440	69	4239	759
	$\pi/2$	4	2	32	25	259	225	2456	2586
	$3\pi/4$	0	3	11	39	70	439	782	4309

Table 1: Simulated results for 4 given initial numbers of particles and for different values of the angle θ.

For $\boldsymbol{\theta = 0}$: As it can be seen from Table 1, the simulated results validate the expected behavior of the particles. The first Stern-Gerlach device SG1 determinates the initial

state (spin *up* or down) and blocks the particles with spin *down*. As the two consecutive Stern Gerlach are aligned, the probability of obtaining particles with spin *down* in SG2 and SG3 is null. As $n \to \infty$, the probability of obtaining *up* spin particles tends to $\frac{1}{2}$.

For $\boldsymbol{\theta = \frac{\pi}{4}}$: The first Stern Gerlach continues acting as a selector of the particles with spin *up*, however the SG2 and SG3 are not aligned, allowing that approximately 7% of the particles have spin *down*, as it can be seen in Figure 5b.

For $\boldsymbol{\theta = \frac{\pi}{2}}$: due to the block imposed by the SG1, approximately only half the particles reach SG2 and SG3, and the relative orientation of both devices causes the final proportion of both spin *up* and spin *down* particles to be approximately a quarter of the total, as it can be seen in Figure 5a.

For $\boldsymbol{\theta = \frac{3\pi}{4}}$: As expected, the numbers of resulting particles with spin *up* and with spin *down* is quite similar to the numbers obtained for $\theta = \frac{\pi}{4}$ but exchanging the results for *up* and *down* particles.

Summarizating, as the number of particles in the experiment increases, the experimental probability (calculated $P_{\uparrow\uparrow}^{exp} = {}^{u}\!/_{N}$ and $P_{\uparrow\downarrow}^{exp} = {}^{d}\!/_{N}$) tends to the expected probabilities ($P_{\uparrow\uparrow} = {}^{1}\!/_{2} \cos^2 ({}^{\theta}\!/_{2})$ and $P_{\downarrow\downarrow} = {}^{1}\!/_{2} \sin^2 ({}^{\theta}\!/_{2})$), as shown in Table 2.

		n							
		10^1		10^2		10^3		10^4	
		$\left\lvert \mathcal{P}_{\uparrow\uparrow} - \frac{u}{N} \right\rvert$	$\left\lvert \mathcal{P}_{\uparrow\downarrow} - \frac{d}{N} \right\rvert$	$\left\lvert \mathcal{P}_{\uparrow\uparrow} - \frac{u}{N} \right\rvert$	$\left\lvert \mathcal{P}_{\uparrow\downarrow} - \frac{d}{N} \right\rvert$	$\left\lvert \mathcal{P}_{\uparrow\uparrow} - \frac{u}{N} \right\rvert$	$\left\lvert \mathcal{P}_{\uparrow\downarrow} - \frac{d}{N} \right\rvert$	$\left\lvert \mathcal{P}_{\uparrow\uparrow} - \frac{u}{N} \right\rvert$	$\left\lvert \mathcal{P}_{\uparrow\downarrow} - \frac{d}{N} \right\rvert$
θ	0	0.1000	0.0000	0.0100	0.0000	0.0110	0.0000	0.0026	0.0000
	$\frac{\pi}{4}$	0.0732	0.0732	0.0068	0.0632	0.0132	0.0042	0.0029	0.0027
	$\frac{\pi}{2}$	0.1500	0.0500	0.0700	0.0000	0.0090	0.0250	0.0044	0.0086
	$\frac{3\pi}{4}$	0.0732	0.1268	0.0368	0.0368	0.0032	0.0122	0.0050	0.0041

Table 2: Simulated results for 4 given initial numbers of particles and for different values of the angle θ.

6 Conclusions

The results yielded by the designed system (obtained by the P-lingua simulator) are consistent with the theory, as have been explained above. Therefore, we have achieved our main objective: showing that P systems can be applied to physics, and more specifically, they can be used to implement a simplified/theoretical version of the S-G experiment, and only a little part of the power of such computational systems was used. No other exceptional consequence was predicted, as we understand that this works basically as a pedagogical application. Further research could consist in trying to apply these systems to non-trivial physical phenomena, where an analytic result might not be possible to obtain. Taking into account the non deterministic approach inherent to the model, as explained

before, and the ability to make all the computations and apply all the rules in a parallel sequence (following the maxpar criterion), it seems a very suitable framework to implement other experiments from the modern physics world. For example, light polarization works in a similar way to the Stern-Gerlach experiment; with some modifications to the model we could simulate how light behaves when passing through a polarizer. Other applications, however, can be arbitrary hard, as the very nature of the objects used in the membranes make it very difficult to exemplify a portion of matter, for instance. Also, it is important to remark that a lot more theory about computation is developed around *MC* that the one shown here, taking the subject as far as showing that these cell-like scheme is a universal Turing machine [1], and thus able to make any computation our normal computers can. There is no theoretical limit on what can be implemented.

Acknowledgments.

We would like to thank: Agustín Riscos-Núñez, Carmen Graciani and Mario J. Pérez-Jiménez from Universidad de Sevilla and Ricardo Graciani from Universitat de Barcelona for their valuable comments and suggestions, as well as for their reviews of the several versions of this paper; the 14th BWMC attendants for their patience and advice on our first ideas; and also Francesc Salvat, Assumpta Parreño, Ricardo Graciani and Bruno Juliá-Díaz from Universitat de Barcelona for offering us the opportunity to attend the Brainstorming.

References

1. G. Ciobanu, G. Paun, and M. J. Pérez-Jiménez. *Applications of membrane computing*, volume 17. Springer, 2006.
2. M. Colomer, A. Margalida, and M. J. Pérez-Jiménez. Population Dynamics P System (PDP) Models: A Standardized Protocol for Describing and Applying Novel Bio-Inspired Computing Tools. *PLOS ONE*, 8(4):1–13, 2013.
3. W. Gerlach and O. Stern. Der experimentelle Nachweis der Richtungsquantelung im Magnetfeld. *Zeitschrift fuer Physik*, 9:349–352, 1922.
4. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
5. M. J. Pérez-Jiménez. Simulación y Análisis Computacional en Biología de Sistemas. Tema IV: Modelización computacional basada en sistemas P. Modelos estocásticos y probabilísticos.
6. M. Planck. Ueber das gesetz der energieverteilung im normalspectrum. *Annalen der Physik*, 309(3):553–563, 1901.

Uranium-238 decay chain

Maria Arazo[1], Marc Barroso[1], Óscar De la Torre[1], Laura Moreno[1], Ariadna Ribes[1], Patricia Ribes[1], Ana Ventura[1], and David Orellana-Martín[2]

[1] Universitat de Barcelona
Email: {maria.arazo, marc.barroso4, oscar.delatorre.perez, 95morenolaura, aribesmetidieri, ribesmetidieri, a.venturabarroso}@gmail.com
[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
E-mail: dorellana@us.es

Summary. The main objective of this article is to modelize the process of decay of Uranium 238 within the framework of Membrane Computing, so the evolution of great numbers of particles can be progressively followed and the results of the desintegrations (nuclei coming from α and β^- decays) can be counted.

In order to model the process in an accurate manner, exploiting the properties of maximal parallelism and non-determinism of *Membrane Computing*, a Population Dynamic P system (or PDP for short) restricted to one environment and a P system conformed by only the skin have been selected.

The difficulty in the characterisation of this reactions lays in the simultaneity of the different decays, since the number of desintegrations of nucleous of each specie depend on the number of atoms of the initial population. In order to solve this problem and keep their attachment, the characteristic time of production of each decay has been translated into probabilities of deintregration of a nucleous using the decay constant λ.

1 Introduction

In this paper we are considering the Uranium-238 decay, which will be explained in the following sections. One of the first objectives was to prove that making use of *Membrane Computing* and the P-lingua simulation, we could obtain the results previously known, e.g. the ways that intermediate products of the decay took to arrive to the final product or the amount of different elements that were produced during the chain. Nevertheless, during the development of this project another interesting problem, which will be explained and discussed in later on, appeared: time implementation. At this point, our main goal was to look for different ways of modeling the physical process as close as possible to reality. Even so, it is still interesting to know which products we obtain in each disintegration, so we are able to proof the most probable ways of decay.

The problem with half-lifetimes.

During the modeling of the decay processes we found out some problems when implementing the time involving the reactions, i.e. the disintegration of one element into another one has an intrinsic half-lifetime associated to it. This parameter $T_{1/2}$, found in equations (1) and (2), determines the time that takes for the element to reduce the number of its nuclei to half of the initial ones. Also, an important constant is τ, the decay constant, defined as: $\tau = \frac{1}{\lambda}$, which represents the probability of a nucleus to decay, per unit of time.

$$\frac{\mathrm{d}N}{\mathrm{d}t} = -\lambda N \tag{1}$$

$$T_{1/2} = \frac{\ln 2}{\lambda} \tag{2}$$

Being N the number of nuclei at a given time t, and λ, the number of disintegrations per second, which is a constant for a given reaction.

From (1) it can be noticed that the rate of disintegration depends not only on the constant of disintegration, λ, but also on the population of nuclei at the time we are calculating the disintegration rate. This is the reason why rather than considering the rate as the parameter to characterize the reactions, sometimes is better to consider what we define as half-lifetime, $T_{1/2}$, which is constant because it only depends on λ.

These processes occur all at the same time, so to say, from the first moment when we obtain the second nucleus of the chain, another reaction begins to take place: it does not wait for all the first elements to react. Taking into account that P-Systems are based in systems that evolve by steps of time we were aware we had to find a way to approximate as close as we could to the fact that time is continuous. To do so while trying not to differ a lot from what happens in reality we went through different models making some changes in the implementation of the time. The two methods that we selected, which will be further explained in following sections, were the following:

- Steps of time: The first approach to the problem of Uranium decay consisted in translating the half-lifetimes of the different decays of the chain by a logarithmic scale so the considered range of variation was reduced enough in other to assign a proportional and arbitrary amount of time for each step. There it has been considered that one reaction must be applied to all nuclei before beginning the following reaction of the chain. In this case, index notation was used to represent the duration of each step. Although not being a model really close to the real situation, one could obtain the expected results. So, for example, the first reaction was assigned a counter that went from 1 to 7. This counter ensured that no reaction could begin before having ended previously the earlier step in the chain.

  In a way, this process roughly simulated the different periods of time required for each element of the chain to vanish. However it doesn't allow that different

elements react at the same time. The next step in the chain has to wait until the previous one finished. Therefore, although the simulation that implements this rules approaches reasonably well the amount of particles gathered at the end of the process, it was not a good approach to reality, as in a real decay several reactions of the chain take place at the same time.

- Probabilistic model: in which it has been taken into account that once a nucleus has decayed into the next one, the following reaction can take place for that recently generated nucleus. This model is a useful way to determine which particles were generated at each moment, i.e. one could thoroughly examine the intermediate stages of the decay.

2 Uranium-238 decay chain

2.1 Radioactive series

Nuclear decays [2] are transitions to less energetic —and thus more stable— states. An initial unstable nucleus can naturally decay into another nucleus, usually but not necessarily lighter, following different modes characterized by the emitted particles and the resultant nuclei. The ones concerning our study are the α decay (3) and the β^- decay (4), where the emitted particles can be He nuclei (α particles) or electrons (along with their corresponding antineutrino).

$$ {}^{A}_{Z}X \rightarrow {}^{A-4}_{Z-2}Y + {}^{4}_{2}\mathrm{He} \tag{3} $$

$$ {}^{A}_{Z}X \rightarrow {}^{A}_{Z+1}Y + e^- + \bar{\nu}_e \tag{4} $$

Where A is the mass number and Z the atomic number. Other possible decay modes are the β^+ decay (with emission of positrons and electronic neutrinos), the gamma emission, and the electronic capture. Of the three kinds of possible emitted particles (α, β and γ), γ particles have the largest penetrating power, while α particles interact more with matter.

The resulting nuclei of a nuclear decay can still be unstable and therefore decay into another nuclei and the corresponding particle. In this way, several decays may take place until a stable nucleus is reached. This process of chained decays that begins on a unstable parent nucleus and end on a number of stable nuclei is called a *radioactive series* or *decay chain*.

The parent nuclei of the radioactive series usually have very large lifetimes (i.e. the time it takes to the initial population to disappear entirely). There are four main radioactive series (three of them being natural), and all of them end in lead, which is stable.

2.2 U-238 decay chain

As we can see in Table 1, the Uranium decay chain consists of 15 main steps (i.e. decay reactions). This table shows the most probable decay modes, but there are other decays with an extremely low probability of occurring, showed with more detail in the second part of Table 1. Nevertheless, independently from the path chosen, the final product is always lead (Pb-206), which is stable. A diagram of the whole U-238 decay chain and its less probable decay modes can be found in Figure 1.

Parent	$\mathbf{T_{1/2}}$ **(s)**	$\boldsymbol{\lambda}$ **(decays/s)**	$\boldsymbol{\tau}$ **(s)**	**Decay modes**	**Reaction**	$f_{r,1}$
$^{238}_{92}$U (a)	1.41×10^{17}	4.92×10^{-18}	2.03×10^{17}	α (100%): $^{234}_{90}$Th	$[a]_0 \rightarrow x[b]_0$	9.68×10^{-1}
$^{234}_{90}$Th (b)	2.08×10^{6}	3.33×10^{-7}	3.00×10^{6}	β^- (100%): $^{234}_{91}$Pa	$[b]_0 \rightarrow z[c]_0$	5.33×10^{-1}
$^{234}_{91}$Pa (c)	2.41×10^{4}	2.87×10^{-5}	3.48×10^{4}	β^- (100%): $^{234}_{92}$U	$[c]_0 \rightarrow z[d]_0$	6.22×10^{-1}
$^{234}_{92}$U (d)	7.74×10^{12}	8.95×10^{-14}	1.12×10^{13}	α (100%): $^{230}_{90}$Th	$[d]_0 \rightarrow x[e]_0$	2.29×10^{-1}
$^{230}_{90}$Th (e)	2.38×10^{12}	2.92×10^{-13}	3.43×10^{12}	α (100%): $^{226}_{88}$Ra	$[e]_0 \rightarrow x[f]_0$	2.53×10^{-1}
$^{226}_{88}$Ra (f)	5.05×10^{10}	1.37×10^{-11}	7.28×10^{10}	α (100%): $^{222}_{86}$Rn	$[f]_0 \rightarrow x[g]_0$	3.30×10^{-1}
$^{222}_{86}$Rn (g)	3.30×10^{5}	2.10×10^{-6}	4.77×10^{5}	α (100%): $^{218}_{84}$Po	$[g]_0 \rightarrow x[h]_0$	5.70×10^{-1}
$^{218}_{84}$Po (h)	1.86×10^{2}	3.73×10^{-3}	2.68×10^{2}	α (99.98%): $^{214}_{82}$Pb	$[h]_0 \rightarrow x[v]_0$	7.20×10^{-1}
				β^- (0.02%): $^{218}_{85}$At	$[h]_0 \rightarrow z[j]_0$	1.44×10^{-4}
$^{214}_{82}$Pb (v)	1.62×10^{3}	4.27×10^{-4}	2.34×10^{3}	β^- (100%): $^{214}_{83}$Bi	$[v]_0 \rightarrow z[k]_0$	6.77×10^{-1}
$^{214}_{83}$Bi (k)	1.19×10^{3}	5.81×10^{-4}	1.72×10^{3}	β^- (99.979%): $^{214}_{84}$Po	$[k]_0 \rightarrow x\,[n]_0$	1.43×10^{-4}
				α (0.021%): $^{210}_{81}$Tl	$[k]_0 \rightarrow z\,[m]_0$	6.83×10^{-1}
$^{214}_{84}$Po (m)	1.64×10^{-4}	4.22×10^{3}	2.37×10^{-4}	α (100%): $^{210}_{82}$Pb	$[m]_0 \rightarrow x[p]_0$	1.00×10^{0}
$^{210}_{82}$Pb (p)	7.00×10^{8}	9.90×10^{-10}	1.01×10^{9}	β^- (100%): $^{210}_{83}$Bi	$[p]_0 \rightarrow x\,[o]_0$	7.90×10^{-9}
				α ($1.9{\times}10^{-6}$%): $^{206}_{80}$Hg	$[p]_0 \rightarrow z\,[q]_0$	4.16×10^{-1}
$^{210}_{83}$Bi (q)	4.33×10^{5}	1.60×10^{-6}	6.25×10^{5}	β^- (100%): $^{210}_{84}$Po	$[q]_0 \rightarrow x\,[s]_0$	7.45×10^{-7}
				α (13.2×10^{-5}%): $^{206}_{81}$Tl	$[q]_0 \rightarrow z\,[r]_0$	5.64×10^{-1}
$^{210}_{84}$Po (r)	1.20×10^{7}	5.80×10^{-8}	1.72×10^{7}	α (100%): $^{206}_{82}$Pb	$[r]_0 \rightarrow x[t]_0$	4.98×10^{-1}
$^{206}_{82}$Pb (t)	Stable	—	—	—	—	—
Other (less probable) decays						
$^{218}_{85}$At (j)	1.50	4.62×10^{-1}	2.16	α (99.9%): $^{214}_{83}$Bi	$[j]_0 \rightarrow x\,[k]_0$	8.16×10^{-1}
				β^- (0.1%): $^{218}_{86}$Rn	$[j]_0 \rightarrow z\,[l]_0$	8.17×10^{-4}
$^{218}_{86}$Rn (l)	3.50×10^{-2}	1.98×10^{1}	5.05×10^{-2}	α (100%): $^{214}_{84}$Po	$[l]_0 \rightarrow x[m]_0$	8.92×10^{-1}
$^{210}_{81}$Tl (n)	7.80×10^{1}	8.89×10^{-3}	1.13×10^{2}	β^- (100%): $^{210}_{82}$Pb	$[n]_0 \rightarrow z[p]_0$	7.38×10^{-1}
$^{206}_{80}$Hg (o)	4.99×10^{2}	1.39×10^{-3}	7.20×10^{2}	β^- (100%): $^{206}_{81}$Tl	$[o]_0 \rightarrow z[s]_0$	7.00×10^{-1}
$^{206}_{81}$Tl (s)	2.52×10^{2}	2.75×10^{-3}	3.64×10^{2}	β^- (100%): $^{206}_{82}$Pb	$[r]_0 \rightarrow x[t]_0$	4.98×10^{-1}

Table 1: Half-life times, decay constants, mean lifetimes, decay modes, reaction and probability functions for each reaction in the U-238 decay chain. The letter in brackets corresponds to the letter assigned to each nucleus for implementation. Half-lifes and probabilities for the chain decays obtained from [3] [5]; probabilities for the less probable decays obtained from [4]. The probability function associated to each transformation rule depends on the decay constant and the decay mode probability.

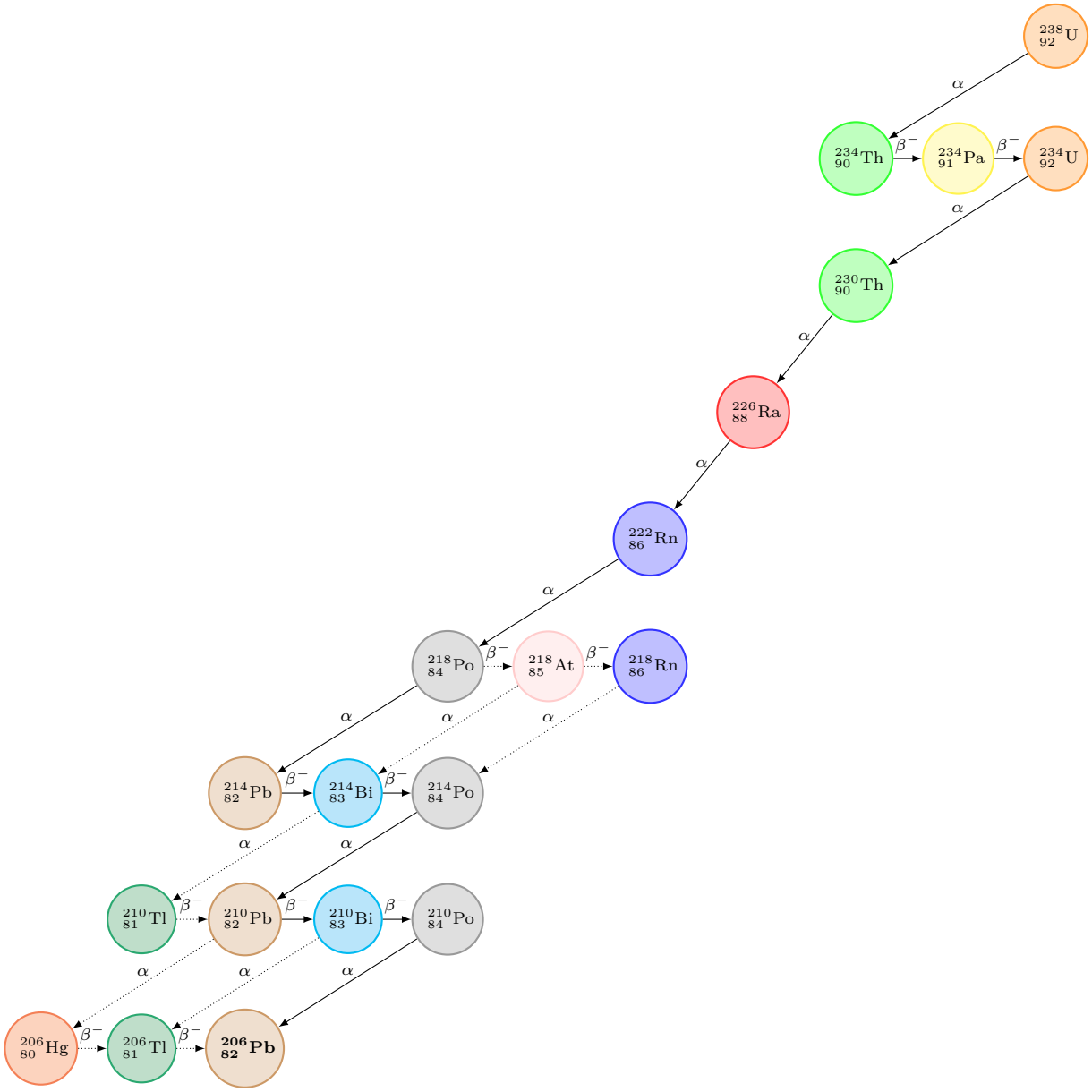


Fig. 1: Uranium decay chain.

3 P-system model

Population Dynamics P systems (PDP systems) [1] are a kind of P systems that combines the characteristics of both cell-like and tissue-like models. A PDP system is constituted by (i) a set of connected environments placed in the nodes of a directed graph (ii) identical cell-like structures of hierarchically arranged membranes placed inside each environment, (iii) a working alphabet of objects and (iv) a set of rules which describe how objects evolve and move inside the P systems ($\mathcal{R}$) and among the environments ($\mathcal{R}_\varepsilon$).

Formally, a Population Dynamics P system of degree (q,m) with $q, m \geq 1$, taking T time units, $T \geq 1$, is a tuple

$$(G, \Gamma, \Sigma, T, \mathcal{R}_\varepsilon, \mu, \mathcal{R}, \{f_{r,j} : r \in \mathcal{R}, 1 \leq j \leq m\}, \{\mathcal{M}_{ij} : 1 \leq i \leq q, 1 \leq j \leq m\}) \tag{5}$$

where:

- $G = (V, S)$ is a directed graph and $V = \{e_1, \ldots, e_m\}$ are the elements called environments.
- $\Gamma \cup \Sigma$ is the working alphabet.
- T is a natural number that represents the simulation time of the system.
- $\mathcal{R}_\varepsilon$ is a set of communication rules between environments of the form

$$(x)_{e_j} \xrightarrow{p(x,j,j_1,\ldots,j_h)} (y_1)e_{j1} \cdots (y_h)_{e_{jh}} \tag{6}$$

  where $x, y_1, \cdots y_h \in \Gamma, (e_j, e_{jl}) \in S (l = 1, \ldots, h)$ and $p_{(x,j,j1,\ldots,j_h)}(t) \in [0, 1]$, for each $t = 1, \ldots T$.
  The previous definition means that, when a communication rule is applied, object x contained in environment e_j passes to environments $e_{j1} \ldots e_{jh}$, possibly modified into objects $y_1, \ldots y_h$. If more than one rule can be applied to $(x)_{e_j}$, then the rule executed is chosen randomly according to the probabilities $p(x, j, j_1, ..., j_h)$.
- μ is the membrane structure of the cells contained in each of the m environments and each consisting on a set of q hierarchically arranged membranes injectively labeled by $1, \ldots q$. The skin membrane, or outer membrane is labeled by 1. The membranes can also have electrical charges or polarizations, $EC = \{0, +, -\}$.
- $\mathcal{R}$ is a set of evolution rules applied within each cell. They are of the form $r : u[v]_i^\alpha \to u'[v']_i^{\alpha'}$ where $u, v, u', v' \in M(\Gamma)$, $i \in 1, \ldots q$, and $\alpha' \in EC$.
- For each $r \in \mathcal{R}$ and for each $j, 1 \leq j \leq m$, $f_{r,j}$ is a computable function which satisfies that, for each $u, v \in M(\Gamma)$ all the rules $r \in \mathcal{R}$ whose left-hand side is (i, α, u, v) and the right-hand side have a polarization α', $\sum_{j=1} f_{r,j}(t) = 1$ $\forall t \leq T$.
- $\mathcal{M}_{1j}, \ldots, \mathcal{M}_{qj} \in M(\Gamma)$ are the initial multisets of objects for environments $j = 1, \ldots m$ placed inside the membranes $1, \ldots q$ of μ.

The tuple of multisets of objects present at any moment in the m environments and at each of the regions of the P systems (cell-like structures) constitutes the a

configuration of the system at any time. At the initial configuration of the system, all environments are assumed to be empty and all the membranes have neutral polarization.

The system evolves from one configuration to another at each time step by executing simultaneously all the applicable rules of the set $\mathcal{R} = \mathcal{R}_\varepsilon \cup \bigcup_{i=1}^{m} \mathcal{R}_{\Pi_j}$ [3] in a maximal way. When there are rules acting on overlapping left-hand sides, i.e. $u[v]_i^\alpha$, $u'[v']_i^\alpha$ where $u, u', v, v' \in M(\Gamma)$, $u \neq u' \vee v \neq v'$ and $u \cap u' \neq \emptyset \vee v \cap v' \neq \emptyset$, the rule which is executed is selected randomly according to the probability associated with each rule.

Finally, it is interesting to highlight the fact that a global clock is considered in the system, marking the time for the whole system, so the application of all rules (both from $\mathcal{R}_\varepsilon$ and $\mathcal{R}$) are synchronized in all environments.

[3] $\Pi_j = \{\Gamma, \mu, \mathcal{R}, \mathcal{M}_{1j}, \ldots, \mathcal{M}_{qj}\}$ denotes the P system in environment e_j and R_{Π_j}, the set of rules defined on the considered P system.

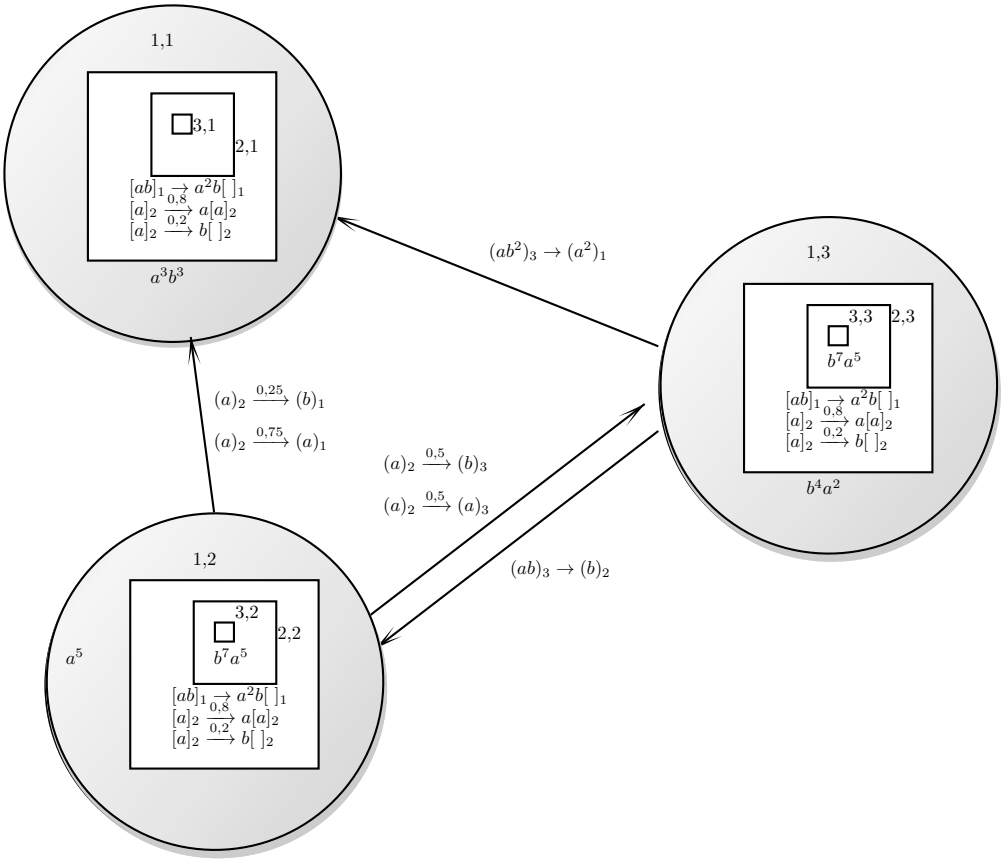


Fig. 2: A graphical example of a PDP system

4 Implementation

4.1 First model

The Step of time model is the simplest possible modelization of the Uranium decay problem, as it characterizes the half-lifetime, the characteristic time parameter, of the nuclear reactions using the clock steps of time defined in the membrane computing model. This first rought approximation is based on two assumptions:

- The nuclear reaction of a given element cannot begin until all the progenitor nuclei of this element have reacted.
- The duration of each reaction can be represented assigning different clock steps to every reaction.

The timescale of the reactions (characterized by the half-lifetime, $T_{1/2}$) involved in the network of nuclear reactions is huge, varying 21 orders of magnitude: the $T_{1/2}$ of the fastest reaction is of about $\approx 10^{-4}$ s while for the slowest $T_{1/2} \approx 10^{17}$ s.

In order to translate the half-lifetime of each reaction to a number of clock iterations, a logarithmic scale is considered. The number of clock iterations is therefore calculated assigning a scaled integer number to each reaction according to the "weight" of time for each reaction.

The implementation of this model is done therefore in a single cell-like membrane through rules of the type

$$\begin{aligned}
&[a_i \rightarrow\ a_{i+1}\ 1 \leq i \leq 7]_1 \\
&[a_8 \rightarrow\ b_1, z]_1 \\
&[b_i \rightarrow\ b_{i+1}\ 1 \leq i \leq 4]_1 \\
&[b_5 \rightarrow\ c_1, x, y]_1
\end{aligned}$$

when there's a single via decay or

$$\begin{aligned}
&[h_i \longrightarrow\ h_{i+1}\ 1 \leq i \leq 3]_1 \\
&[h_4 \xrightarrow{99{,}98\%}\ v_1, z]_1 \\
&[h_4 \xrightarrow{0{,}02\%}\ j_1, x, y]_1
\end{aligned}$$

when competition rules are considered.

The letter assignation is specified at Table 1, as it is the same as the one used in the probabilistic model. x, y and z represent α particles, e^- (electrons) and $\overline{\nu}_e$ (electron antineutrino).

As can be seen, after seven clock steps of computation the nuclei a decays into b. The nuclei b then waits 4 steps of computation before evolving. When competition rules apply, the nuclei h also waits a given number of computations after evolving according to a given probability.

Although it is not being a model really close to the real situation, with it one could obtain the results expected. The main application of the *Steps of time* model is then, to obtain the number of nuclei and particles of each kind once the process has ended, taking into account that some nuclei can decay by different vias according to a given probability. However, it cannot predict the number of nuclei of each kind after a given amount of time. The representation of the model is staggered-like, so it doesn't represent a continuous and soft process and therefore the approximation is not accurate.

To sum up, the biggest disadvantage in this modelization was that the next step in the chain had to wait until the previous one had finished completely. Therefore, although the simulation which implemented these rules approached reasonably the amount of particles gathered at the end of the process, it wasn't a good approach to reality, as in a real decay several reactions of the chain take place at the same time.

4.2 Second model

In order to implement the experiment of the Uranium Decay using the framework of *Membrane Computing*, it has been chosen the PDP system model restricted to a single environment containing a membrane structure composed by a sole membrane of neutral polarization $\alpha = 0$, so the system can be described as a P system

$$\Pi_1 = (\Gamma = \{a, b...z\}, \mu = [\,], \mathcal{R} = \{r_{1,i}, i = 1, ..., 38\}, \mathcal{M}_{1,q=1} = a^n)$$

Within this membrane, different objects, which represent the intermediate products in the decay chain, evolve in each step of the computation using PDP evolution rules of the type described in the previous section.

The alphabet of objects Γ is composed by all the intermediate products described in Table 1. A letter has been associated to each decay product in order to enable an easier modeling of the problem. As discussed in section 2, each decay mode α or β^- generates a different kind of particles, which also have a letter associated (x for α particles and z for particles generated in β^- decay), so the total number of particles obtained from the α and β^- decay can be accounted at the end of the computation.

In order to model the smooth and continuous decay of every specie in time, competition rules have been considered. For example, given the first reaction, the decay of U_{92}^{238} into Th_{90}^{234} through the α mode.

$$[a]_1 \xrightarrow{1-f_a} [a]_1$$
$$[a]_1 \xrightarrow{f_a} x[b]_1$$

where a and b are the letters associated to U_{92}^{238} and Th_{90}^{234} respectively. The rule applied will be chosen taking into account the probability associated with it, denoted by $f_{a,1} \equiv f_a$. From this it can be seen why it is so important that the probability sums up to 1.

The decay of a into b takes place through the α decay mode, so a second product x (α particle) is generated outside the membrane.

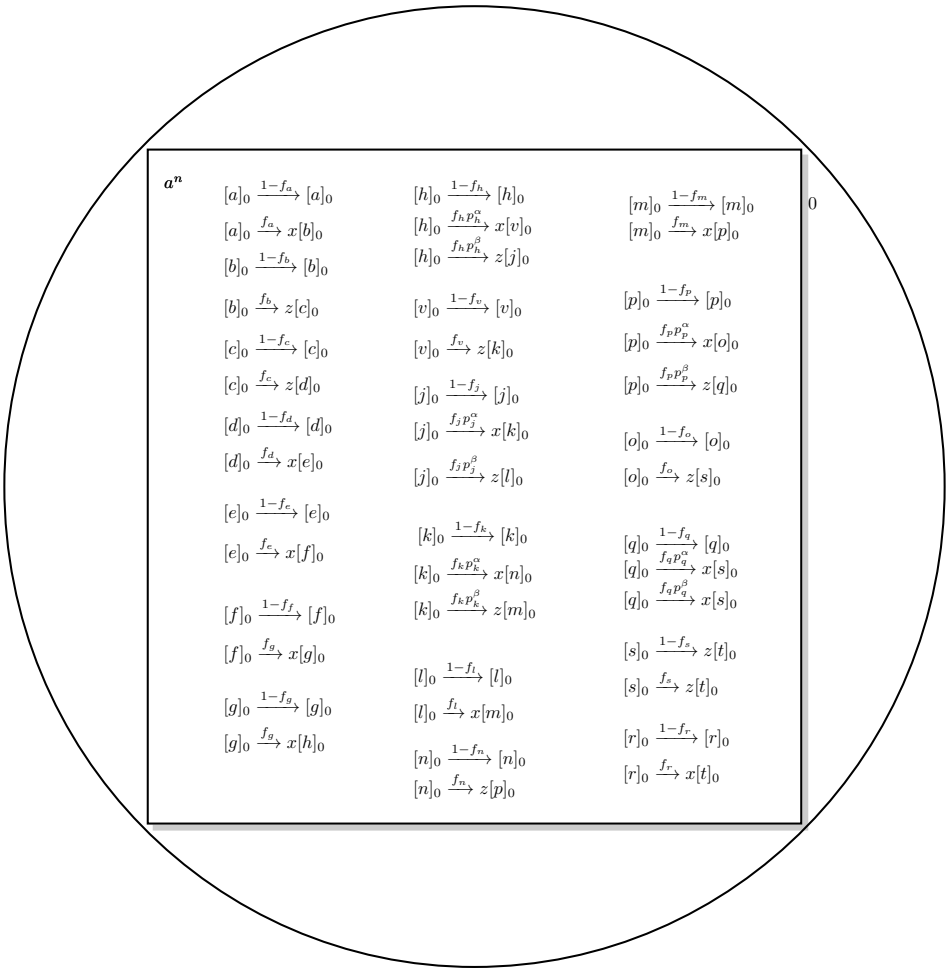


Fig. 3: Implementation of the model with the rules applied

As λ is a physical parameter constant in time and characteristic of each reaction, it has proved to be the most suitable magnitude to compute the the probability functions $f_{r,1}$. The functions $f_{r,1}$ have been calculated as the normalized logarithm of the scaled (> 1) λ 's.

The most significant difficulty in the assignation of $f_{r,1}$ to every transformation rule consisted on the great order of magnitude of the times of decay of the different reagents, as $\lambda \in [5{\cdot}10^{18}, 5{\cdot}10^{-3}]$ decays/s (nearly 21 orders of magnitude). Moreover, in other to compute the probabilities of each reaction, λ needed to be dimensionless and normalized in such a way that the probabilities of the rules with the same left-hand objects sum up to 1.

In order to solve these problems, some assumptions and approximations have been made. First of all, all values of λ in Table 1 have been divided between the order of magnitude of minimum λ. This way, we ensure that when the logarithm is applied the numbers will always be positive (as $\frac{\lambda}{\lambda_{\min}} > 1$), as well as granting that in logarithm scale, the order of the numbers is not changed. As have been mentioned, having such a great range of λ is truly inconvenient for doing the computations, so the rough solution that has been found consists in applying logarithms to the dimensionless quotient $\frac{\lambda}{\lambda_{min}}$. In other to obtain a global probability, which assigns probability equal to one to the rule with the greatest chance to occur, $\log(r)$ is divided by $\log(r)_{max}$ for every rule. When the decay mode is not unique, this function is also multiplied by the probability of occurrence p_r^{γ}, with $\gamma = \{\alpha, \beta\}$ representing the possible decay modes.

The method used for obtaining $f_{r,1}$ explained above, assigns a probability to the evolving transformation rules, which transforms a reactive into a different object. As has been explained in section 3, the sum of all probabilities applied over the same left-hand object must be one. As a consequence, the probability of occurrence of the non-evolution transformation rule r: $[a]_0 \xrightarrow{1-f_{r,1}} [a]_1$ has been computed as $1 - f_{r,1}$.

The rules start applying when at least one object of the left-hand side of a rule is generated and continue applying until all this kind of objects are consumed. So, as more intermediate products are generated, more reagents begin to evolve in each computation step, so after a given number of time steps several products of the decay chain will be evolving at the same time (modeled as discrete clock steps in the computation). In this manner, the dependence of the decay in the abundances of each reactive has been roughly simulated in a first approximation. Moreover, the problem of the first model, when a reactive couldn't begin to evolve until all reagents of the previous decay have been consumed has been solved. It's necessary to notice that although this model approximates better the decay as a continuous process, the steps of time are still discrete, represented by each computation step.

5 Code

```
@model<probabilistic>
def main()
{
  @mu=[]'1;
  @ms(1) = a*500000;

  [a]'1 --> [a]'1 ::  0.968;
  [a]'1 --> x[b]'1 :: 0.032;

  [b]'1 --> [b]'1 :: 0.467 ;
  [b]'1 --> z[c]'1 :: 0.533 ;
```

```
[c]'1 --> [c]'1 :: 0.378  ;
[c]'1 --> z[d]'1 :: 0.622 ;

[d]'1 --> [d]'1 ::  0.771;
[d]'1 --> x[e]'1 ::  0.229;

[e]'1 --> [e]'1 ::  0.747;
[e]'1 --> x[f]'1 ::  0.253;

[f]'1 --> [f]'1 ::  0.67;
[f]'1 --> x[g]'1 ::  0.33;

[g]'1 --> [g]'1 ::  0.43;
[g]'1 --> x[h]'1 ::  0.57;

[h]'1 --> [h]'1 ::  0.28;
[h]'1 --> x[v]'1 ::  0.71856;
[h]'1 --> z[j]'1 ::  0.00144;

[v]'1 --> [v]'1 ::  0.323;
[v]'1 --> z[k]'1 :: 0.677;
[j]'1 --> [j]'1 ::  0.183;
[j]'1 --> x[k]'1 ::  0.816183;
[j]'1 --> z[l]'1 ::  0.000817;

[k]'1 --> [k]'1 ::  0.317;
[k]'1 --> x[n]'1 ::  0.68285657;
[k]'1 --> z[m]'1 ::  0.00014343;
[l]'1 --> [l]'1 ::  0.108;
[l]'1 --> x[m]'1 ::  0.892;

[n]'1 --> [n]'1 :: 0.262 ;
[n]'1 --> z[p]'1 ::  0.738;
[m]'1 --> [m]'1 ::  0.0;
[m]'1 --> x[p]'1 ::  1.0;

[p]'1 --> [p]'1 ::  0.584;
[p]'1 --> x[o]'1 ::  0.4159999921;
[p]'1 --> z[q]'1 ::  0.000000007904;

[o]'1 --> [o]'1 ::  0.30;
[o]'1 --> z[s]'1 ::  0.70;
[q]'1 --> [q]'1 ::  0.436;
[q]'1 --> x[s]'1 ::  0.563999255;
[q]'1 --> z[r]'1 ::  0.000000745;

[s]'1 --> [s]'1 :: 0.286 ;
[s]'1 --> z[t]'1 ::  0.714;
```

```
  [r]'1 --> [r]'1 ::  0.502;
  [r]'1 --> x[t]'1 ::  0.498;
}
```

6 Results

The results obtained when running the P-lingua code with $a = 5 \cdot 10^5$ are shown in Figure 4 and Figure 5.

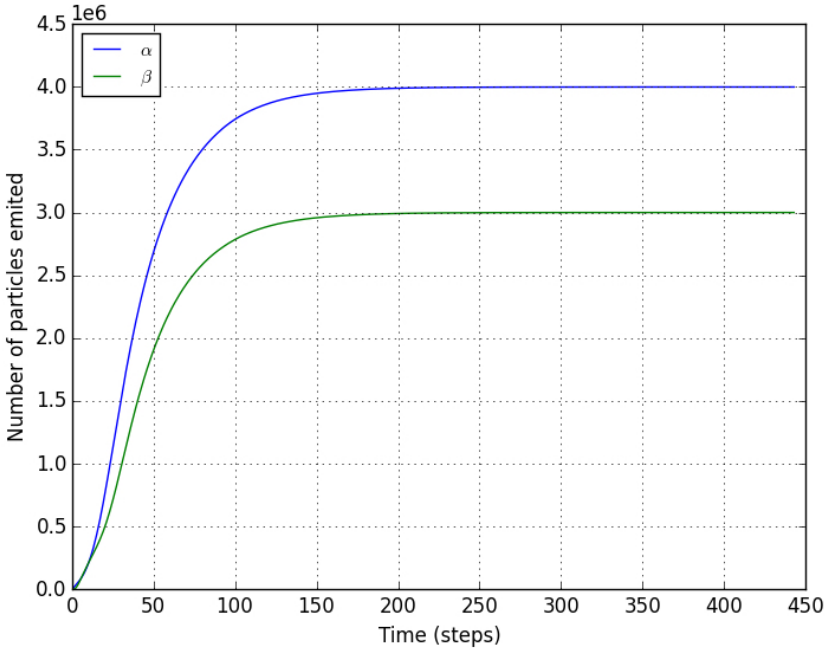


Fig. 4: Number of particles emitted through the different reactions, in logarithmic scale. It can be seen how both particles, α and the ones emitted through β^- reactions (named as β) reach an almost stationary value, with a larger final number of α particles, since α reactions take place more frequently. As reactions begin, a lot of the initial particles evolve giving their products but, as more reactions get to the final product, lead, less reactions take place at each step, so the number of particles emitted at each step reduces considerably, changing only one particle per step so, given the scale of the figure, this becomes imperceptible.

Both figures show the expected behavior of the Uranium decay chain, as they satisfy that

- after a large enough number of computation steps, all the emmited particles in the reaction (α and β^- particles) have reached a stable state.
- the decay chain takes place in a staggered way (we cannot see the discrete increments), as was sought in this second model.
- the slope of the evolution of each product matches the λ coefficients we implemented, showing a correct relationship between the probability functions and λ.
- the process is faster at generating particles in the beginning and at the end of the reaction, whereas the middle products last for a while.

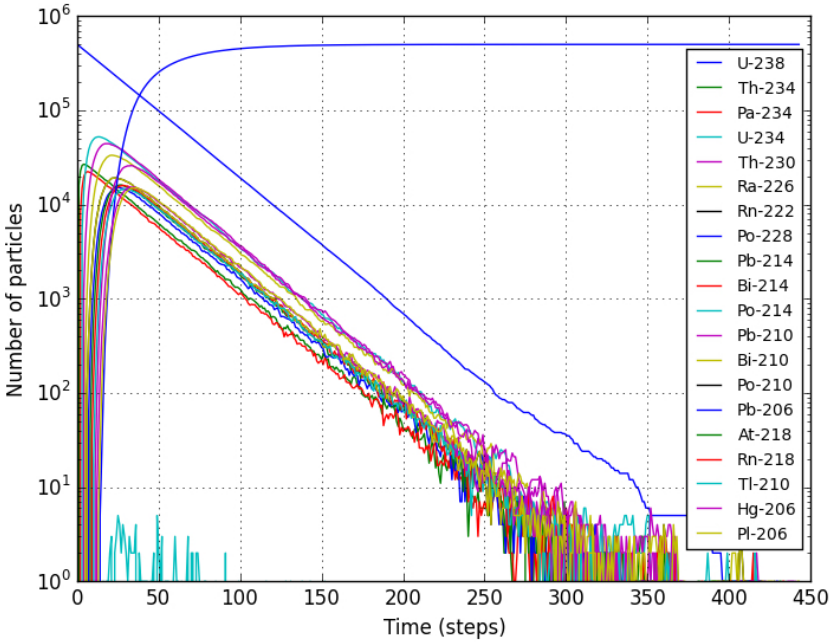


Fig. 5: Nuclei population. It is represented the number of particles at each step of time, in logarithmic scale. It can be seen that the number of initial particles (Uranium-238) decreases whereas the final product increses until it reaches an stationary value, the same as the initial number of particles, as expected. As reactions take place, the new elements are created, showing an impressive increase that slowly decreases then as reactions continue. As we reach more advanced stages (Steps of time $\approx$ 250 and more) a noise in the number of elements appears: this is because less reactions of the same elements take place simultaneously and so the number of particles changes sharply at each step, depending on if the reaction involving that specific element has taken place or not.

The fluctuations, which can be appreciated in the last steps of time in Figure 5, are probably due to the overlapping of different reactions in the advanced steps of the uranium chain which simultaneously produce and consume a certain nucleus, i.e., as the products evolve, more reactives are generated at the same time that they are evolving to the next product. That way, a dependence on the decay rate on the relative abundances of each reactive is appreciated.

7 Conclusions

Decay chains are based in a system of differential equations that once solved allow to obtain the products at each time, t. However, the solution is reached after solving a coupled system of numerous differential first order equations, which is computationally costly. The *MC* tools allow to reproduce the process and to obtain the expected final products just by making some slight approximations.

This means that competition rules which appear naturally in *MC* can assume the role of the bounds between differential equations almost trivially, so the mentioned system of differential equations does not need to be solved in order to simulate the real situation.

In addition, this article attempts different ways of implementing time in a decay process. Instead of modeling time as an independent parameter, which would be the model where indexes are used (considering steps of computation as time), it has finally been introduced as a part of the probability, given by the decay constant λ. The first method is really unefficient because the system wastes a lot of time just skipping processes (while indexes change) and so by this time, the program is not really working on the chain reaction itself.

Acknowledgments.

We would like to thank: Agustín Riscos-Núñez, Carmen Graciani and Mario J. Pérez-Jiménez from Universidad de Sevilla and Ricardo Graciani from Universitat de Barcelona for their valuable comments and suggestions, as well as for their reviews of the several versions of this paper; the 14th BWMC attendants for their patience and advice on our first ideas; and also Francesc Salvat, Assumpta Parreño, Ricardo Graciani and Bruno Juliá-Díaz from Universitat de Barcelona for offering us the opportunity to attend the Brainstorming.

References

1. M. Colomer, A. Margalida, and M. J. Pérez-Jiménez. Population Dynamics P System (PDP) Models: A Standardized Protocol for Describing and Applying Novel Bio-Inspired Computing Tools. *PLOS ONE*, 8(4):1–13, 2013.
2. K. S. Krane. *Introductory nuclear physics.* John Wiley and Sons Inc., New York, NY, 1987.

3. Live Chart of Nuclides: nuclear structure and decay data. https://www-nds.iaea.org/relnsd/vcharthtml/VChartHTML.html.
4. National Nuclear Data Center: Chart of Nuclides. http://www.nndc.bnl.gov/chart/.
5. N. Radionuclide Half-Life Measurements. http://www.nist.gov/pml/data/halflife-html.cfm.

On the cellular automata P systems and chain reactions

Marc Barroso Mancha

Universitat de Barcelona
Email: marc.barroso4@gmail.com

This paper has been written after attending the 14th Brainstorming Week on Membrane Computing as a physics student from University of Barcelona. The work presented here tries to represent what I have learned during that period, while trying to apply some of the most innovative concepts shown in the presentations attended during that week into some interesting situations.

1 Introduction

This brief paper aims to introduce the cellular automata P systems as an example of ESNP (extended spiking neural P systems) with transmittable states, and then apply the available rules to simulate a simple model of a random walk in 2D. Let's start by formally introducing the system. We have the usual definition:

$$\Pi = (O, Q, \sigma_1, ..., \sigma_n, in, out) \tag{1}$$

· where $O = \{a\}$, such that a is called the spike
· Q is a finit alphabet of states
· σ_i are neurons, defined by: $\sigma_i = (\alpha_i, n_i, f_i, R_i)$ where
  → $\alpha_i \in$ Q, is the initial state
  → n_i is the initial number of spikes
  → f_i is the state combining function
  → R_i is a finite set of rules
· in is the input neuron
· out is the output neuron

It is also necessary to explain how the rules work. The most general form is:

$$\alpha\, /a^c \rightarrow (t_1, a^{k_1}, \beta_1), ..., (t_m, a^{k_m}, \beta_m) \text{ such that } \alpha, \beta_j \in \mathrm{Q} \tag{2}$$

that just means that when the state of the given neuron is α, and if the neuron has exactly c objects 'a' inside, it should send k_j spikes to the neuron t_j, while also transmitting the state β_j, for $1 \leq j \leq m$. When multiple rules are applied into the same neuron in the same clock tick (i.e. different states are transmitted to the same neuron σ_i), the function f_i dictates how they should be combined to obtain a unique final state.

2 Cellular automata

A cellular automata is defined as a grid of cells, such as every cell can be in one of a finite number of states. The set of cells next to each one it's called its neighborhood. Then, at every generation, some fixed rules determine the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. One of the most important applications is the known as "Conway's Game of Life", originally created by the mathematician John Conway in 1970. Its importance is due to the fact that it can be proven to be a universal Turing machine (that is, anything that can be computed algorithmically can be computed within Conway's Game of Life - even the Game of Life itself!). Rudolf Freund and Sergiu Ivanov show in their presentation "Extended SNP Systems with States" that the ESNP with transmittable states is analogue to a cellular automata (with the only change we are going to do is talk about neurons instead of cells). More specifically, when only two states are considered, there is an easy set of rules that enables us to simulate the Game of Life. Thus, they showed that we can obtain universality with only two states in the ESNP paradigm. The description of such system can be found on their presentation.

3 Simple nuclear chain reaction model

In order to implement these ideas into something more tangible, we can think of the following situation: we have an organized grid of atoms (represented by the neurons), where all of them are stable (in the sense that no rules could be applied initially, that is, the system would halt immediately). In the position (i, j) we introduce an unstable atom that will explode in one clock tick, releasing n number of particles (represented by the objects), that will go to any of its neighbors, making them unstable, and thus propagating some kind of state (generating the chain reaction). This is known as a two dimensional random walk. Some attractive

studies can consist of varying the number of objects an explosion yields, observing whether the reaction consume all the possible atoms or not (if an atom that has already exploded is considered to be destroyed insted of being replaced), or considering some time of interaction in every step. Another interesting behavior (also more difficult to implement) would be trying to change the geometry of the system (so the number of neighbors could vary from atom to atom), and see if you could get more efficiency some way or another.

This is obviously the first iteration of the model one can think of: further complications can be considered, as making the grid in 3D (just by adding layers upon layers of atoms), or even trying to add probabilities so further atoms than the more direct neighbors have a chance of becoming unstable. This model can be used to simulate the path of a photon that emerges from the Sun's core and is trying to reach the surface. A very simple model has been implemented at what aims to be an ESNP simulator, programmed in Python. It consists of a square grid of arbitrary dimensions, and an unstable atom in the middle. The initial reaction lasts two ticks, and send one spike in two random different directions. For the sake of simplicity, we have used three different states: 0 - stable atom, 1 - unstable atom, 2 - already exploded atom, even if two were already enough, as explained before. We understand that this model has some difficulties (we only used the most direct neighbors; this can be extended to have probabilities for all 8 adjoin neurons), and can even be too simplified (we know that atoms won't be organized in a rectangular grid, as shown here, but will have some kind of spatial distribution). But while modeling this problem, we found out that this could also be implemented as some kind of A^* path-finding algorithm or some kind of path algorithm, only halting the computation when a certain point is reached or if no rules have been applied, and then letting a lot of systems run in parallel. With a comparison between the number of rules used, one can actually get a good representation of the optimal route. Could we explot from the fact that we can acces more states and take advantage of it, working under the P-systems paradigm? Further research and modeling must be done to answer this.

Improving Simulations of Spiking Neural P Systems in NVIDIA CUDA GPUs: CuSNP

[1]Jym Paul Carandang, [1]John Matthew B. Villaflores, [1]Francis George C. Cabarle, [1]Henry N. Adorna, [2]Miguel Ángel Martínez-del-Amor

[1]Algorithms and Complexity
Department of Computer Science
University of the Philippines Diliman
Diliman 1101 Quezon City, Philippines;
email: jacarandang@gmail.com, matthewvillaflores@gmail.com, fccabarle@up.edu.ph
hnadorna@up.edu.ph
[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mdelamor@us.es

Summary. Spiking neural P systems (in short, SN P systems) are parallel models of computations inspired by the spiking (firing) of biological neurons. In SN P systems, neurons function as spike processors and are placed on nodes of a directed graph. Synapses, the connections between neurons, are represented by arcs or directed endges in the graph. Not only do SN P systems have parallel semantics (i.e. neurons operate in parallel), but their structure as directed graphs allow them to be represented as vectors or matrices. Such representations allow the use of linear algebra operations for simulating the evolution of the system configurations, i.e. computations. In this work, we continue the implementations of SN P systems with delays, i.e. a delay is associated with the sending of a spike from a neuron to its neighbouring neurons. Our implementation is based on a modified representation of SN P systems as vectors and matrices for SN P systems without delays. We us massively parallel processors known as graphics processing units (in short, GPUs) from NVIDIA. For experimental validation, we use SN P systems implementing generalized sorting networks. We report a speedup, i.e. the ratio between the running time of the sequential over the parallel simulator, of up to approximately 51 times for a 512-size input to the sorting network.

Key words: Membrane computing; Spiking neural P system; NVIDIA CUDA; graphics processing units

1 Introduction

Membrane computing, initiated in [15], involves models of computations inspired by structures and functions of various types of biological cells. Models in membrane computing are known as membrane or P systems. The specific type of P system we consider in this work are spiking neural P systems, in short, SN P systems. SN P systems, first introduced in [7], are inspired by the pulse coding of information that occur in biological neurons. In pulse coding from neuroscience, pulses known as *spikes* are indistinct, so information is instead encoded in their multiplicity or the time step(s) they are emitted.

SN P systems are known to be computationally universal (i.e. equivalent to Turing machines) in both generative (an output is given, but not an input) and accepting (an input is given, but not an output) modes. SN P systems can also solve hard problems in feasible (polynomial to constant) time. Another active line of investigation on the computability and complexity of SN P systems is taking mathematical and biological inspirations in order to create new variants, e.g. asynchronous operation, weighted synapses, rules on synapses, structural plasticity. We do not go into details, and we refer to [7, 9, 14, 17, 5] and references therein.

Software simulators for P systems, whether sequential or parallel, have been provided. Sequential simulators include for example those implemented using PLingua, a programming language designed for P systems, e.g. [11]. Simulators using massively parallel processors known as graphics processing units (in short, GPUs) for cell-like P systems as well as SN P systems include [13], a comprehensive survey in [12], and [4, 10].

In this work, we report our ongoing efforts to simulate SN P systems on GPUs manufactured by NVIDIA. In particular, our contributions in this report are as follows: (a) modified matrix representation of [18] in order to be able to simulate SNP systems with delays, (b) the entire simulation of SN P systems with delays is now performed in the GPU, compared to a small portion of the simulation in [4], and (c) using generalized sorting network of SN P systems, we report up to 51 times speedup in our experiments with a 512 input size network. A preliminary version of this work is available in [6].

This work is organized as follows: Section 2 provides preliminaries for the remainder of this work; Section 3 provides the definition of SN P systems as well as their linear algebra representations; Section 4 provides an overview of the NVIDIA CUDA architecture; Section 5 provides the simulation algorithm for our work; Section 6 provides experimental results for the sequential and parallel simulators; Finally, Section 7 provides conclusions from our work as well as future research directions.

2 Preliminaries

We recall some formal language theory (available in many monographs). We only briefly mention notions and notations which will be useful throughout the paper.

We denote the set of natural (counting) numbers as $\mathbb{N} = \{0, 1, 2, \ldots\}$, where $\mathbb{N}^+ = \mathbb{N} - \{0\}$. Let V be an alphabet, V^* is the set of all *finite* strings over V with respect to *concatenation* and the *identity element* λ (the empty string). The set of all non-empty strings over V is denoted as V^+, so $V^+ = V^* - \{\lambda\}$.

A language $L \subseteq V^*$ is *regular* if there is a regular expression E over V such that $L(E) = L$. A regular expression over an alphabet V is constructed starting from λ and the symbols of V using the operations union, concatenation, and $+$. Specifically, (i) λ and each $a \in V$ are regular expressions, (ii) if E_1 and E_2 are regular expressions over V then $(E_1 \cup E_2)$, E_1E_2, and E_1^+ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each expression E we associate a *language* $L(E)$ defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$ for all $a \in V$, (ii) $L(E_1 \cup E_2) = L(E_1) \cup L(E_2)$, $L(E_1E_2) = L(E_1)L(E_2)$, and $L(E_1^+) = L(E_1)^+$, for all regular expressions E_1, E_2 over V. Unnecessary parentheses are omitted when writing regular expressions. If $V = \{a\}$, we simply write a^* and a^+ instead of $\{a\}^*$ and $\{a\}^+$. If $a \in V$, we write $a^0 = \lambda$.

3 Spiking Neural P Systems

We assume some familiarity with membrane computing concepts, widely available online (e.g. [1]) or in print (e.g. [16]). First, we formally define SN P systems, followed by linear algebra representations of their computations.

3.1 Spiking Neural P System

A Spiking Neural P system Π is of the form:

$$\Pi = (O, \sigma_1, ..., \sigma_m, syn, in, out)$$

1. $O = \{a\}$ is the alphabet containing a single symbole (the spike);
2. $\sigma_1, ..., \sigma_m$ are neurons, of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$ where:
   a) $n_i \geq 0$ is the initial number of spikes contained in σ_i.
   b) R_i is a finite set of rules of the following two forms:
      i. $E/a^c \rightarrow a^p; d$ where E is a regular expression over O and $c \geq p \geq 1, d \geq 0$.
      ii. $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a^p; d$ of type (i) from R_i, we have $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, ..., m\}x\{1, 2, ..., m\}$ with $i \neq j$ for all $(i, j) \in syn, 1 \leq I, j \leq m$ (synapses between neurons);
4. $in, out \in \{1, 2, ..., m\}$ indicate the input and the output neurons, respectively.

The rules of type (i) as mentioned in the construct of neurons are firing (or spiking) rules while the type (ii) are called forgetting rules. An SN P system whose firing rules have $p = 1$ is said to be of the standard type (non-extended). Given a spiking rule, it is applied as follows. If a neuron σ_i contains k spikes, and

$a^k \in L(E), k \geq c$, then the rule $E/a^c \rightarrow a^p; d \in R_i$ can be applied. This means we remove c spikes so that $k - c$ spikes remain in σ_i, the neuron is then fired and produces p spikes (1 in the case of standard SN P systems) after d time units.

Spikes are fired after $t + d$ where t is the current time step of the computation. For the case that $d = 0$, the spikes are fired immediately. When the time step of the computation is between t and $t + d$, we say that the neuron σ has not fired the spike yet and σ is closed, meaning it cannot receive spikes from other neuron connected to it. In the case that a neuron with an in-going synapse to σ fires, the spike(s) is(are) lost. During the time step $t+d$, the spikes are fired, and the neuron is now open to receive spikes. At $t + d + 1$ the neuron can begin applying rules. When neuron σ_i emits the spike, the spikes reach immediately all neuron σ_j such that $(i, j) \in syn$ and σ_j is open.

A forgetting rule is applied as follows. If the neuron σ_i contains exactly s spikes, then the rule $a^s \rightarrow \lambda$ from R_i can be applied, meaning all of the s spikes are removed from σ_i.

For rules of the form $E/a^c \rightarrow a^p; d$ of type (1) where $E = a^c$, we write it in the shortened form $a^c \rightarrow a^p; d$. There are cases when two or more rules are applicable in a step of the computation, at these cases, only one rule is applied and is non-deterministically chosen. However, by definition, it is impossible to have a spiking rule and a forgetting rule to be applied at the same time. In short, for each neuron, at most one rule will be applied at a time unit.

A *configuration* or state of the system at time t can be described by $C_t = \langle r_1/k_1, \ldots, r_m/k_m \rangle$ for $1 \leq i \leq m$, where neuron i contains $r_i \geq 0$ spikes and remains closed for k_i more steps. The initial configuration of the system is therefore $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$. Rule application provides us a *transition* from one configuration to another. A computation is any (finite or infinite) sequence of configurations such that: (a) the first term is the initial configuration C_0; (b) for each $n \geq 2$, the nth 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*, i.e. a configuration where all neurons are open and no rule can be applied.

Two common ways to interpret output of an SN P system are as follows: (1) obtaining the time interval between exactly the first two steps when the output neuron σ_{out} spikes, e.g. number $n = t_n - t_1$ is computed, where σ_{out} produced its first two spikes at steps t_1 and t_n; (2) counting the number of spikes produced by σ_{out} until the system halts. Note that for (1) the system need not halt, since we only consider the first two steps when σ_{out} spikes. In this work, we consider systems that produce their output using the manner given in (2).

Spiking Neural Systems are usually represented as a directed graphs. Figure 1 shows an example of an SN P system with 3 neurons. This system is formally defined as:

$$\Pi = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \{(1, 2), (2, 1), (1, 3), (3, 1)\}, 1) \text{ where:}$$

1. $\sigma_1 = (0, \{a/a \rightarrow a; 0, a^2/a^2 \rightarrow \lambda\})$

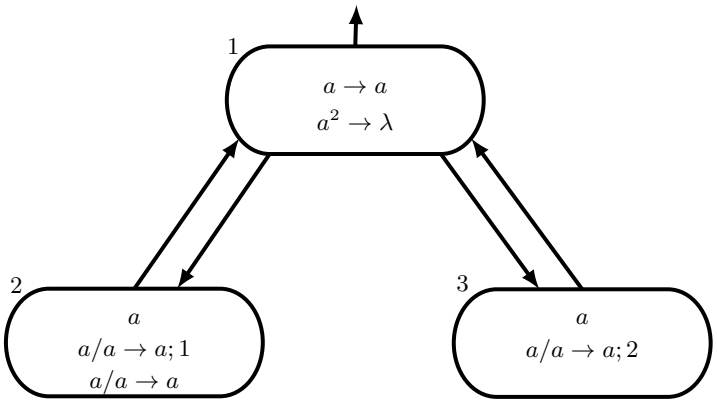


Fig. 1: Example of an SN P System

2. $\sigma_2 = (1, \{a/a \to a; 1, a/a \to a\})$
3. $\sigma_3 = (1, \{a/a \to a; 2\})$

Notice that in σ_1, the rule $a/a \to a; 0$ was written $a \to a$ which is convention when $L(E) = a^p$, we can only write a^p. Also, we do not write the delay d if it is equal to 0. In this example, some rules used this convention while others did not to show the interchangeability between the two.

In Figure 1, there are three neurons, two of which have initial spikes (neurons σ_2 and σ_3). Neurons 1 and 3 have synapses between each other, likewise between σ_1 and σ_2. A synapse exist from σ_1 to the environment to indicate it is the output neuron. In the first step of computation, σ_1 will not fire since it does not contain any spike. Neuron 3 will fire and consume its spike but it will close and will not transmit a spike yet since it has a delay. For σ_2, we non-deterministically choose which rule to apply. Assuming we choose the rule $a/a \to a; 1$, the neuron will consume all of its spikes and similar to σ_3, it will close and will not send any spikes yet.

Therefore, after the first step of computation, there are no spikes in the system. In the next step of computation, no rules will spike since there are no spikes in the system but σ_2 will release a spike since the delay is done. After this step, σ_1 will contain a spike. At the third step of computation, σ_3 will release a spike that was delayed and σ_3 becomes open also. Neuron 1 will send a spike to both σ_2 and σ_3. After this step, each neuron will have one spike. Assuming we always choose to apply the first rule of σ_2, this SN P system will cycle every three steps of computation.

Going back to the first step of computation, assuming we choose to apply the second rule in σ_2, in the next configuration, σ_1 will have one spike while σ_2 and σ_3 will have zero. Neuron 1 will then fire, sending a spike to σ_2. Neuron 3 will not receive a spike since it is currently closed. In the third step of computation, assuming we selected the second rule in σ_2 again, σ_3 will send the delayed spike and

σ_2 will spike immediately. After this step, σ_1 will have two spikes. The forgetting rule will be applied and the computation halts.

3.2 Matrix Representation of Spiking Neural P systems with Delay

In [18], a matrix representation of SN P system without delay was introduced. In this work we introduce modifications to this representation which allows us to devise simulations for SN P systems with delay. Let Π be an SN P system with delay having m neurons and n rules. We use the following definitions, modified from [18], to represent our simulation algorithm:

Definition 1: (Configuration Vector). The vector $C^{(k)} = \langle c_1, c_2, ..., c_m \rangle$ is called the configuration vector at the kth step of computation where each $c_i, i = 1, 2, ..., m$, is the amount of spikes neuron i contains.

Specifically, the vector $C^{(0)} = \langle c_1, c_2, ..., c_m \rangle$ is called the initial configuration vector of Π, where c_i is the amount of the initial spikes present in neuron $\sigma_i, i = 1, 2, ..., m$ before the computation starts.

Definition 2: (Spiking Vector). Let $C^{(k)} = \langle c_1, c_2, ..., c_m \rangle$ be the kth configuration vector of Π. Assume a total order $d : 1, ..., n$ is given for all the n rules, so the rules can be referred to as $s_1, ..., s_n$. A spiking vector $S^{(k)}$ is defined as follows:

$$S^{(k)} = \langle s_1^{(k)}, s_2^{(k)}, ..., s_n^{(k)} \rangle$$

$$s_i^{(k)} \begin{cases} 1, \text{if the regular expression } E_i \text{ of rule } r_i \text{ is satisfied by} \\ \text{the numbers of spikes } c_j \text{ (rule } r_i \text{ is in neuron } \sigma_j \text{) and} \\ \text{rule } r_i \text{ is chosen and applied;} \\ 0, \text{otherwise} \end{cases}$$

Definition 3: (Status Vector). The vector $St^{(k)} = \langle st_1, st_2, ..., st_m \rangle$ is called the status vector at the kth step of computation where each $st_i, i = 1, 2, ..., m$, determines the status of the neuron m.

$$st_i = \begin{cases} 1, \text{if neuron } m \text{ is open} \\ 0, \text{if neuron } m \text{ is closed} \end{cases}$$

Note that a neuron is said to be closed when a rule with a delay is activated and is waiting for that delay to become zero. A neuron that is closed may not receive any incoming spikes.

Definition 4: (Rule Representation). The set $R = \{r_1, r_2, ..., r_n\}$ is the set of rules where each r_i, $i = 1, 2, ..., n$ is a vector representing each rule in Π. Each r_i is defined as follows.

$$r_i = \langle E, j, d', c \rangle$$

where:

1. E is the regular expression for rule i
2. j is the neuron that contains the rule r_i
3. $d' = \begin{cases} -1, \text{if the rule is inactive (i.e. not applied)} \\ 0, \text{if the rule is fired} \\ \geq 1, \text{if the rule is currently on delay (i.e. } \sigma_j \text{ is closed)} \end{cases}$
4. c is the number of spikes that neuron σ_j will consume if it applies r_i.

Definition 5: (Delay Vector). The delay vector $D = \langle d_1, d_2, ..., d_n \rangle$ contains the delay value for each rule $r_i, i = 1, 2, ..., n$ in Π.

Definition 6: (Loss Vector). The vector $LV^{(k)} = \langle lv_1, lv_2, ..., lv_m \rangle$ is the loss vector where each lv_i, for each neuron $\sigma_i, i = 1, 2, ..., m$, contains the number of spikes consumed, c, if σ_i applies r_i at step k.

Definition 7: (Gain Vector). The vector $GV^{(k)} = \langle gv_1, gv_2, ..., gv_m \rangle$ is the gain vector which contains the total number of spikes gained, gv_i, for each neuron $\sigma_i, i = 1, 2, ..., m$, at the kth step of computation not considering whether the neuron is open or closed.

Definition 8: (Transition Vectors). Given the total order $d : 1, ..., n$ for all the n rules, the transition vector Tv of the system Π, is a set of vectors defined as follows:

$$Tv = \langle tv_1, ..., tv_n \rangle$$

$$tv_i = \langle p_1, ..., p_m \rangle$$

$$p_j = \begin{cases} p, \text{ if rule } r_i \text{ is in neuron } \sigma_s (s \neq j \text{ and } (s, j) \in syn) \\ \text{ and it is applied producing } p \text{ spikes;} \\ 0, \text{if rule } r_i \text{ is in neuron } \sigma_s (s \neq j \text{ and } (s, j) \notin syn). \end{cases}$$

The set Tv replaces the spiking transition matrix used in [18] since in [18], each matrix entry can contain values either from spikes consumed or produced by each neuron with respect to rules of other neurons. Transition vectors, however, contain only the p spikes gained from other neurons, otherwise 0.

Definition 9: (Indicator Vector) The indicator vector $IV^k = \langle iv_1, iv_2, ..., iv_m \rangle$ will be multiplied to Transition Vector, $Tv \cdot IV^k$, in order to get the net number of spike a neuron will get not considering a neuron's status.

Definition 10: (Net Gain Vector). Let $LV^{(k)} = \langle lv_1, lv_2, ..., lv_m \rangle$ be the kth loss vector vector, $GV^{(k)} = \langle gv_1, gv_2, ..., gv_m \rangle$ is the kth gain vector vector, and $St^{(k)} = \langle st_1, st_2, ..., st_m \rangle$ is the kth status vector vector. The net gain vector at step k is defined as

$$NG^{(k)} = GV^{(k)} \otimes st^{(k)} + LV^{(k)}$$

4 NVIDIA CUDA

CUDA or Compute Unified Device Architecture is a parallel programming computing platform and application programming interface model developed by NVIDIA

[3]. CUDA allows software developers to use a CUDA enabled graphics processing unit(GPUs) for general purpose processing, an approach known as GPGPU.

Functions that execute in the GPU, known as kernel functions, are executed by one or more threads arranged in thread blocks. In the CUDA programming model, the GPU is often referred to as the device, while the CPU is referred to as the host. The CPU (the host) is the one performing kernel function calls to be executed on the device (the GPU). CUDA works on an SPMD principle or the single program multiple data principle. That is, similar code runs on the threads, and the threads can be accessing multiple (possibly different values of) data. CUDA also has implements a memory hierarchy, similar to how there exist memory hierarchies and cache organizations within the CPU.

The host and the device have a separate memory space so copying data from the host and device memory may be necessary. The memory hierarchy for the device includes its global memory, shared memory, and constant memory. Each memory type has its own advantage such as bandwidth size, access speed and control. The developer has the ability to fine tune the memory use and type for kernel functions in order to optimize computations. Poor memory management can cause bottlenecks, e.g. when copying memory from device to host and vice-versa often. A good memory access pattern would be transferring all the required data to the device and do all the processing within the device before returning the computation result to the host. This access pattern prevents the high-latency transfers between the device and the host.

Optimizing block structure is also important to maximize the parallel structure of the GPU. The physical execution of threads occur in warps of 32 and a not optimal block structure could result in serializing of execution. Lastly, the kernel code itself must be optimized to maximize the use of the threads. GPUs are often used to accelerate computations involving highly parallelizable tasks such as linear algebra operations, while the CPU is more efficient with highly sequential tasks.

5 Algorithm for simulating SN P Systems with delay

In our simulation of SN P systems with delays, we consider the following *cases* when applying rules in the system:

1. Trivially, the spikes contained in the neuron do not satisfy the regular expression E of a rule, hence the rule is not applied.
2. When E of a rule is satisfied and the rule applied with a delay $d > 0$, hence c spikes are consumed (the neuron becomes closed) and begin the countdown of the delay until delay becomes 1.
3. When the countdown for the delay in Case 2 reaches 0 (neuron becomes open), we consider the *net* number of spikes: those spikes produced to other neurons and received from other neurons, possibly including previous spikes in the neuron before the neuron closed.

4. When a rule applied has $d = 0$ which is similar to Case 2 and 3 except that we do not perform any countdown and the neuron for the rule does not become closed.

We also introduce the operation $\otimes$, where $C = A \otimes B$ is the element-wise multiplication of vectors A and B, i.e., for each $x_i \in A$, $y_i \in B$, $z_i \in C$, $z_i = x_i * y_i$. For example, given vector $A = \langle 2, 6, -4, 3\rangle$ and $B = \langle 5, 6, 1, 2\rangle$, we have $A \otimes B = \langle 10, 36, -4, 6\rangle$.

The main simulation algorithm is given in Algorithm 1, which refers to definitions given in Section 3.2. The algorithm is devised to return or produce the $(k+1)$th configuration of an SN P system, given the current step k.

1: **procedure** SIMULATE SNP $(C^{(k)}, R, Tv, St^{(k)})$
2: Reset$(Lv^{(k)})$
3: Reset$(Gv^{(k)})$
4: Reset$(NG^{(k)})$
5: Reset$(Iv^{(k)})$
6: Compute $S^{(k)}$
7: **for** $r_i = \langle E, j, d', c\rangle \in R$ **do** ▷ Check for the cases
8: **if** $S_i^{(k)} = 1$ **then** ▷ Case 2
9: $Lv_j^{(k)} \leftarrow c$
10: $d' \leftarrow d_i$
11: $St_j^{(k)} \leftarrow 0$
12: **if** $d' = 0$ **then** ▷ Case 4
13: $Iv_j^{(k)} \leftarrow 1$
14: $St_j^{(k)} \leftarrow 1$
15: **end if**
16: **else if** $d' = 0$ **then** ▷ Case 3
17: $Iv_j^{(k)} \leftarrow 1$ ▷ Set indicator bit to 1
18: $St_j^{(k)} \leftarrow 1$
19: **end if**
20: **end for**
21: $Gv^{(k)} \leftarrow Tv * Iv^{(k)}$
22: $NG^{(k)} \leftarrow Gv^{(k)} \otimes St^{(k)} + Lv^{(k)}$
23: $C^{(k+1)} \leftarrow C^{(k)} + NG^{(k)}$
24: **for** $r_i = \langle E, j, d', c\rangle \in R$ **do** ▷ Countdown
25: **if** $d' \neq -1$ **then**
26: $d' \leftarrow d' - 1$
27: **end if**
28: **end for**
return $C^{(k+1)}$
29: **end procedure**

Algorithm 1: Simulation of Π from $C^{(k)}$ to $C^{(k+1)}$.

Note that Reset(X) for some vector X resets the vector to a 0 vector to prevent its previous values from interfering with the next iteration of the simulation (i.e. the next step of the computation). Also, the algorithm does not discriminate between the firing and the forgetting rule. That is, a forgetting rule is simply treated as a firing rule that doesn't produce any spikes.

Algorithm 1 takes in an SN P system Π represented using definitions in Section 3.2. The algorithm accepts the initial configuration vector $C^{(0)}$, the rules representation R, the transition vectors Tv, and the status vector $St^{(k)}$. After determining the Spiking Vector $S^{(k)}$, details provided below in Algorithm 2, we check the three cases defined previously (except the trivial case 1). If case 2 applies, where a rule is applied, we set the Loss Vector $Lv^{(k)}$ to c (for the corresponding neuron that contains the rule). Then the counter is started by setting the $d' = d_i$. We then make sure only one applied rule in a neuron will modify a single element of $Lv^{(k)}$, based on the semantics of rule application of SN P systems. The element of $St^{(k)}$ corresponding to the neuron is set to 0, hence closing the neuron.

If the delay of the rule is 0 (case 4), we set the corresponding $Iv_j^{(k)}$ to 1 and open the neuron. Also, the corresponding Status vector element $St_j^{(k)}$ is set to 1. For case 3, we set Tv_j to 1 and open the neuron again by setting $St_j^{(k)}$ to 1. We obtain the Gain Vector $GV^{(k)}$ by multiplying the Transistion Vectors Tv (the rules that released their spikes, i.e. rules where case 3 and 4 applies) to $Iv^{(k)}$. We obtain the Net Gain vector $NG^{(k)}$ using element-wise multiplication of $GV^{(k)}$ and $St^{(k)}$, then adding $Lv^{(k)}$.

The Status Vector acts as a selector where a neuron receives spikes based on its status, after consumed spikes are removed. Finally, we compute for $C^{(k+1)}$ by adding $C^{(k)}$ to $NG^{(k)}$. We reduce each d' for $0 \leq i \leq n$ which signifies the count down. On selecting the Spiking Vector $S^{(k)}$, Algorithm 2 is used.

```
1: procedure COMPUTE S^(k) (C^(k), R^(k))
2:     array n_tmp(0 : m)                          ▷ Initialize an array of size m
3:     for r_i ∈ R do
4:         if St_j^(k) == 0 then
5:             S_i^(k) ← 0                         ▷ Neuron that owns the rule is closed
6:         else if n_tmp_j == 1 then
7:             S_i^(k) ← 0                         ▷ Neuron that already has a rule that applied
8:         else
9:             if L(E_i) matches C_j^(k) then
10:                S_i^(k) ← 1                     ▷ E of rule matches with C^(k)
11:                n_tmp_j ← 1
12:            else
13:                S_i^(k) ← 0                     ▷ E does not match with C^(k)
14:            end if
15:        end if
16:    end for
```

17: **end procedure**

Algorithm 2: Computation of Spiking Vector $S^{(k)}$.

Note that Algorithm 2 only computes for one valid spiking vector of the system, since we only simulate deterministic systems.

6 Results

Algorithms 1 and 2 were implemented in both sequential and parallel code. C++ was used for the sequential implementation while CUDA C for the parallel implementation. The regular expression E in the rules are represented as integers: a^k is stored as k and a^* is stored as -1. We are currently at work in order to include simulations of more general regular expressions, e.g. $a^i(a^j)^*, i, j \geq 0$. Software for this work is available at [2]. The sequential and parallel implementations simulate deterministic SN P systems with delays.

For the CUDA implementation, computations in Algorithms 1 and 2 are performed in the GPU, until simulation halts. The recommended technique or workflow for GPU computing is to initialize inputs at the host (i.e. the CPU), copy inputs to the device (i.e. the GPU), finish all computations in the device, and finally copy results back to the host. This workflow for GPU computing is necessary in order to prevent the overhead (time delays incurred) of data transfer from host to device.

SN P systems implementing generalized sorting networks (provided in [8]) were used as inputs. The input sizes for the sorting networks are of the form 2^n for $n = 1, 2, \ldots, 9$, i.e. from 2 up to 512. The values to be sorted are natural numbers between 0 and 99, randomly generated. For the case of input sizes greater than 100, there will be repetitions of several numbers to be sorted.

The machine set-up for the experiments performed in this work runs an Ubuntu 15.04 64-bit operating system, with an Intel Core i7-4790 CPU with maximum frequency of 4 GHz, and 16 GBytes of memory. The GPU of the set-up is an NVIDIA GeForce GTX 750 with 512 CUDA cores (Kepler microarchitecture) with maximum frequency of 1084 MHz and 2047 MBytes of memory.

The SN P systems used as inputs for both sequential and parallel simulators are the systems implementing generalized sorting networks in [8]. In particular, a sorting network has n input neurons in order to sort n natural numbers. A sorting network of input size n has input neurons $\sigma_1, \ldots, \sigma_n$ containing $r_1, \ldots, r_n$ spikes initially, where the values $r_1, \ldots, r_n$ are the numbers to be sorted (delays are not used in this case).

Figures 2 and 3 illustrate the running time (vertical axis) versus input size (horizontal axis) of both the sequential (i.e. C++SNP) and parallel (i.e. CuSNP) simulators. The 9 inputs, from 2 up to 512, were separated into two charts (given by Figures 2 and 3) since the running time of C++SNP for a 512-input sorting network is much greater (approximately 10 minutes) than the remaining smaller input sizes (under 2 minutes).

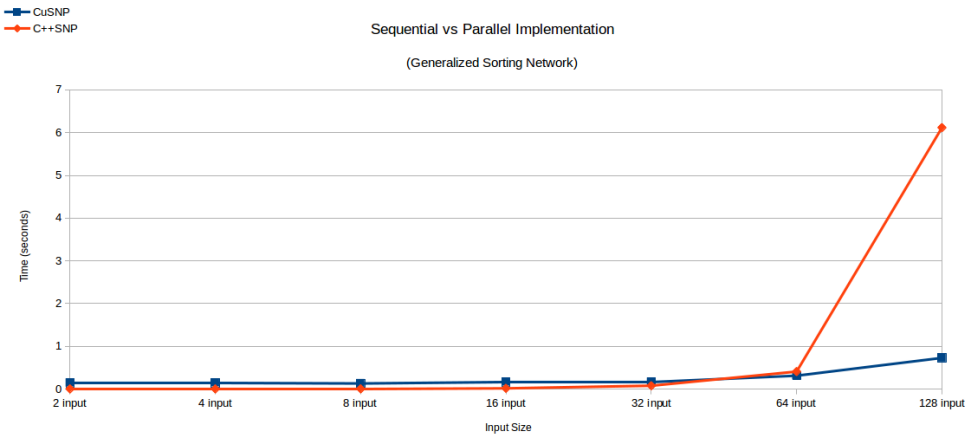


Fig. 2: Runtime Comparison of C++SNP (Sequential) vs CuSNP (Parallel) implementations, simulating SN P systems as generalized sorting networks with input sizes 2 up to 128 (continued in Figure 3). Vertical axis is time, given in seconds.

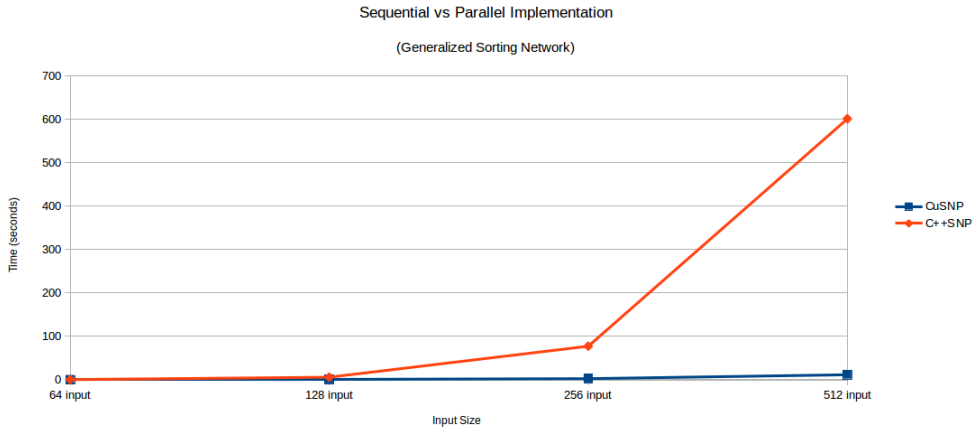


Fig. 3: Runtime Comparison of C++SNP (Sequential) vs CuSNP (Parallel) implementations, simulating SN P systems as generalized sorting networks with input sizes 64 up to 512 (continued from Figure 2. Vertical axis is time, given in seconds.

In Figure 4 we see a chart indicating the speedup, in this case the ratio between the running time of C++SNP over CuSNP, for each input size of the sorting network. Following the GPU computing workflow mentioned above, the larger inputs benefit from being parallelized using CUDA GPUs. It must be noted that for input size 64 and lower, the sequential simulator C++SNP runs faster than CuSNP (see Figure 2, due to the overhead mentioned above. However, for input size 128 and larger, CuSNP overtakes C++SNP in terms of running time as more parallelism is introduced given larger input. This overtaking is also seen in the speedup in Figure 4, where the speedup for input size 64 and lower is less than 1, while speedup for input size 128 and above is greater than 1. In particular, the maximum speedup we obtained is approximately 51 for input size of 512.

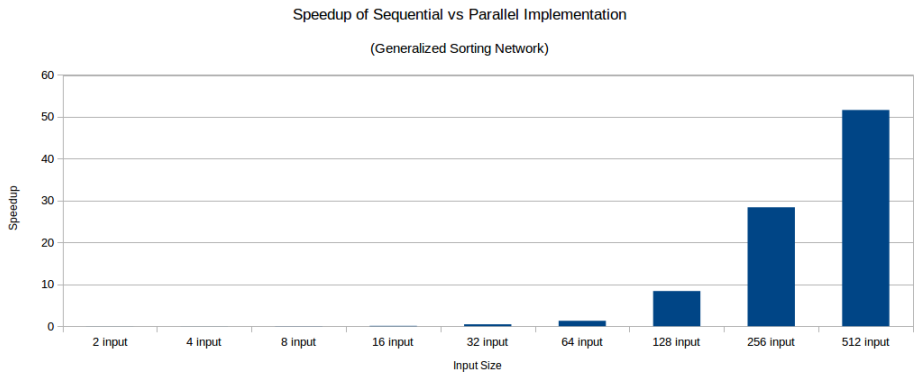


Fig. 4: Runtime speedup of C++SNP (sequential) vs CuSNP (parallel) implementations.

7 Final remarks

In this work we presented our ongoing efforts to simulate SN P systems in NVIDIA CUDA GPUs. In particular, the software available in [2] includes parallel and sequential simulators which simulate SN P systems with delays. We modified the matrix representation in [18] in order to simulate SN P systems with delays. Our experiments were performed using systems implementing generalized sorting networks form [8], and we achieved speedup values of up to 51 times for a 512-size input, i.e. for a 512-size sorting network, CuSNP (parallel simulator) is 51 times faster than C++SNP (sequential simulator). This speedup was obtained, largely in part due to improved memory access pattern between the host (CPU) and the device (GPU).

Much work remains to be done, and we are currently extending the regular expressions available for both simulators to include more general regular expressions (using implementations of finite automata). Also, we are currently optimizing the data types and structures of CuSNP, among other improvements. In a succeeding work, we will also provide detailed profiles of kernel functions (using tools provided by NVIDIA) in CuSNP in order to identify further possibilities for optimizations of simulator performance.

References

1. P systems web page. `http://ppage.psystems.eu/`.
2. CUDA SNP simulators version 06.05.16 (CuSNP v06.05.16). `http://aclab.dcs.upd.edu.ph/productions/software/cusnp_v060516`, 2016.
3. NVIDIA CUDA C programming guide. `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`, 2016.
4. F. G. C. Cabarle, H. N. Adorna, M. A. Martínez-del Amor, and M. J. Pérez-Jiménez. Improving GPU Simulations of Spiking Neural P Systems. *Romanian Journal of Information Science and Technology*, 15(1):5–20, 2012.
5. F. G. C. Cabarle, H. N. Adorna, M. J. Pérez-Jiménez, and T. Song. Spiking neural P systems with structural plasticity. *Neural Computing and Applications*, 26(8):1905–1917, 2015.
6. J. P. Carandang and J. M. Villaflores. CuSNP: Improvements on GPU Implementation of SNP Systems in NVIDIA CUDA. P*roc. 16th Philippine Computing Science Congress (PCSC2016), Puerto Princesa, Palawan, Philippines*, pages 77–84, 2016.
7. M. Ionescu, G. Păun, and T. Yokomori. Spiking Neural P Systems. *Fundamenta Informaticae*, 71(2,3):279–308, Feb. 2006.
8. M. Ionescu and D. Sburlan. Some Applications Of Spiking Neural P Systems. *Computing and Informatics*, 27(3):515–258, 2008.
9. A. Leporati, C. Zandron, C. Ferretti, and G. Mauri. Solving Numerical NP-Complete Problems with Spiking Neural P Systems. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 4860 of *LNCS*, pages 336–352. 2007.
10. L. F. Macías-Ramos, M. A. Martínez-del Amor, and M. J. Pérez-Jiménez. Simulating FRSN P systems with real numbers in P-Lingua on sequential and CUDA platforms. *Lecture Notes in Computer Science*, 9504:227–241, 12/2015 2015.
11. L. F. Macías-Ramos, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia-Cabrera, M. J. Pérez-Jiménez, and A. Riscos-Núñez. A P-Lingua Based Simulator for Spiking Neural P Systems. In M. Gheorghe, G. Păun, G. Rozenberg, A. Salomaa, and S. Verlan, editors, *Membrane Computing*, volume 7184 of *Lecture Notes in Computer Science*, pages 257–281. Springer Berlin Heidelberg, 2012.
12. M. A. Martínez-del Amor, M. García-Quismondo, L. F. Macías-Ramos, L. Valencia-Cabrera, A. Riscos-Núñez, and M. J. Pérez-Jiménez. Simulating P Systems on GPU Devices: A Survey. *Fundam. Inf.*, 136(3):269–284, July 2015.
13. M. A. Martínez-Del-Amor, L. F. Macías-Ramos, and M. J. Pérez-Jiménez. Parallel simulation of PDP systems: Updates and roadmaps. *13th Brainstorming Week on Membrane Computing, BWMC15, Seville, Spain*, pages 227–244, 2015.

14. L. Pan, G. Păun, and M. J. Pérez-Jiménez. Spiking neural P systems with neuron division and budding. *Science China Information Sciences*, 54(8):1596–1607, 2011.
15. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108 – 143, 2000.
16. G. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford Univeristy Press, 2010.
17. T. Song, L. Pan, and G. Păun. Spiking neural P systems with rules on synapses. *Theoretical Computer Science*, 529:82–95, 2014.
18. X. Zeng, H. Adorna, M. Á. Martínez-del Amor, L. Pan, and M. J. Pérez-Jiménez. *Membrane Computing: 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers*, chapter Matrix Representation of Spiking Neural P Systems, pages 377–391. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

Generalized P Colonies with passive environment

Lucie Ciencialová, Luděk Cienciala, and Petr Sosík

Research Institute of the IT4Innovations Centre of Excellence,
Faculty of Philosophy and Science, Silesian University in Opava, Czech Republic
{lucie.ciencialova,ludek.cienciala,petr.sosik}@fpf.slu.cz

Summary. We study two variants of P colonies with initial content of P colony and so called passive environment: P colonies with two objects inside each agent that can only consume or generate objects, and P colonies with one object inside each agent using rewriting and communication rules. We show that the first kind of P colonies with one consumer agent and one sender agent can generate all sets of natural numbers computed by register machines, and hence they are computationally universal in the Turing sense. Similarly, also the second kind of systems with three agents with rewriting/consuming rules is computationally complete. The paper improves previously published universality results concerning generalized P colonies, and it also extends our knowledge about very simple multi-agent systems capable of universal computation.

Key words: P colony, computational completeness, register machine

1 Introduction

P colony was introduced in [9] as a very simple variant of membrane systems inspired by so called *colonies* of formal grammars. See [11] for more information about membrane systems and [7] for details on grammar systems theory. There are three basic entities in the P colony model: objects, agents and the environment. A P colony is composed of agents, each containing a collection of objects embedded in a membrane. The objects can be placed in the environment, too. Agents are equipped with programs composed of rules that allow interactions of objects. The number of objects inside each agent is set by definition and it is usually very low – 1, 2 or 3. The environment of P colony serves as a communication channel for agents: an agent is able to affect the behaviour of another agent by sending objects via the environment. There is also a special type of *environmental objects* denoted by e which are present in the environment in an unlimited number of copies.

A specific variant of P colony called *eco-P colony* with two object inside each agent, where the environment can change independently on the agents, was introduced in [1]. The evolution of the environment is controlled by a 0L scheme

applying context free rules in parallel to all possible objects in the environment which are unused by the agents in the current step of computation.

The activity of agents is based on rules that can be rewriting, communication or checking; these three types was introduced in [9]. Furthermore, generating, consuming and transporting rules were introduced in [5].

Rewriting rule $a \to b$ allows an agent to rewrite (evolve) one object a placed inside the agent to object b.

Communication rule $a \leftrightarrow b$ exchanges one object c placed inside the agent for object d from the environment.

Checking rule r_1/r_2, where each of r_1, r_2 is a rewriting or a communication rule, sets a priority between these two rules. The agent try to apply the first rule and if it cannot be performed, the agent executes the second rule.

Generating rule $a \to bc$ creates two objects b, c from one object a.

Consuming rule $ab \to c$ rewrites two objects a, b to one object c.

Transporting rule of the form $(a\ in)$ or $(a\ out)$ is used to transport one object from the environment into the agent, or from the agent to the environment, respectively. The rule is always associated with a consuming/generating rule to keep a constant number of object inside the agent.

The rules are combined into programs in such a way that all object inside the agent are affected by execution of the rules in every step. Consequently, the number of rules in the program is the same as the number of object inside the agent. The programs that contain consuming rules are called consuming programs and the programs with generating rules are called generating programs. The agent that only contains consuming resp. generating programs is called consumer resp. sender.

P colonies with senders and consumers without evolving environment were studied in [5] and the authors proved their computational completeness (in the Turing sense), as well as computational completeness of P colonies with senders and consumers with 0L scheme for the environment. Many papers were devoted to P colonies with rewriting and communication rules without evolving environment, e.g., [4, 6, 8], and there are two book chapters in [2] and [11] describing this topic.

In this paper we focus on P colonies with initial content of P colony with "passive" environment. The paper is structured as follows: The second section is devoted to definitions and notations used in the paper. The third section contains results obtained during studies of P colonies with senders and consumers. In the fourth section we study P colonies with one object inside the agent and rewriting/communication rules. The paper concludes with a summary of presented results.

2 Definitions

Throughout the paper we assume the reader is familiar with basic of formal automata and language theory. We introduce notation used in the paper.

We use $\mathbb{N}\cdot\mathsf{RE}$ to denote the family of recursively enumerable sets of natural numbers and $\mathbb{N}$ to denote the set of natural numbers.

Σ is a notation for the alphabet. Let Σ^* be set of all words over alphabet Σ (including the empty word ε). For the length of the word $w \in \Sigma^*$ we use the notation $|w|$ and the number of occurrences of symbol $a \in \Sigma$ in w is denoted by $|w|_a$.

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 over the set of objects V is denoted by V^*. The cardinality of M, denoted by $card(M)$, is defined by $card(M) = \sum_{a \in V} f(a)$. Any multiset of objects M with the set of objects $V = \{a_i, \dots a_n\}$ 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 M, and ε represents the empty multiset.

The mechanism of evolution of the environment is based on a *0L scheme.* It is a pair (Σ, P), where Σ is the alphabet of 0L scheme and P is the set of context free rules fulfilling the condition $\forall a \in \Sigma\ \exists \alpha \in \Sigma^*$ such that $(a \to \alpha) \in P$. For $w_1, w_2 \in \Sigma^*$ we write $w_1 \Rightarrow w_2$ if $w_1 = a_1 a_1 \dots a_n, w_2 = \alpha_2 \alpha_2 \dots \alpha_n$, for $a_i \to \alpha_i \in P, 1 \leq i \leq n$.

A *register machine* [10] is the construct $M = (m, H, l_0, l_h, P)$ where:
- m is a number of registers, H is a set of instruction labels,
- l_0 is an initial/start label, l_h is the final label,
- P is a finite set of instructions injectively labelled with the elements from the given set H.

The instructions of the register machine are of the following forms:

l_1 : $(ADD(r), l_2, l_3)$ Add 1 to the contents of the register r and proceed to the instruction (labelled with) l_2 or l_3.

l_1 : $(SUB(r), l_2, l_3)$ If the register r is not empty, then subtract 1 from its contents and go to instruction l_2, otherwise proceed to instruction l_3.

l_h : *HALT* Stop the machine. The final label l_h is only assigned to this instruction.

Without loss of generality, one can assume that in each *ADD*-instruction $l_1 : (ADD(r), l_2, l_3)$ and in each conditional *SUB*-instruction $l_1 : (SUB(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct. The register machine M computes a set $N(M)$ of numbers in the following way: we start with all registers empty (hence storing the number zero) with the instruction with label l_0 and we proceed to apply the instructions as indicated by the labels (and made possible by the contents of registers). If we reach the halt instruction, then the number stored at that time in the register 1 is said to be computed by M and hence it is introduced in $N(M)$. (Because of the nondeterminism in choosing the continuation of the computation in the case of *ADD*-instructions, $N(M)$ can be an infinite set.) The family of sets of numbers computed by register machines is denoted by $\mathbb{N}\cdot RM$.

Theorem 1. [10] $\mathbb{N}\cdot RM = \mathbb{N}\cdot\mathsf{RE}$.

2.1 Generalized P colonies

Definition 1. *A P colony with capacity $c \geq 1$ is the structure*

$$\Pi = (\Sigma, e, f, v_E, D_E, B_1, \ldots, B_n), \text{ where}$$

- *Σ is the alphabet of the colony, its elements are called objects,*
- *e is the basic (environmental) object of the colony, $e \in \Sigma$,*
- *f is final object of the colony, $f \in \Sigma$,*
- *v_E is the initial content of the environment, $v_E \in (\Sigma - \{e\})^*$,*
- *D_E is 0L scheme (Σ, P_E), where P_E is the set of context free rules,*
- *B_i, $1 \leq i \leq n$, are the agents, every agent is the structure $B_i = (o_i, P_i)$, where o_i is the multiset over Σ, it defines the initial state (content) of the agent B_i and $|o_i| = c$ and $P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is the finite set of programs of three types:*
  - *(1) generating program with generating rules $a \to bc$ and transporting rules d out - the number of generating rules is the same as the number of transporting rules.*
  - *(2) consuming program with consuming rules $ab \to c$ and transporting rules d in - the number of consuming rules is the same as the number of transporting rules.*
  - *(3) rewriting/communication program can contain three types of rules:*
    - *$\diamond$ $a \to b$, called a rewriting rule,*
    - *$\diamond$ $c \leftrightarrow d$, called a communication rule,*
    - *$\diamond$ r_1/r_2, called a checking rule; each of r_1, r_2 is a rewriting or a communication rules.*

Every agent has only one type of programs. The agent with generating programs is called *sender* and the agent with consuming programs is called *consumer.* The capacity of P colony with senders and consumers must be even number.

The *initial configuration* of a P colony is the $(n+1)$-tuple $(o_1, \ldots, o_n, v_E)$, with symbols $o_1, \ldots, o_n, v_E$ as in Definition 1. In general, the *configuration* of the P colony Π is defined as $(n+1)$-tuple $(w_1, \ldots, w_n, w_E)$, where w_i represents the multiset of objects inside i-th agent, $|w_i| = c$, $1 \leq i \leq n$, and $w_E \in (\Sigma - \{e\})^*$ is the multiset of objects different from e placed in the environment.

At each step of the (parallel) computation every agent tries to find one of its programs to apply. If the number of applicable programs is higher than one, the agent non-deterministically chooses one. At each step of computation, the set of active agents executing a program must be maximal, i.e., no further agent can be added to it.

By applying programs, the P colony passes from one configuration to another configuration. Objects in the environment unaffected by any program in the given step are rewritten by the 0L scheme D_E. A sequence of configurations starting from the initial configuration is called a *computation.* A configuration is *halting* if the P colony has no applicable program. Each halting computation has associated

a *result* – the number of copies of the final object placed in the environment in halting configuration.

$$N(\Pi) = \{|w_E|_f \mid (o_1, \ldots, o_n, v_E) \Rightarrow^* (w_1, \ldots, w_n, w_E)\},$$

where $(o_1, \ldots, o_n, v_E)$ is the initial configuration, $(w_1, \ldots, w_n, w_E)$ is the final configuration, and $\Rightarrow^*$ denotes reflexive and transitive closure of $\Rightarrow$.

Let us denote $NEPCOL(i, j, k, u, v, w)$ the family of the sets computing by P colonies with at most $j \geq 1$ agents with $i \geq 1$ objects inside the agent and with at most $k \geq 1$ programs associated with each agent such that:

- $u = check$ if the P colony uses rewriting/communication rules with checking rules
- $u = no\text{-}check$ if the P colony uses rewriting/communication rules without checking rules
- $u = s/c/sc$ if the P colony contains only sender / only consumer / both sender and consumer agents
- $v = pas$ if the rules of 0L scheme are of type $a \to a$ only,
- $v = act$ if the set of rules of 0L scheme contains at least one rule of another type than $a \to a$,
- $w = ini$ if the environment or agents contain initially objects different from e, otherwise w is omitted,

If a numerical parameter is unbounded, we denote it by a $*$.

In [5] the authors deal with P colonies with senders and consumers with "passive" environment, they show that

$$NEPCOL(2, 3, *, sc, pas) = \mathbb{N}\cdot\mathsf{RE}.$$

In [1] there are results of P colonies with "active" environment:

$$NEPCOL(2, 2, *, c, act, ini) = \mathbb{N}\cdot\mathsf{RE}$$

$$NEPCOL(2, 2, *, sc, pas, ini) \supseteq \mathbb{N}\cdot RM_{pb}.$$

Other results are shown for P colonies with "passive" environment and rewriting/communication rules and with only one object inside the agent in [3]

$$NEPCOL(1, 4, *, check, pas) = \mathbb{N}\cdot\mathsf{RE}$$

and in [5]

$$NEPCOL(1, 6, *, no\text{-}check, pas) = \mathbb{N}\cdot\mathsf{RE}.$$

3 P colonies with senders and consumers

In this section we study computational power of P colonies with two objects inside the agent - consumer or sender. We extend the previous results reported in [1].

Theorem 2. $NEPCOL(2,2,*,sc,pas,ini) = \mathbb{N}\cdot\mathsf{RE}$.

Proof. Consider register machine $M = (m, H, l_0, l_h, P)$. All labels from the set H are objects in P colony. The content of register r is represented by the number of copies of objects a_r placed in the environment.

Let u be a mapping $u : H \to \{a_r \mid 1 \le r \le m\} \cup \{L_i \mid l_i \in H\}$ defined as

$$u(l_i) = \begin{cases} a_r \text{ for } l_i : (ADD(r), l_j, l_k) \\ L_i \text{ for } l_i : (SUB(r), l_j, l_k) \end{cases}$$

We construct the P colony $\Pi = (\Sigma, e, a_1, a_2^w, D_E, B_1, B_2)$ with:

- $\Sigma = \{l_i, L_i, L_i^1, L_i^2, W_i^0, W_i^1, W_i^2, l_i^0, l_i^1 \mid l_i \in H\} \cup \{a_i \mid 1 \le i \le m\} \cup$ $\cup \{e, C, Q\}$,
- $B_1 = (l_0 u(l_0), P_1)$,
- $B_2 = (ee, P_2)$.

At the beginning of computation the agent B_1 contains object l_0 representing the label of the initial instruction of M.

An instruction $l_i = (ADD(r), l_j, l_k)$ is simulated by agent B_1 by using following programs:

B_1 :

$1 : \langle l_i \to l_j u(l_j); a_r\ out\rangle;$
$2 : \langle l_i \to l_k u(l_k); a_r\ out\rangle;$

The computation is done in the following way: agent B_1 (sender) simulates the addition of one to the content of register r (sending one copy of object a_r to the environment), and it generates the object l_j or l_k – the label of instruction which will the simulated register machine M execute next. Simultaneously it "precomputes" objects for the execution of the next instruction.

An instruction $l_i : (SUB(r), l_j, l_k)$ is simulated by the following rules and programs:

B_1 :

$3 : \langle l_i \to W_i^0 e, L_i\ out\rangle;$
$4 : \langle W_i^0 \to W_i^1 L_i^1, e\ out\rangle;$
$5 : \langle W_i^1 \to W_i^2 L_i^2, L_i^1\ out\rangle;$
$6 : \langle W_i^2 \to l_j^0 l_j^0, L_i^2\ out\rangle;$
$7 : \langle W_i^2 \to l_k^0 l_k^0, L_i^2\ out\rangle;$
$8 : \langle l_j^0 \to l_j^1 e, l_j^0\ out\rangle;$
$9 : \langle l_k^0 \to l_k^1 e, l_k^0\ out\rangle;$
$10 : \langle l_j^1 \to l_j^2 e, e\ out\rangle;$
$11 : \langle l_k^1 \to l_k^2 e, e\ out\rangle;$
$12 : \langle l_j^2 \to l_j^3 e, e\ out\rangle;$
$13 : \langle l_k^2 \to l_k^3 e, e\ out\rangle;$
$14 : \langle l_j^3 \to l_j u(l_j), e\ out\rangle;$
$15 : \langle l_k^3 \to l_k u(l_k), e\ out\rangle;$

B_2 :

$A : \langle ee \to e; L_i\ in\rangle;$
$B : \langle L_i e \to L_i; a_r\ in\rangle;$
$C : \langle L_i a_r \to L_i^1; L_i^1\ in\rangle;$
$D : \langle L_i e \to L_i^2; L_i^2\ in\rangle;$
$E : \langle L_i^1 L_i^1 \to c; l_j^0\ in\rangle;$
$F : \langle L_i^1 L_i^1 \to q; l_k^0\ in\rangle;$
$G : \langle L_i^2 L_i^2 \to q; l_j^0\ in\rangle;$
$H : \langle L_i^2 L_i^2 \to c; l_k^0\ in\rangle;$
$I : \langle l_j^0 c \to c; L_i^2\ in\rangle;$
$J : \langle l_k^0 c \to c; L_i^1\ in\rangle;$
$K : \langle L_i^1 c \to e; e\ in\rangle;$
$L : \langle L_i^2 c \to e; e\ in\rangle;$
$M : \langle l_j^0 q \to q; e\ in\rangle;$
$N : \langle l_k^0 q \to q; e\ in\rangle;$
$O : \langle qe \to q; e\ in\rangle;$

If there are objects l_i (the label of *SUB*-instruction) and L_i inside the agent B_1, the agent sends object L_i (using the rule labelled 3) to the environment. This is the message for the agent B_2 to try to consume one copy of object a_r from the environment (try to subtract one from the content of register r.)

If the agent B_2 is successful (using the program labelled B), then the second agent consumes L_i^1. If there is no a_r in the environment, the agent has to wait one step and then it consumes object L_i^2.

The agent B_1 generates object l_j^0 or l_k^0, non-deterministically choosing the instruction to be simulated next (program 6 or 7). If the non-deterministic choice was wrong (the agent generates l_j^0 and register r was empty or the agent generates l_k^0 and the register was nonempty), the agent B_2 would use program labelled O and the computation never halts.

If the register r stores nonzero value:

	B_1	B_2	Env	P_1	P_2
1.	l_iL_i	ee	$a_r^x w$	3	–
2.	$W_i^0 e$	ee	$L_i a_r^x w$	4	A
3.	$W_i^1 L_i^1$	$L_i e$	$a_r^x w$	5	B
4.	$W_i^2 L_i^2$	$L_i a_r$	$L_i^1 a_r^{x-1} w$	6 or 7	C
5.	$l_j^0 l_j^0$	$L_i^1 L_i^1$	$L_i^2 a_r^{x-1} w$	8	–
6.	$l_j^1 e$	$L_i^1 L_i^1$	$l_j^0 L_i^2 a_r^{x-1} w$	10	E
7.	$l_j^2 e$	cl_j^0	$L_i^2 a_r^{x-1} w$	12	I
8.	$l_j^3 e$	cL_i^2	$a_r^{x-1} w$	14	L
9.	$l_j u(l_j)$	ee	$a_r^{x-1} w$	?	–

If the register r stores zero:

	B_1	B_2	Env	P_1	P_2
1.	l_iL_i	ee	w	3	–
2.	$W_i^0 e$	ee	$L_i w$	4	A
3.	$W_i^1 L_i^1$	$L_i e$	w	5	–
4.	$W_i^2 L_i^2$	$L_i e$	$L_i^1 w$	6 or 7	–
5.	$l_j^0 l_j^0$	$L_i e$	$L_i^1 L_i^2 w$	8	D
6.	$l_j^1 e$	$L_i^2 L_i^2$	$l_j^0 L_i^1 w$	10	G
7.	$l_j^2 e$	ql_j^0	$L_i^1 w$	12	M
8.	$l_j^3 e$	qe	$L_i^1 w$	14	O
9.	$l_j u(l_j)$	qe	$L_i^1 w$	?	O

	B_1	B_2	Env	P_1	P_2
1.	l_iL_i	ee	$a_r^x w$	3	–
2.	$W_i^0 e$	ee	$L_i a_r^x w$	4	A
3.	$W_i^1 L_i^1$	$L_i e$	$a_r^x w$	5	B
4.	$W_i^2 L_i^2$	$L_i a_r$	$L_i^1 a_r^{x-1} w$	7 or 6	C
5.	$l_k^0 l_k^0$	$L_i^1 L_i^1$	$L_i^2 a_r^{x-1} w$	9	–
6.	$l_k^1 e$	$L_i^1 L_i^1$	$l_k^0 L_i^2 a_r^{x-1} w$	11	E
7.	$l_k^2 e$	ql_k^0	$L_i^2 a_r^{x-1} w$	13	I
8.	$l_k^3 e$	qe	$a_r^{x-1} w$	15	O
9.	$l_k u(l_k)$	qe	$a_r^{x-1} w$	?	O

	B_1	B_2	Env	P_1	P_2
1.	l_iL_i	ee	w	3	–
2.	$W_i^0 e$	ee	$L_i w$	4	A
3.	$W_i^1 L_i^1$	$L_i e$	w	5	–
4.	$W_i^2 L_i^2$	$L_i e$	$L_i^1 w$	7 or 6	–
5.	$l_k^0 l_k^0$	$L_i e$	$L_i^2 L_i^1 w$	9	D
6.	$l_k^1 e$	$L_i^2 L_i^2$	$l_k^0 L_i^1 w$	11	H
7.	$l_k^2 e$	cl_k^0	$L_i^1 w$	13	J
8.	$l_k^3 e$	cL_i^1	w	15	K
9.	$l_k u(l_k)$	ee	w	?	–

No program is needed in $P_1 \cup P_2$ to simulate the instruction l_h :*HALT*. The P colony Π starts its computation with object l_0 in the environment and it simulates the instruction labelled l_0. By the programs it places and deletes from the environment the objects a_r and it halts its computation only after object l_h appears in the environment. The result of computation is the number of copies of

object a_1 placed in the environment at the end of computation. No other halting computation can be executed in the P colony.

4 P colonies with rewriting/communication rules

In this section we deal with P colonies with passive environment and with one object inside each agent. We prove that such a P colony with three agents can generate every recursively enumerable set of natural numbers.

Theorem 3. $NEPCOL(1, 3, *, no\text{-}check, pas, ini) = \mathbb{N}\cdot\mathsf{RE}$.

Proof. Let us consider register machine $M = (m, H, l_0, l_h, P)$. For all labels from the set H we construct corresponding objects in P colony Π. The content of register r will be represented by the number of copies of objects a_r placed in the environment.

We construct the P colony $\Pi = (\Sigma, e, a_1, d, D_E, B_1, B_2, B_3)$ with:

- $\Sigma = \{l_i, l_i', l_i'', \overline{l_i}, \overline{\overline{l_i}}, \underline{l_i}, l_i^1, l_i^2, l_i^3, l_i^4, M_i, M_i^1, M_i^2, M_i^3, M_i^4, N_i, N_i^1, N_i^2, N_i^3, N_i^4 \mid l_i \in H\} \cup \{a_i \mid 1 \le i \le m\} \cup \{e, d, f, g\}$,
- $B_1 = (l_0, P_1)$,
- $B_2 = (d, P_2)$,
- $B_3 = (e, P_3)$.

The object l_0 corresponds to the label of the first instruction executed by the register machine.

The instruction $l_i : (ADD(r), l_j, l_k)$ will be simulated by the agents B_1 and B_2 by using following programs:

B_1 :

$1 : \langle l_i \to l_i' \rangle$;
$2 : \langle l_i' \leftrightarrow e \rangle$;
$3 : \left\langle e \to \underline{l_i''} \right\rangle$;
$4 : \left\langle \underline{l_i''} \to \underline{\underline{l_i''}} \right\rangle$;
$5 : \left\langle \underline{\underline{l_i''}} \leftrightarrow l_i''' \right\rangle$;
$6 : \left\langle \underline{\underline{l_i''}} \to e \right\rangle$;
$7 : \left\langle \overline{\overline{l_i'''}} \to l_i^{iv} \right\rangle$;
$8 : \langle l_i^{iv} \to l_i^v \rangle$;
$9 : \langle l_i^v \to l_j \rangle$;
$10 : \langle l_i^v \to l_k \rangle$;

B_2 :

$A : \langle d \leftrightarrow l_i' \rangle$;
$B : \langle l_i' \to l_i''' \rangle$;
$C : \langle l_i''' \leftrightarrow e \rangle$;
$D : \langle e \leftrightarrow l_i'' \rangle$;
$E : \langle l_i'' \to a_r \rangle$;
$F : \langle a_r \leftrightarrow d \rangle$;

The simulation of ADD-instruction starts by rewriting the object l_i to l_i' by the first agent. The agent B_2 consumes the object l_i', changes it to l_i''' and sends it to the environment. The agent B_1 rewrites the object e to some l_j'', for $l_j \in H$. If this l_j'' has the same index as l_i''' placed in the environment (i.e., $i = j$), the computation passes to the next phase. If $i \neq j$, the agent B_1 tries to generate another l_j''. When the computation gets over this checking step, agent B_2 generates one copy of object a_r and places it to the environment (adding 1 to the content of register i). Then agent B_1 non-deterministically chooses to generate object l_j or l_k.

The instruction $l_i : (SUB(r), l_j, l_k)$ is simulated by using the following rules and programs:

B_1 :

$11 : \langle l_i \to l'_i \rangle;$	$15 : \langle \underline{l''_i} \to l''_i \rangle;$	$19 : \langle l^{iv}_i \to M_i \rangle;$	$23 : \langle \underline{L_i} \leftrightarrow d \rangle;$
$12 : \langle l'_i \leftrightarrow e \rangle;$	$16 : \langle \underline{\underline{l''_i}} \leftrightarrow l'''_i \rangle;$	$20 : \langle M_i \leftrightarrow d \rangle;$	$24 : \langle d \leftrightarrow L^2_i \rangle;$
$13 : \langle e \to \underline{l''_i} \rangle;$	$17 : \langle l''_i \to e \rangle;$	$21 : \langle d \leftrightarrow L_i \rangle;$	$25 : \langle L^2_i \to l_j \rangle;$
$14 : \langle \underline{l''_i} \to \underline{\underline{l''_i}} \rangle;$	$18 : \langle l'''_i \to l^{iv}_i \rangle;$	$22 : \langle L_i \to \underline{L_i} \rangle;$	$26 : \langle d \leftrightarrow L^3_i \rangle;$
			$27 : \langle L^3_i \to l_k \rangle;$

B_2 :

$G : \langle d \leftrightarrow l'_i \rangle;$	$K : \langle l''_i \to L_i \rangle;$	$N : \langle a_r \to h \rangle;$	$Q : \langle L^2_i \leftrightarrow d \rangle;$
$H : \langle l'_i \to l'''_i \rangle;$	$L : \langle L_i \leftrightarrow a_r \rangle;$	$O : \langle h \leftrightarrow \underline{L_i} \rangle;$	$R : \langle M_i \to N_i \rangle;$
$I : \langle l'''_i \leftrightarrow e \rangle;$	$M : \langle L_i \leftrightarrow M_i \rangle;$	$P : \langle \underline{L_i} \to L^2_i \rangle;$	$S : \langle N_i \leftrightarrow d \rangle;$
$J : \langle e \leftrightarrow l''_i \rangle;$			

B_3 :

$A' : \langle e \leftrightarrow N_i \rangle;$	$C' : \langle L^3_i \leftrightarrow \underline{L_i} \rangle;$	$E' : \langle e \leftrightarrow h \rangle;$	$G' : \langle y \leftrightarrow M_i \rangle;$
$B' : \langle N_i \to L^3_i \rangle;$	$D' : \langle \underline{L_i} \to e \rangle;$	$F' : \langle h \to y \rangle;$	$H' : \langle M_i \to e \rangle;$

The simulation starts by generating the objects l''_i, l'''_i in the same way as in the addition part described above. Then the agent B_2 simulates subtraction (if the subtracted register is nonzero). If there was some a_r in the environment, the agent generates object L^2_i. This object agent B_1 can rewrite to l_j). If the register r was empty, the agent B_2 generates object N_i and this object can be rewritten by agent B_3 to object L^3_i. Finally, agent B_1 can rewrite object L^3_i to l_k.

If the register r stores nonzero value:

	B_1	B_2	B_3	Env	P_1	P_2	P_3
1.	l_i	d	e	$da_r^x w$	11	–	–
2.	l_i'	d	e	$da_r^x w$	12	–	–
3.	e	d	e	$l_i' da_r^x w$	13	G	–
4.	$\underline{l_i''}$	l_i'	e	$dda_r^x w$	14	H	–
5.	$\underline{l_i''}$	l_i'''	e	$dda_r^x w$	15	I	–
6.	$\underline{\underline{l_i''}}$	e	e	$l_i''' dda_r^x w$	16	–	–
7.	l_i'''	e	e	$l_i'' dda_r^x w$	18	J	–
8.	l_i^{iv}	l_i'''	e	$dda_r^x w$	19	K	–
9.	M_i	L_i	e	$dda_r^x w$	20	L	–
10.	d	a_r	e	$M_i L_i da_r^{x-1} w$	21	N	–
11.	L_i	h	e	$M_i dda_r^{x-1} w$	22	–	–
12.	$\underline{L_i}$	h	e	$M_i dda_r^{x-1} w$	23	–	–
13.	d	h	e	$\underline{L_i} M_i da_r^{x-1} w$	–	O	–
14.	d	$\underline{L_i}$	e	$h M_i da_r^{x-1} w$	–	P	E'
15.	d	L_i^2	h	$M_i da_r^{x-1} w$	–	Q	F'
16.	d	d	x	$L_i^2 M_i a_r^{x-1} w$	24	–	G'
17.	L_i^2	d	M_i	$dya_r^{x-1} w$	25	–	H'
18.	l_j	d	e	$dya_r^{x-1} w$	?	–	–

If the register r stores value zero:

	B_1	B_2	B_3	Env	P_1	P_2	P_3
1.	l_i	d	e	dw	11	–	–
2.	l_i'	d	e	dw	12	–	–
3.	e	d	e	$l_i'dw$	13	G	–
4.	l_i''	l_i'	e	ddw	14	H	–
5.	$\overline{l_i''}$	l_i'''	e	ddw	15	I	–
6.	$\overline{\overline{l_i''}}$	e	e	$l_i'''ddw$	16	–	–
7.	l_i'''	e	e	$l_i''ddw$	18	J	–
8.	l_i^{iv}	l_i'''	e	ddw	19	K	–
9.	M_i	L_i	e	ddw	20	–	–
10.	d	L_i	e	M_idw	–	M	–
11.	d	M_i	e	L_iddw	21	R	–
12.	L_i	N_i	e	ddw	22	S	–
13.	$\underline{L_i}$	d	e	N_idw	23	–	A'
14.	d	d	N_i	$\underline{L_i}w$	–	–	B'
15.	d	d	L_i^3	$\underline{L_i}w$	–	–	C'
16.	d	d	$\underline{L_i}$	L_i^3w	26	–	D'
17.	L_i^3	d	e	dw	27	–	–
18.	l_k	d	e	dw	?	–	–

No program is needed in $P_1 \cup P_2 \cup P_3$ to simulate the instruction l_h :$HALT$. When l_h appears, the computation halts since no agent can execute a program. The result is the number of objects a_1 placed in the environment and it corresponds to the result of a successful computation of the register machine.

5 Conclusions

In this paper we presented the results obtained during the research of P colonies with passive environment. We have shown that P colonies with with one consumer and one sender agent can generate all sets of natural numbers computable by register machines.

Analogously, if we place three agents with one object inside each of them and with no-checking rewriting/communication programs into the passive environment, the obtained P colony is again computationally complete in the Turing sense.

Acknowledgments.

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. L. Cienciala and L. Ciencialová. Eco-P colonies. In G. Păun, M. Pérez-Jiménez, and A. Riscos-Núñez, editors, *Pre-Proceedings of the 10th Workshop on Membrane Computing, Curtea de Arges, Romania*, pages 201–209, 2009.
2. L. Cienciala and L. Ciencialová. P colonies and their extensions. In J. Kelemen and A. 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 158–169, Berlin Heidelberg, 2011. Springer-Verlag.
3. L. Cienciala, L. Ciencialová, and A. Kelemenová. On the number of agents in P colonies. In *Membrane Computing*, volume 4860 of *LNCS*, pages 193–208. Springer, 2007.
4. L. Ciencialová, L. Cienciala, E. Csuhaj-Varjú, A. Kelemenová, and V. György. On very simple P colonies. In *Proceeding of The Seventh Brainstorming Week on Membrane Computing*, volume 1, pages 97–108, 2009.
5. L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová, and G. Vaszil. Variants of P colonies with very simple cell structure. *Int. J. of Computers, Communications & Control*, 3(IV):224–233, 2009.
6. R. Freund and M. Oswald. P colonies working in the maximally parallel and in the sequential mode. In *Proceedings - Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2005*, pages 419–426, Sept 2005.
7. J. Kelemen and A. Kelemenová. A grammar-theoretic treatment of multiagent systems. *Cybern. Syst.*, 23(6):621–633, Nov. 1992.
8. J. Kelemen and A. Kelemenová. On P colonies, a biochemically inspired model of computation. *Proc. of the 6th International Symposium of Hungarian Researchers on Computational Intelligence*, pages 40–56, 2005.
9. J. Kelemen, A. Kelemenová, and G. Păun. 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)*, pages 82–86. Boston, Mass, 2004.
10. M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.
11. G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.

Solving the 3-COL Problem by Using Tissue P Systems without Environment and Proteins on Cells

Daniel Díaz-Pernil[1], Hepzibah A. Christinal[2], Miguel A. Gutiérrez-Naranjo[3]

[1]Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics - University of Sevilla, 41012, Spain
sbdani@us.es

[2]Karunya University, Coimbatore, Tamilnadu, India
christyhep@gmail.com

[3]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012, Spain
magutier@us.es

Summary. The 3-COL problem consists on deciding if the regions of a map can be coloured with only three colors bearing in mind that two adjacent regions must be coloured with different colors. It is a **NP** problem and it has been previously used in complexity studies in membrane computing to check the ability of a model for solving problems of such complexity class. Recently, tissue P systems with proteins on cells have been presented and its ability to solve **NP**-problems has been proved, but it remained as an open question to know if such model was still able to solve such problems if the environment was removed. In this paper we provide an affirmative answer to this question by showing a uniform family of tissue P systems without environment and with proteins on cells which solves the 3-COL problem in linear time.

1 Introduction

The **P** versus **NP** problem is one of the most important unsolved problem in computer science and it was chosen as one of the seven Millennium Prize Problems [9]. The precise statement of the problem was introduced in 1971 by Stephen Cook [5], although it was essentially mentioned in a personal communication between K. Gödel and J. von Neumann [8].

Whereas the main question is unsolved (i.e., to decide if **P** and **NP** are or not the same complexity class), many efforts have been oriented in the last years in order to find *frontiers of tractability*, i.e., to identify some features of the com-

putational models such that the corresponding device is able to solve or not **NP** problems depending if it is endowed or not with such feature.

In membrane computing there is an extensive literature devoted to this issue (see [21] and the references therein) and the present paper is a novel contribution in such research line. We consider here a variant of one of the most popular P systems architectures: tissue P systems. Such model was firstly presented in [13, 14] by placing the cells in a general graph instead on a tree-like graph as in the cell-like model. Under the hypothesis **P**≠**NP**, Zandron *et al.* [29] established the limitations of P systems that do not use membrane division concerning the efficient solution of **NP**-complete problems. Under this premise, Gh. Păun *et al.* presented in [24] the model of tissue P systems with cell division, able to solve **NP**-problems. Since then, many other variants have been presented, e.g., [6, 10, 11, 16, 17].

Recently, tissue P systems with protein on cells have been introduced [25]. Previously, tissue P systems with proteins on membranes had been presented [22] and many of their properties have been explored (see, e.g., [23, 26, 27]). Nonetheless, the model of tissue P systems with protein on cells is quite different to the model with proteins on membranes: In the first one, proteins can move with multisets of objects but they cannot change. In the model with proteins on membranes, they can be changed, but they cannot move between membranes.

Tissue P system with proteins on cells is endowed with cell division and its ability for solving **NP** problems has been proved [15, 22], but it is an open question to know if after dropping some of the features, the model is still able to solve **NP** problems. In this paper, we prove that the model of tissue P system with proteins on cells can solve **NP** problems if the environment is removed. The environment in tissue P systems has a singular characteristic which makes it different to any other region: based on a biological inspiration, cells can take from the environment the necessary resources for any computation in a similar way that a cell can take as many oxygen molecules from the atmosphere as it needs. This means that the number of objects in the environment is not important and the designer does not need to take care of it. Avoiding the environment is a strong restriction, since all the resources are inside the cells and nothing is taken from outside. The importance of the environment in other membrane computing models has been previously discussed in the literature (see, e.g. [4, 12, 19, 20]). In this paper we provide a uniform family of P systems with proteins on cells without environment which solves the 3-COL problem in linear time and hence, we prove that such systems are able of solving **NP** problems even the environment is dropped.

The paper is organized as follows: Next we give a formal description of the P system model used in this paper and recall some basics on recognizer P systems. In Section 3 we present the uniform family of P systems which solve the 3-COL problem in linear time and discuss the amount of resources needed. Finally, the paper ends with some conclusions.

2 Formal Framework

Tissue P systems with proteins on cells and cell division were introduced in [25]. In the same paper, the definition of *recognizer* tissue P systems [24] is presented in this framework. We adapt these definitions to the case where the environment is not considered.

Definition 1. *A tissue P system without environment, with protein on cells and cell division of degree $q \geq 1$ is a tuple of the form*

$$\Pi = (\Gamma, P, M_1/p_1, \ldots, M_q/p_q, \mathcal{R}, i_{in}, i_{out}),$$

where:

- *Γ, P are finite non–empty alphabets such that $\Gamma \cap P = \emptyset$; Γ is the working alphabet and P is the set of proteins;*
- *$\mathcal{M}_i$ are finite multisets over Γ, $1 \leq i \leq q$;*
- *p_i are elements from P, $1 \leq i \leq q$;*
- *$\mathcal{R}$ is a finite set of rules of the following types:*
  - *Communication rules: $(i, (p_k, u)/(p_l, v), j)$, for $i, j \in \{1, \ldots, q\}$, $i \neq j$, $p_k, p_l \in P$, $u, v \in \Gamma^*$. The length of a communication rule is the total number of objects and proteins involved in that rule.*
  - *Division rules: $[p_j|a]_i \rightarrow [p_k|b]_i\ [p_l|c]_i$ for $i \in \{1, \ldots, q\}$, $p_j, p_k, p_l \in P$, $a, b, c \in \Gamma$, $i \neq i_{out}$*
- *i_{in}, $i_{out} \in \{1, \ldots, q\}$.*

A tissue P system without environment, with protein on cells and cell division can be viewed as a set of q cells, labelled by $\{1, \ldots, q\}$ such that $M_1, \ldots, M_q$ represent the finite multisets of objects initially placed in the q cells of the system and $p_1, \ldots, p_q$ represent one and only one copy of protein initially placed on the q cells of the system; i_{in} is the cell where the input is placed in the initial configuration; and i_{out} represents a distinguished cell which will encode the output of the system. A configuration of the P system at any instant is described by all multisets of objects over Γ associated with all the cells present in the system and the proteins presented on all cells. The initial configuration is $(M_1/p_1, \ldots, M_q/p_q)$. A communication rule of type $(i, (p_k, u)/(p_l, v), j)$ is applicable to a configuration at an instant if cell i contains the protein p_k and the multiset u of objects, cell j contains the protein p_l and the multiset v of objects (multisets u, v may be empty; the empty multiset will be denoted by the symbol λ). When applying such a rule, under the control of the proteins p_k on cell i and p_l on cell j, both the protein p_k and the multiset u of objects are sent from cell i to cell j, and simultaneously, the protein p_l and the multiset v of objects are sent from cell j to cell i. A division rule $[p_j|a]_i \rightarrow [p_k|b]_i\ [p_l|c]_i$ is applicable to a configuration at an instant if cell i contains the protein p_j and the object a. When applying such a rule, under the influence of protein p_j and the object a in cell i, the cell is divided into two cells with the same label; in the first copy of the cell the protein p_j is replaced by p_k

and the object a is replaced by b, in the second copy of the cell the protein p_j is replaced by p_l and the object a is replaced by c; all the remaining objects in the original cell are replicated and distributed in each of the new cells.

Rules are used in a maximally parallel way: at each step, all cells which can evolve must evolve and a maximal multiset of rules is applied (no further rule can be added being applicable). As usual in the variant of tissue P systems, this way of applying rules has only one restriction: when a cell is divided, the division rule is the only one which is applied to that cell at that step. The new cells resulting from division could participate in the interaction with other cells by means of communication rules at the next step (if they are not divided once again).

2.1 Recognizer Tissue P Systems with Protein on Cells and Cell Division

We recall the main notions related to the theory of recognizer P systems, which can be adapted to this model in a natural way. For a detailed description see, e.g., [18, 21]. A decision problem X is a pair (I_X, θ_X) such that I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total Boolean function over I_X. In general, in a *P system with input and output* of any P system variant we consider a working alphabet Γ, with q membranes labelled by $1, \ldots, q$, and initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_q$ associated with them; Σ, which is an (input) alphabet strictly contained in Γ; the initial multisets are over $\Gamma - \Sigma$; and i_{in}, i_{out} are the labels of two distinguished membranes (input and output). Let Γ be the working alphabet of Π, μ its membrane structure, and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ the initial multisets of Π. Let m be a multiset over Σ. The *initial configuration* of the P system is $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_{in}} \cup m, \ldots, \mathcal{M}_q)$.

A *recognizer P system* is a P system with input and output such that:

- The working alphabet contains two distinguished elements *yes*, *no*.
- All its computations halt.
- If $\mathcal{C}$ is a computation of Π, then either the object *yes* or the object *no* (but not both) must have been released into the output region (denoted with label i_{out}), and only in the last step of the computation. We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object *yes* (respectively, *no*) appears in the output region associated to the corresponding halting configuration of $\mathcal{C}$.

A decision problem X can be solved in a polynomially uniform way by a family $\mathbf{\Pi} = \{\Pi(n)\}_{n \in \mathbb{N}}$ of P systems of type $\mathcal{F}$ if the following holds:

- There is a deterministic Turing machine M such that, for every $n \in \mathbb{N}$, starting M with the unary representation of n on its input tape, it constructs the P system $\Pi(n)$ in polynomial time in n.
- There is a deterministic Turing machine N that started with an instance $I \in I_X$ with size n on its input tape, it computes a multiset w_I (called the *encoding of* I) over the input alphabet of $\Pi(n)$ in polynomial time in n.

- For every instance $I \in I_X$ with size n, starting $\Pi(n)$ with w_I in its input membrane, every computation of $\Pi(n)$ halts and sends out to the environment *yes* if and only if I is a positive instance of X.

According to the standard notation, $\mathbf{TPDC}(k)$ denotes the class of recognizer tissue P systems without environment with protein on cells and communication rules of length at most k and $\mathbf{PMC}_{\widehat{TPDC}(k)}$ the set of all decision problems which can be solved by means of such class. This class is closed under polynomial time reduction and under complement.

3 The 3-COL Problem

A k–coloring ($k \geq 1$) of an undirected graph $\mathcal{G} = (V, E)$ is a function $f : V \to \{1, \ldots, k\}$, where $\{1, \ldots, k\}$ are interpreted as colors. We say that $\mathcal{G}$ is k–colorable if there exists a k–coloring, f, such that $f(u) \neq f(v)$ for every edge $\{u, v\} \in E$ (such a k–coloring f is said to be *valid*).

In particular, when $k = 3$, we have the well-known 3–coloring problem: *given an undirected graph $\mathcal{G}$, decide whether or not $\mathcal{G}$ is 3-colorable*; that is, if there exists a valid 3–coloring of $\mathcal{G}$. For the sake of readability, we will use $\{R, G, B\}$ instead of $\{1, 2, 3\}$ to represent the colors (R, G and B standing for *red*, *green* and *blue*, respectively). This problem is related to the famous Four Color Conjecture (proved by Appel and Haken [2, 3]). The **NP**-completeness of the 3–coloring problem was proved by Stockmeyer [28] (see [7]).

Next, we will prove that the 3–coloring problem can be solved in a linear time by a family of recognizer tissue P systems without environment and with proteins on cells. As usual, we will address the resolution via a brute force algorithm, which consists in the following stages:

- *Generation Stage*: All the possible 3–coloring are generated, each of them placed in a different cell. By using the division rules, an exponential amount of cells can be obtained in linear time. In parallel, the cell containing initially a copy of the description of the graph is also divided generating as many copies of the graph as 3-colorings.
- *Checking Stage*: If a generated 3–coloring has two objects K_i and K_j ($K \in \{R, G, B\}$) and the graph has an edge A_{ij} linking the nodes i and j, this coloring is not valid. Since the number of cells containing a copy of the description of the graph is large enough, the checking for all the colorings can be done in parallel by pairing cells encoding 3-colorings with cells encoding copies of the graph. This stage takes only one step.
- *Output Stage*: It suffices that one of the possible coloring satisfies the conditions in order to have a positive answer. If such coloring exists, a distinguished protein will be send to the appropriate cell. We can control via a counter the number of steps for it. If such protein occurs in the right cell at the right moment, the system sends `yes` to the output cell. If such step is reached and the protein has not been released, an object `no` is sent to the output cell.

Each of the P systems of the uniform family $\mathbf{\Pi} = \{\Pi_n\}_{n\in\mathbb{N}}$ described below depends only on one parameter n which represents the number of nodes of the graph. Each of these Π_n is supplied with the encoding of a concrete instance of a graph with n vertices in order to start the computation. The graph will be encoded by using an *input alphabet* $\Sigma = \{A_{ij} : 1 \leq i < j \leq n\}$, and an object A_{ij} will belong to the *input multiset* if and only if there is an edge in the graph linking the nodes i and j. For the sake of simplicity we drop the subscript in Π_n. Formally, for each $n \in \mathbb{N}$, the tissue P system is defined as

$$\Pi = (\Gamma, P, \Sigma, \mathcal{M}_1/p_1, \mathcal{M}_2/p_2, \mathcal{M}_3/p_3, \mathcal{M}_4/p_4, \mathcal{M}_5/p_5, \mathcal{R}, i_{in}, i_{out}),$$

- $\Gamma = \Sigma \cup \{A_i, R_i, G_i, B_i, U_i, V_i : 1 \leq i \leq n\}$
  $\cup \{a_i : 0 \leq i \leq 2n+1\}$
  $\cup \{b_i : 0 \leq i \leq 2n+2\}$
  $\cup \{T, \texttt{yes}, \texttt{no}\}$
- $\Sigma = \{A_{ij} : 1 \leq i < j \leq n\}$
- $P = \{p_{i,j} : i \in \{1,2\}\, j \in \{1,\ldots,2n+1\}$
  $\cup \{q_{i,j} : i \in \{1,2\}\, j \in \{1,\ldots,2n\}$
  $\cup \{p_0\}$
- $\mathcal{M}_1 = \{A_1, \ldots, A_n\}$ with the initial protein $p_{1,1}$ in cell 1;
- $\mathcal{M}_2 = \{A_1, \ldots, A_n\}$ with the initial protein $p_{2,1}$ in cell 2;
- $\mathcal{M}_3 = \{a_0\}$ with the initial protein p_0 in cell 3;
- $\mathcal{M}_4 = \{b_0, \texttt{yes}, \texttt{no}\}$ with the initial protein p_0 in cell 4;
- $\mathcal{M}_5 = \{a_1, \ldots, a_{2n+1}, b_1, \ldots, b_{2n+2}\}$ with the initial protein p_0 in cell 5;
- $\mathcal{R}$ is the following set of rules:
  1. *Division rules:* For $i \in \{1,2\}$ and $j \in \{1,\ldots,n\}$
     $r_{1,i,j} \equiv [p_{i,j}|A_j]_i \to [q_{i,j}|U_j]_i\ [q_{i,j}|V_j]_i$
     $r_{2,i,j} \equiv [q_{i,j}|U_j]_i \to [p_{i,j+1}|R_j]_i\ [p_{i,j+1}|G_j]_i$
     $r_{3,i,j} \equiv [q_{i,j}|V_j]_i \to [p_{i,j+1}|B_j]_i\ [p_{i,j+1}|T]_i$

  2. *Communication rules:*
     $r_{4,i,j,K} \equiv (1, (p_{1,2n+1}, A_{ij})/(p_{2,2n+1}, K_i\, K_j), 2)$
     for $i, j \in \{1,\ldots,n\}$, $i < j$, $K \in \{R, G, B\}$
     $r_{5,i} \equiv (3, (p_0, a_i)/(p_0, a_{i+1}), 5)$ for $i = \{0,\ldots,2n\}$
     $r_{6,i} \equiv (4, (p_0, b_i)/(p_0, b_{i+1}), 5)$ for $i = \{0,\ldots,2n+1\}$
     $r_7 \equiv (2, (p_{2,2n+1}, \lambda)/(p_0, a_{2n+1}), 3)$
     $r_8 \equiv (4, (p_0, b_{2n+2}\, \texttt{yes})/(p_{2,2n+1}, \lambda), 3)$
     $r_9 \equiv (4, (p_0, b_{2n+2}\, \texttt{no})/(p_0, \lambda), 3)$
- $i_{in} = 1$, is the input cell
- $i_{out} = 3$, is the output cell

3.1 An Overview of the Computation

The system is deterministic and it exploits the parallelism intrinsic to membrane computing systems and the specific feature of tissue P system with proteins on

cells which fix one and only one protein in each membrane. From the initial configuration, four processes start:

1. Cell 1 is divided by the application of rules $r_{1,1,j}$, $r_{1,2,j}$ and $r_{1,3,j}$. The configuration $\mathbb{C}_{2n}$ has 2^n membranes with label 1, each of them containing a copy of the input and a protein $p_{1,2n+1}$.
2. Cell 2 is divided by the application of rules $r_{2,1,j}$, $r_{2,2,j}$ and $r_{2,3,j}$ in parallel with the cell of label 1. The configuration $\mathbb{C}_{2n}$ has 2^n membranes with label 2 all of them with the protein $p_{2,2n+1}$. Some of these membrane contain one or more copies of the object T. Each of the remaining 3^n membranes contain a 3-coloring, i.e., a multiset of objects $C_1C_2 \dots C_n$ with $C \in \{R, G, B\}$.
3. Cell 3 interchanges one object with cell 5 during the $2n$ first steps, so at $\mathbb{C}_{2n}$ it contains the protein p_0 and the object a_{2n}.
4. Analogously, cell 4 interchanges one object with cell 5 during the $2n$ first steps, so at $\mathbb{C}_{2n}$ it contains the protein p_0 and the object b_{2n}.

At the configuration $\mathbb{C}_{2n}$, cells 1 contain the protein $p_{1,2n+1}$ and cells 2 contain the protein $p_{2,2n+1}$. If a cell 2 contain two objects K_iK_j with the same color ($K \in \{R, G, B\}$) and there exists an edge A_{ij} in the input, then the rule $r_{4,i,j,K}$ is applied and the corresponding cells interchange their proteins. Since there are enough cells with label 1, the following holds:

- If a cell 2 represent a valid coloring, then the rule $r_{4,i,j,K}$ is not applied and the cell has the protein $p_{2,2n+1}$ at the configuration $\mathbb{C}_{2n+1}$.
- Otherwise, if the coloring represented in the cell is not valid, then the rule $r_{4,i,j,K}$ is applied and the cell has the protein $p_{1,2n+1}$ at the configuration $\mathbb{C}_{2n+1}$.
- Moreover, at $\mathbb{C}_{2n+1}$, cell 3 has protein p_0 and an object a_{2n+1} and cell 4 has protein p_0 and and object b_{2n+1}

Let us recall that if there exists at least one valid coloring, then the answer to the 3-COL problem must be affirmative. Let us consider that there exist such valid coloring and then, at $\mathbb{C}_{2n+1}$ there exists (at least) one cell 2 with protein $p_{2,2n+1}$. In such case the rule 7 applied and at $\mathbb{C}_{2n+2}$ the cell 3 contains the protein $p_{2,2n+1}$. Otherwise, if none of the cells 2 has the protein $p_{2,2n+1}$, then the rule 7 is not applied and cell 3 has the protein p_0 at $\mathbb{C}_{2n+2}$. In such configuration the object b_{2n+2} has reached cell 4. Finally, depending on the protein p_0 or $p_{2,2n+1}$ in cell 3, rule 8 or rule 9 will be applied sending the right answer to cell 3. No more rules can be applied and $\mathbb{C}_{2n+3}$ is a halting configuration.

3.2 Computational Efficiency

The amount of resources used in the construction of the P system Π_n can be summarized as follows: The working alphabet Γ is $O(n)$ with $10n + 8$ objects; the input alphabet is $O(n^2)$ with $\frac{n^2-n}{2}$ objects; the set of proteins is $O(n)$ with $6n+3$ proteins; and the number of rules is $O(n^2)$ with $\frac{3}{2}n^2 + \frac{17}{2}n + 6$ rules. All the

computation halt after $2n+3$ steps. Finally, the communication rules have length 5 at most. Therefore the main result of this paper holds.

Theorem 1. 3-$COL \in \mathbf{PMC}_{\widehat{TPDC}(5)}$

Corollary 1. $\mathbf{NP} \cup co\text{–}\mathbf{NP} \subseteq \mathbf{PMC}_{\widehat{TPDC}(5)}$

These results hold from the previous construction and the closure under polynomial-time reduction and under complement of the complexity class.

4 Conclusions

Whereas the **P** vs. **NP** is unsolved, the search of new frontiers of tractability allows us to have a deeper knowledge of the problem. In the framework of membrane computing, and in natural computing in general, the use of bio-inspired features in such complexity studies shed a new light on an old problem. In this paper we present a new solution to the 3-COL problem with tissue P systems with proteins on cells and without environment which uses communication rules of length at most 5. By using environment, the solution for the SAT problem proposed in [25] uses communication rules of length at most 4. In [15] the proposed solution for the 3-COL problem also uses communication rules of length at most 4. Although both problems, SAT and 3-COL, are different, it remains open the question if it is possible to find a solution to a **NP** problem in the model of tissue P systems with proteins on cells by removing the environment and using communication rules of length at most 4.

References

1. Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.): Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers, Lecture Notes in Computer Science, vol. 8340. Springer (2014)
2. Appel, K., Haken, W.: Every planar map is 4-colorable - 1: Discharging. Illinois Journal of Mathematics 21, 429–490 (1977)
3. Appel, K., Haken, W.: Every planar map is 4-colorable - 2: Reducibility. Illinois Journal of Mathematics 21, 491–567 (1977)
4. Christinal, H.A., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Tissue-like p systems without environment. In: Martínez del Amor, M.A., Păun, Gh., Pérez Hurtado, I., Riscos-Núñez, A. (eds.) Eighth Brainstorming Week on Membrane Computing. pp. 53–64. Fénix Editora, Sevilla, Spain (2010)
5. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158. STOC '71, ACM, New York, NY, USA (1971)
6. Freund, R., Păun, Gh., Pérez-Jiménez, M.J.: Tissue P systems with channel states. Theoretical Computer Science 330(1), 101–116 (2005)

7. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
8. Hartmanis, J.: Gödel, von Neumann and the P =? NP problem. In: Rozenberg, G., Salomaa, A. (eds.) Current Trends in Theoretical Computer Science - Essays and Tutorials, World Scientific Series in Computer Science, vol. 40, pp. 445–450. World Scientific (1993)
9. Jaffe, A.M.: The millennium grand challenge in mathematics. Notices of the American Mathematical Society 53(6), 652 – 660 (2006)
10. Krishna, S.N., Lakshmanan, K., Rama, R.: Tissue P systems with contextual and rewriting rules. In: Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) WMC-CdeA. Lecture Notes in Computer Science, vol. 2597, pp. 339–351. Springer, Berlin Heidelberg (2002)
11. Lakshmanan, K., Rama, R.: On the power of tissue P systems with insertion and deletion rules. In: Alhazov, A., Martín-Vide, C., Păun, Gh. (eds.) Preproceedings of the Workshop on Membrane Computing. pp. 304–318. Tarragona (July 17-22 2003)
12. Macías-Ramos, L.F., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rius-Font, M., Valencia-Cabrera, L.: The efficiency of tissue P systems with cell separation relies on the environment. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) Membrane Computing - 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7762, pp. 243–256. Springer (2012)
13. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON. Lecture Notes in Computer Science, vol. 2387, pp. 290–299. Springer, Berlin Heidelberg (2002)
14. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
15. Mathu, T., Christinal, H.A., Díaz-Pernil, D.: A uniform family of tissue P systems with protein on cells solving 3-coloring in linear time. In: Calude, C.S., Dinneen, M.J. (eds.) Unconventional Computation and Natural Computation - 14th International Conference, UCNC 2015, Auckland, New Zealand, August 30 - September 3, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9252, pp. 239–249. Springer (2015)
16. Pakash, V.: On the power of tissue P systems working in the maximal-one mode. In: Alhazov, A., Martín-Vide, C., Păun, Gh. (eds.) Preproceedings of the Workshop on Membrane Computing. pp. 356–364. Tarragona (July 17-22 2003)
17. Pan, L., Pérez-Jiménez, M.J.: Computational complexity of tissue-like P systems. Journal of Complexity 26(3), 296–315 (2010)
18. Pérez-Jiménez, M.J.: An approach to computational complexity in membrane computing. In: Mauri, G., Păun, Gh., Pérez-Jiménez, M.J., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 3365, pp. 85–109. Springer (2004)
19. Pérez-Jiménez, M.J., Riscos-Núñez, A., Rius-Font, M., Romero-Campero, F.J.: A polynomial alternative to unbounded environment for tissue P systems with cell division. International Journal of Computer Mathematics 90(4), 760–775 (2013)
20. Pérez-Jiménez, M.J., Riscos-Núñez, A., Rius-Font, M., Valencia-Cabrera, L.: The relevance of the environment on the efficiency of tissue P systems. In: Alhazov et al. [1], pp. 308–321

21. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity - membrane division, membrane creation. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 302 – 336. Oxford University Press, Oxford, England (2010)
22. Păun, A., Popa, B.: P systems with proteins on membranes. Fundamenta Informaticae 72(4), 467–483 (2006)
23. Păun, A., Păun, M., Rodríguez-Patón, A., Sidoroff, M.: P systems with proteins on membranes: a survey. International Journal of Foundations of Computer Science 22(1), 39–53 (2011)
24. Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Tissue P systems with cell division. International Journal of Computers, Communication and Control 3(3), 295–303 (2008)
25. Song, B., Pan, L., Pérez-Jiménez, M.J.: Tissue P systems with protein on cells. Fundamenta Informaticae 144(1), 77–107 (2016)
26. Sosík, P.: Active membranes, proteins on membranes, tissue P systems: Complexity-related issues and challenges. In: Alhazov et al. [1], pp. 40–55
27. Sosík, P., Păun, A., Rodríguez-Patón, A.: P systems with proteins on membranes characterize PSPACE. Theoretical Computer Science 488, 78–95 (2013)
28. Stockmeyer, L.: Planar 3-colorability is NP-complete. SIGACT News 5(3), 19–25 (1973)
29. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C., Dinneen, M.J. (eds.) UMC. pp. 289–301. Springer (2000)

Semantics of Deductive Databases in a Membrane Computing Connectionist Model

Daniel Díaz-Pernil[1], Miguel A. Gutiérrez-Naranjo[2]

[1]Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics - University of Sevilla, 41012, Spain
sbdani@us.es

[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012, Spain
magutier@us.es

Summary. The integration of symbolic reasoning systems based on logic and connectionist systems based on the functioning of living neurons is a vivid research area in computer science. In the literature, one can found many efforts where different reasoning systems based on different logics are linked to classic artificial neural networks. In this paper, we study the relation between the semantics of reasoning systems based on propositional logic and the connectionist model in the framework of membrane computing, namely, spiking neural P systems. We prove that the fixed point semantics of deductive databases and the immediate consequence operator can be implemented in the spiking neural P systems model.

1 Introduction

Two of the most well-known paradigms for implementing automated reasoning in machines are, on the one hand, the family of connectionist systems, inspired in the network of biological neurons in a human brain and, on the other hand, logic-based systems, able to represent and reason with well-structured symbolic data. The integration of both paradigms is a vivid area in artificial intelligence (see, e.g., [2, 3, 8]).

In the framework of membrane computing, several studies have been presented where P systems are used for representing logic-based information and performing reasoning by the application of bio-inspired rules (see [7, 11]). These papers study approaches based on cell-like models, as P systems with active membranes, and deal with procedural aspects of the computation. The approach in this paper is different in both senses.

On the one hand, the connectionist model of P systems is considered, i.e, the model of P system inspired by the neurophysiological behavior of neurons sending

electrical impulses along axons to other neurons (the so-called spiking neural P systems, SN P systems for short). On the second hand, we consider the semantics of propositional deductive databases in order to show how SN P systems can deal with logic-based representing and reasoning systems.

One of the key points of the integrate-and-fire formal spiking neuron models [6] (and, in particular, of the SN P systems) is the use of the *spikes* as a support of the information. Such spikes are short electrical pulses (also called action potentials) between neurons and can be observed by placing a fine electrode close to the soma or axon of a neuron. From the theoretical side, it is crucial to consider that all the biological spikes of an alive biological neuron look alike. This means that we can consider a bio-inspired binary code which can be used to formalize logic-based semantics: the emission of one spike will be interpreted as *true* and the absence of spikes will be interpreted as *false*. As we will show below, SN P systems suffice for dealing with the semantics of propositional logic systems.

The main result of this paper is to prove that given a reasoning system based on propositional logic it is possible to find an SN P system with the same declarative semantics. A declarative semantics for a rule-based propositional system is usually given by selecting models which satisfy certain properties. This choice is often described by an operator mapping interpretations to interpretations. In this paper we consider the so-called *immediate consequence operator* due to van Emden and Kowalski [5]. It is well-know that such operator is order continuous and its least fix point coincides with the least model of KB. We adapt the definition of the immediate consequence operator to a restricted form of SN P system and we prove that a least fix point, and hence a least model, is obtained for the given reasoning system.

The paper is organized as follows: firstly, we recall some aspects about SN P systems and the semantics of deductive databases. In Section 3 we prove that standard SN P systems can deal with the semantics of deductive databases. Finally, some conclusions are provided in the last section.

2 Preliminaries

We assume the reader to be familiar with basic elements about membrane computing and the semantics of rule-based systems. Next, we briefly recall some definitions. We refer to [13] for a comprehensive presentation of the former and [1, 4, 12] for the latter.

2.1 Spiking Neural P Systems

SN P systems were introduced in [10] with the aim of incorporating in membrane computing ideas specific to spike-based neuron models. It is a class of distributed and parallel computing devices, inspired by the neurophysiological behavior of neurons sending electrical impulses (*spikes*) along axons to other neurons.

In SN P systems the cells (also called *neurons*) are placed in the nodes of a directed graph, called the *synapse graph*. The contents of each neuron consist of a number of copies of a single object type, called the *spike*. Every cell may also contain a number of *firing* and *forgetting* rules. Firing rules allow a neuron to send information to other neurons in the form of *spikes* which are accumulated at the target cell. The applicability of each rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use one of its rules, then one of such rules must be used. If two or more rules could be applied, then only one of them is non-deterministically chosen. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other. As usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, and hence the functioning of the system is synchronized.

Formally, an SN P system of the degree $m \geq 1$ is a construct[1]

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn)$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) *firing* rules $E/a^p \to a$, where E is a regular expression over a and $p \geq 1$ is an integer number;
      (2) *forgetting* rules $a^s \to \lambda$, with s an integer number such that $s \geq 1$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons.

The rules of type (1) are firing rules, and they are applied as follows. If the neuron σ_i contains k spikes, $k \geq p$, and a^k belongs to the language $L(E)$ associated to the regular expression E, then the rule $E/a^p \to a$ can be applied. The application of this rule means removing p spikes (thus only $k - p$ remain in σ_i), the neuron is fired, and it produces one spike which is sent immediately to all neurons σ_j such that $(i, j) \in syn$. The rules of type (2) are forgetting rules and they are applied as follows: if the neuron σ_i contains exactly s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i. If a rule $E/a^p \to a$ of type (1) has $E = a^p$, then we will write it in the simplified form $a^p \to a$. In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. Since two firing rules, $E_1/a^{p_1} \to a$ and $E_2/a^{p_2} \to a$ can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and in that case only one of them is non-deterministically chosen.

The j-th configuration of the system is described by a vector $\mathbb{C}_j = (t_1, \ldots, t_m)$ where t_k represents the number of spikes at the neuron σ_k in such configuration.

[1] We provide a definition without delays, input or output neurons because these features are not used in this paper.

The initial configuration is $\mathbb{C}_0 = (n_1, n_2, \ldots, n_m)$. Using the rules as described above, one can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a computation. A computation halts if it reaches a configuration where no rule can be used. Generally, a computation may not halt. If it halts, the last configuration is called a *halting* configuration.

2.2 Semantics of Rule-based Deductive Databases

Given two pieces of knowledge V and W, expressed in some language, the rule $V \to W$ is usually considered as a causal relation between V and W. In this paper, we only consider propositional logic for representing the knowledge. Given a set of propositional variables $\{p_1, \ldots, p_n\}$, a rule is a formula $B_1 \wedge \cdots \wedge B_m \to A$ where $m \geq 0$, $A, B_1, \ldots B_m$ are variables. The variable A is called the *head* of the rule and the conjunction of variables $B_1 \wedge \cdots \wedge B_m$ is the *body* of the rule. If $m = 0$, it is said that the body of the rule is empty. A finite set of rules KB is a deductive database. An interpretation I is a mapping from the set of variables $\{p_1, \ldots, p_n\}$ to the set $\{0, 1\}$. As usual, we will represent an interpretation I as a vector $(i_1, \ldots, i_n)$ with $I(p_k) = i_k \in \{0, 1\}$ for $k \in \{1, \ldots, n\}$. The set of all the possible interpretations for a set of n variables will be denoted by 2^n. Given two interpretations I_1 and I_2, $I_1 \subseteq I_2$ if for all $k \in \{1, \ldots, n\}$, $I_1(p_k) = 1$ implies $I_2(p_k) = 1$. We will denote by $I_\emptyset$ the interpretation that maps to 0 every variable, $I_\emptyset = (0, \ldots, 0)$. The interpretation I is extended in the usual way, $I(B_1 \wedge \cdots \wedge B_m) = \min\{I(B_1), \ldots, I(B_m)\}$ and for a rule[2]

$$I(B_1 \wedge \cdots \wedge B_m \to A) = \begin{cases} 0 \text{ if } I(B_1 \wedge \cdots \wedge B_m) = 1 \text{ and } I(A) = 0 \\ 1 \text{ otherwise} \end{cases}$$

An interpretation I is a model for a deductive database KB if $I(R) = 1$ for all $R \in KB$. Next, we recall the propositional version of the immediate consequence operator which was introduced by van Emden and Kowalski [5].

Definition 1. *Let KB be a deductive database on a set of variables $\{p_1, \ldots, p_n\}$. The immediate consequence operator of KB is the mapping $T_{KB} : 2^n \to 2^n$ such that for all interpretation I, $T_{KB}(I)$ is an interpretation*

$$T_{KB}(I) : \{p_1, \ldots, p_1\} \to \{0, 1\}$$

such that, for $k \in \{1, \ldots, n\}$, $T_{KB}(I)(p_k) = 1$ if there exists a rule $B_1 \wedge \cdots \wedge B_m \to p_k$ in KB such that $I(B_1 \wedge \cdots \wedge B_m) = 1$; otherwise, $T_{KB}(I)(p_k) = 0$.

The importance of the immediate consequence operator is shown in the following proposition (see [9]).

[2] Let us remark that, from the definition, if $m = 0$, $I(B_1 \wedge \cdots \wedge B_m) = 1$ and, hence, for a rule with an empty body, we have $I(\to A) = 1$ if and only if $I(A) = 1$.

Theorem 1. *An interpretation I is a model of KB if and only if $T_{KB}(I) \subseteq I$.*

Since the image of an interpretation is an interpretation, the immediate consequence operator can be iteratively applied.

Definition 2. *Let KB be a deductive database and T_{KB} its immediate consequence operator. The mapping $T_{KB} \uparrow: \mathbb{N} \to 2^n$ is defined as follows: $T_{KB} \uparrow 0 = I_\emptyset$ and $T_{KB} \uparrow n = T_{KB} \uparrow (T_{KB} \uparrow (n-1))$ if $n > 0$. In the limit, it is also considered*

$$T_{KB} \uparrow \omega = \bigcup_{k \geq 0} T_{KB} \uparrow k$$

The next theorem is a well-known result which relates the immediate consequence operator with the least model of a deductive database (see [12]).

Theorem 2. *Let KB be a deductive database. The following results hold*

- *$T_{KB} \uparrow \omega$ is a model of KB*
- *If I is a model of KB, then $T_{KB} \uparrow \omega \subseteq I$*

Example 1. Let us consider the following knowledge base KB on the set of variables $\Gamma = \{p_1, p_2, p_3, p_4\}$

$$R_1 \equiv \to p_1$$
$$R_2 \equiv p_1 \to p_2$$
$$R_3 \equiv p_1 \wedge p_2 \to p_3$$
$$R_4 \equiv p_3 \to p_4$$
$$R_5 \equiv p_2 \to p_4$$

and let us consider the interpretation $I : \Gamma \to \{0, 1\}$ such that $I(p_1) = 1$, $I(p_2) = 0$, $I(p_3) = 0$ and $I(p_4) = 0$. Such interpretation can be represented as $I = (1, 0, 0, 0)$. The truth assignment of this interpretation to the rules is $I(R_1) = 1$, $I(R_2) = 0$, $I(R_3) = 1$, $I(R_4) = 1$, $I(R_5) = 1$. Since $I(R_2) = 0$, the interpretation I is not a model for KB. The application of the immediate consequence operator produces $T_{KB}(I) = (1, 1, 0, 0)$. We observe that $T_{KB}(I) \not\subseteq I$ and hence, by Th. 1, we can also conclude that I is not a model for KB. Finally, if we consider $I_\emptyset = (0, 0, 0, 0)$, the following interpretations are obtained by the iterative application of the immediate consequence operator

$T_{KB} \uparrow 0 = I_\emptyset = (0, 0, 0, 0)$
$T_{KB} \uparrow 1 = T_{KB}(T_{KB} \uparrow 0) = (1, 0, 0, 0)$
$T_{KB} \uparrow 2 = T_{KB}(T_{KB} \uparrow 1) = (1, 1, 0, 0)$
$T_{KB} \uparrow 3 = T_{KB}(T_{KB} \uparrow 2) = (1, 1, 1, 1)$

In this case $T_{KB} \uparrow 3$ is a fix point for the immediate consequence operator and a model for the deductive database KB.

3 Semantics of Deductive Databases with SN P Systems

The semantics of deductive databases deals with interpretations, i.e., with mappings from the set of variables into the set $\{0,1\}$ (which stand for *false and true*) and try to characterize which of these interpretations make true a whole deductive database which, from the practical side, may contain hundreds of variables and thousand of rules. The immediate consequence operator provides a tool for dealing with this problem and provides a way to characterize such models. In this section we will explore how this problem can be studied in the framework of SN P systems and prove that the immediate consequence operator can be implemented in this model and therefore, membrane computing provides a new theoretical framework for dealing with the semantics of deductive databases.

Our main result claims that SN P systems can compute the immediate consequence operator and hence, the least model of a deductive database.

Theorem 3. *Given a deductive database KB and an interpretation I, a SN P system can be constructed such that*

(a) It computes the immediate consequence operator $T_{KB}(I)$.
(b) It computes the least model for KB in a finite number of steps.

Proof. Let us consider a knowledge database KB, let $\{p_1,\ldots,p_n\}$ be the propositional variables and $\{r_1,\ldots,r_k\}$ be the rules of KB. Given a variable p_i, we will denote by h_i the number of rules which have p_i in the head and given a rule r_j, we will denote by b_j the number of variables in its body. The SN P systems of degree $2n+k+3$

$$\Pi_{KB} = (O, \sigma_1, \sigma_2, \ldots, \sigma_{2n+k+2}, syn)$$

can be constructed as follows:

- $O = \{a\}$;
- $\sigma_j = (0, \{a \to \lambda\})$ for $j \in \{1,\ldots n\}$
- $\sigma_{n+j} = (i_j, R_j)$, $j \in \{1,\ldots n\}$, where $i_j = I(p_j)$ and R_j is the set of h_j rules $R_j = \{a^k \to a \,|\, k \in \{1,\ldots,h_j\}\}$
- $\sigma_{2n+j} = (0, R_j)$, $j \in \{1,\ldots k\}$, where R_j is one of the following set of rules
  - $R_j = \{a^{b_j} \to a\} \cup \{a^l \to \lambda \,|\, l \in \{1,\ldots,b_j - 1\}\}$ if $b_j > 0$
  - $R_j = \{a \to a\}$ if $b_j = 0$.

For a better understanding, the neurons σ_{2n+k+1} and σ_{2n+k+2} will be denoted by σ_G and σ_T.

- $\sigma_G = (0, \{a \to a\})$
- $\sigma_T = (1, \{a \to a\})$
- $syn = \{(n+i, i) \,|\, i \in \{1,\ldots,n\}\}$
  $\cup \left\{ (n+i, 2n+j) \,|\, i \in \{1,\ldots,n\}, j \in \{1,\ldots,k\} \text{ and } p_i \text{ is a variable in the body of } r_j \right\}$
  $\cup \left\{ (2n+j, n+i) \,|\, i \in \{1,\ldots,n\}, j \in \{1,\ldots,k\} \text{ and } p_i \text{ is the variable in the head of } r_j \right\}$

$$\cup \{(G,T),(T,G)\}$$
$$\cup \left\{ \begin{array}{l} (T, 2n+j) \mid j \in \{1,\ldots,k\} \\ \qquad\qquad \text{and } r_j \text{ is a rule with empty body} \end{array} \right\}$$

Before going on with the proof, let us note that the construction of this SN P system is illustrated in the Example 2. The next remarks will be useful:

Remark 1. *For all $t \geq 0$, in the $2t$-th configuration $\mathbb{C}_{2t}$ the neuron σ_T contains exactly one spike and the neuron σ_G does not contain spikes.*
Proof. In the initial configuration $\mathbb{C}_0$, σ_T contains 1 spike and σ_G does not contain spikes. By induction, let us suppose that in the $\mathbb{C}_{2t}$ the neuron σ_T contains exactly one spike and σ_G does not contain spikes. Since the unique incoming synapse in σ_T comes from σ_G and the unique incoming synapse in σ_G comes from σ_T and in both neurons occurs the rule $a \to a$, then in $\mathbb{C}_{2t+1}$ the neuron σ_G contains exactly spike and σ_T does not contain spikes and finally, in $\mathbb{C}_{2t+2}$ the neuron σ_T contains exactly spike and σ_G does not contain spikes.

Remark 2. *For all $t \geq 0$ the following results hold:*

- *For all $p \in \{1,\ldots,k\}$ the neuron σ_{2n+p} does not contain spikes in the configuration $\mathbb{C}_{2t}$*
- *For all $q \in \{1,\ldots,n\}$, the neuron σ_{n+q} does not contain spikes in the configuration $\mathbb{C}_{2t+1}$*

Proof. In the initial configuration $\mathbb{C}_0$, for all $p \in \{1,\ldots,k\}$, the neuron σ_{2n+p} does not contain spikes and each neuron σ_{n+q} contain, at most, one spike. Such spike is consumed by the application of the rule $a \to a$ and, since all the neurons with synapse to σ_{n+q} do not contain spikes at $\mathbb{C}_0$, we conclude that at the configuration $\mathbb{C}_1$, the neurons σ_{n+q} do not contain spikes.

By induction, let us suppose that in $\mathbb{C}_{2t}$, for all $p \in \{1,\ldots,k\}$, the neuron σ_{2n+p} does not contain spikes and for all $q \in \{1,\ldots,n\}$, the neuron σ_{n+q} does not contain spikes in the configuration $\mathbb{C}_{2t+1}$. According to the construction, the number of incoming synapses in each neuron σ_{2n+j} is b_j if $b_j > 1$ and 1 if $b_j = 0$. Such synapses come from neurons that send (at most) one spike in each computational step, so in $\mathbb{C}_{2t+1}$, the number of spikes in the neuron σ_{2n+j} is, at most, b_j if $b_j > 1$ and 1 if $b_j = 0$. All these spikes are consumed by the corresponding rules. Moreover, at $\mathbb{C}_{2t+1}$, all the neurons with outgoing synapses to σ_{2n+p} do not contain spikes, so we conclude that at $\mathbb{C}_{2t+2}$, for all $j \in \{1,\ldots,k\}$, the neuron σ_{2n+j} does not contain spikes. We focus now on the neurons σ_{n+q} with $q \in \{1,\ldots,n\}$. By induction, we assume that they do not contain spikes in the configuration $\mathbb{C}_{2t+1}$. Each neuron σ_{n+q} can receive at most h_q, since there are h_q incoming synapses and the corresponding neuron sends, at most, one spike. Hence, at $\mathbb{C}_{2t+2}$, σ_{n+q} has, at most, h_q spikes. All of them are consumed by the corresponding rule and, since all the neurons which can send spikes to σ_{n+q} do not contain spikes at $\mathbb{C}_{2t+2}$, we conclude that, for all $q \in \{1,\ldots,n\}$, the neuron σ_{n+q} does not contain spikes in the configuration $\mathbb{C}_{2t+3}$.

Remark 3. For all $q \in \{1, \ldots, n\}$, the neuron σ_q does not contain spikes in the configuration $\mathbb{C}_{2t}$.

Proof. The result holds in the initial configuration. For $\mathbb{C}_{2t}$ with $t > 0$ it suffices to check that, as claimed in Remark 2, for all $q \in \{1, \ldots, n\}$, the neuron σ_{n+q} does not contain spikes in the configuration $\mathbb{C}_{2t+1}$ and each σ_q receives at most one spike in each computation step from the corresponding σ_{n+q}. Therefore, in each configuration $\mathbb{C}_{2t+1}$, each neuron σ_q contains, at most, one spike. Since such spike is consumed by the rule $a \to \lambda$ and no new spike arrives, then the neuron σ_q does not contain spikes in the configuration $\mathbb{C}_{2t}$.

Before going on with the proof, it is necessary to formalize what means that the SN P system computes the immediate consequence operator T_{KB}. Given a deductive database KB on a set of variables $\{p_1, \ldots, p_n\}$, an interpretation on KB can be represented as a vector $I = (i_1, \ldots, i_n)$ with $i_j \in \{0, 1\}$ for $j \in \{1, \ldots, n\}$. Let us consider that such values $i_j \in \{0, 1\}$ represent the number of spikes placed in the corresponding neuron σ_{n+j} at the initial[3] configuration $\mathbb{C}_0$. We will consider that the computed output for such interpretation is encoded in the number of spikes in the neurons $\sigma_1, \ldots, \sigma_n$ in the configuration $\mathbb{C}_3$.

The main results of the theorem can be obtained from the following technical remark.

Remark 4. *Let $I = (i_1, \ldots, i_n)$ an interpretation for KB and let $S = (s_1, \ldots, s_n)$ be a vector with the following properties. For all $j \in \{1, \ldots, n\}$*

- *If $i_j = 0$, then $s_j = 0$.*
- *If $i_j \neq 0$, then $s_j \in \{1, \ldots, h_j\}$*

Let us suppose that at the configuration $\mathbb{C}_{2t}$ the neuron σ_{n+j} contains exactly s_j spikes. Then, the interpretation obtained by applying the immediate consequence operator T_{KB} to the interpretation I, $T_{KB}(I)$ is $(q_1, \ldots, q_n)$ where q_j, $j \in \{1, \ldots, n\}$, is the number of spikes of the neuron σ_j in the configuration $\mathbb{C}_{2t+3}$.

Proof. Firstly, let us consider $k \in \{1, \ldots, n\}$ and $T_{KB}(I)(p_k) = 1$. Let us prove that at the configuration $\mathbb{C}_{2t+3}$ there is exactly one spike in the neuron σ_k.

If $T_{KB}(I)(p_k) = 1$, then there exists at least one rule $r_l \equiv B_{d_1} \wedge \cdots \wedge B_{d_l} \to p_k$ in KB such that $I(B_{d_1} \wedge \cdots \wedge B_{d_l}) = 1$.

Case 1: Let us consider that there is only one such rule r_l and the body of r_l is empty. By construction, the neuron σ_{2n+l} has only one incoming synapse from neuron σ_T; the neuron σ_{n+j} contains exactly s_j spikes, $j \in \{1, \ldots, n\}$ and $s_j \in \{1, \ldots, h_j\}$; and according to the previous remarks:

- In $\mathbb{C}_{2t}$ the neuron σ_T contains exactly one spike.

[3] With a more complex design of the SN P system, it may be considered that these neurons do not contain spikes at the initial configuration and the vector $I = (i_1, \ldots, i_n)$ is provided as a spike train via an input neuron, but in this paper we have chosen a simpler design and focus on the computation of the immediate consequence operator. An analogous comment fits for the computed output.

- For all $p \in \{1, \ldots, k\}$ the neuron σ_{2n+p} does not contain spikes in the configuration $\mathbb{C}_{2t}$
- For all $q \in \{1, \ldots, n\}$, the neuron σ_q does not contain spikes in the configuration $\mathbb{C}_{2t}$.

In these conditions, the corresponding rules in σ_T and σ_{n+k} are fired and in $\mathbb{C}_{2t+1}$, the neuron σ_{2n+k} contains one spike. In $\mathbb{C}_{2t+2}$, the neuron σ_{n+k} contains one spike and σ_k does not contain spikes. Finally, in the next step σ_{n+k} sends one spike to σ_k, so, in $\mathbb{C}_{2t+3}$, σ_k contain one spike.

Case 2: Let us now consider that there exists $r_l \equiv B_{d_1} \wedge \ldots B_{d_l} \rightarrow p_k$ in KB such that $I(B_{d_1} \wedge \cdots \wedge B_{d_l}) = 1$ and $d_l > 0$. We suppose that $I(B_{d_1} \wedge \cdots \wedge B_{d_l}) = 1$ and this means that $I(B_{d_1}) = \cdots = I(B_{d_l}) = 1$ and therefore, in $\mathbb{C}_{2t}$, the neuron σ_{n+d_j} contains s_{d_j} spikes, with $s_{d_j} \in \{1, \ldots, h_{d_j}\}$. All these neurons fire the corresponding rule, and σ_{2n+k} has at $\mathbb{C}_{2t+1}$ exactly b_k spikes (since all the incoming synapses send the corresponding spike). The rule $a^{b_k} \rightarrow a$ is fired and in $\mathbb{C}_{2t+2}$ the neuron σ_{n+k} contains at least one spike. It may have more spikes depending on the existence of other rules with p_k in the head, but in any case, the number of spikes is between 1 and h_k. The corresponding rule fires and the neuron σ_k contains one spike in $\mathbb{C}_{2t+3}$.

Finally, we prove the statements claimed by the theorem:

(a) The SN P system computes the immediate consequence operator $T_{KB}(I)$.
Proof. It is directly obtained from *Remark 4*. Let us note that one of the possible vectors $S = (s_1, \ldots, s_n)$ obtained from the interpretation I is exactly the same interpretation $I = (i_1, \ldots, i_n)$. If we also consider the case when $t = 0$, we have proved that from the initial configuration $\mathbb{C}_0$ where i_k represents the number of spikes in the neuron σ_{n+k}, then the configuration $\mathbb{C}_3$ encodes $T_{KB}(I)$.

(b) The SN P system computes the least model for KB in a finite number of steps.
Proof. Let us consider the empty interpretation as the initial one, i.e., $T_{KB} \uparrow 0 = I_\emptyset$. We will prove that

$$(\forall z \geq 1)\, T_{KB} \uparrow z = \mathbb{C}_{2z+1}[1, \ldots, n]$$

where $\mathbb{C}_{2z+1}[1, \ldots, n]$ is the vector whose components are the spikes on the neurons $\sigma_1, \ldots, \sigma_n$ in the configuration $\mathbb{C}_{2z+1}$. We will prove it by induction.

For $z = 1$, we have to prove that $T_{KB} \uparrow 1 = T_{KB}(T_{KB} \uparrow 0) = T_{KB}(I_\emptyset)$ is the vector whose components are the spikes on the neurons $\sigma_1, \ldots, \sigma_n$ in the configuration $\mathbb{C}_3$. The result holds from *Remark 4* and it has been proved in the statement *(a)* of the theorem. By induction, let us consider now that $T_{KB} \uparrow z = \mathbb{C}_{2z+1}[1, \ldots, n]$ holds. As previously stated, this means that in the previous configuration $\mathbb{C}_{2z}$ the spikes in the neurons $\sigma_{n+1}, \ldots, \sigma_{2n}$ can be represented as a vector $S = (s_1, \ldots, s_n)$ be a vector with the properties claimed in *Remark 4*, namely, if the neuron σ_j has no spikes in $\mathbb{C}_{2z+1}$, then $s_j = 0$ and, if the

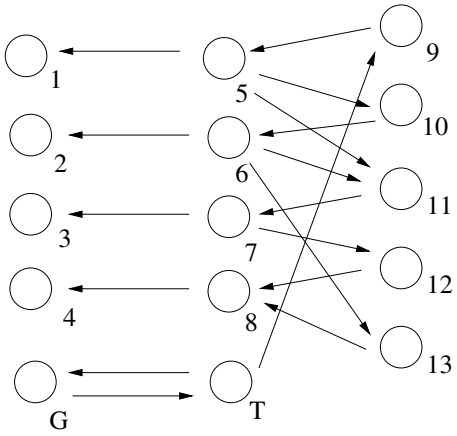


Fig. 1. Graphical representation of the synapses of the SN P system of Example 1.

neuron σ_j has spikes in $\mathbb{C}_{2z+1}$, then $s_j \in \{1, \ldots, h_j\}$. Hence, according to *Remark 4*, three computational steps after $\mathbb{C}_{2z}$, $T_{KB}(\mathbb{C}_{2z+1}[1, \ldots, n])$ is computed

$$T_{KB} \uparrow z+1 = T_{KB}(T_{KB} \uparrow z) = T_{KB}(\mathbb{C}_{2z+1}[1, \ldots, n]) = \mathbb{C}_{2z+3}[1, \ldots, n]$$

Finally, it is well-known that for a database KB, $T_{KB} \uparrow z \subseteq T_{KB} \uparrow z+1$ and, since the KB has a finite number of variables and a finite number of rules, then there exist $n \in \mathbb{N}$ such that $T_{KB} \uparrow n \subseteq T_{KB} \uparrow \omega$ and hence, $T_{KB} \uparrow n$ is a model for KB. □

Example 2. Let us consider the deductive database from Example 1. The SN P system associated with this KB and the interpretation $I_\emptyset$ is

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_{13}, \sigma_G, \sigma_T, syn)$$

where $O = \{a\}$,

$$\begin{array}{lll}
\sigma_1 = (0, \{r_{1,1} \equiv a \to a\}) & \sigma_5 = (0, \{r_{5,1} \equiv a \to a\}) & \sigma_9 = (0, \{r_{9,1} \equiv a \to a\}) \\
\sigma_2 = (0, \{r_{2,1} \equiv a \to a\}) & \sigma_6 = (0, \{r_{6,1} \equiv a \to a\}) & \sigma_{10} = (0, \{r_{10,1} \equiv a \to a\}) \\
\sigma_3 = (0, \{r_{3,1} \equiv a \to a\}) & \sigma_7 = (0, \{r_{7,1} \equiv a \to a\}) & \sigma_{11} = (0, \left\{ \begin{array}{l} r_{11,1} \equiv a \to \lambda \\ r_{11,2} \equiv a^2 \to a \end{array} \right\}) \\
\sigma_4 = (0, \{r_{4,1} \equiv a \to a\}) & \sigma_8 = (0, \left\{ \begin{array}{l} r_{8,1} \equiv a \to a \\ r_{8,2} \equiv a^2 \to a \end{array} \right\}) & \sigma_{12} = (0, \{r_{12,1} \equiv a \to a\}) \\
 & & \sigma_{13} = (0, \{r_{13,1} \equiv a \to a\})
\end{array}$$

$\sigma_G = (0, \{r_{G,1} \equiv a \to a\}$ and $\sigma_T = (0, \{r_{T,1} \equiv a \to a\}$ with the synapses

$$syn = \left\{ \begin{array}{l} (5,1),(6,2),(7,3),(8,4),(5,10),(5,11), \\ (6,11),(6,13),(7,12),(9,5),(10,6),(11,7), \\ (12,8),(13,8),(G,T),(T,G),(T,9) \end{array} \right\}$$

Let us consider the first steps of the computation

$Conf.$	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_G	σ_T
$\mathbb{C}_0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$\mathbb{C}_1$	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
$\mathbb{C}_2$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
$\mathbb{C}_3$	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0
$\mathbb{C}_4$	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1
$\mathbb{C}_5$	1	1	0	0	0	0	0	0	1	1	2	0	1	1	0
$\mathbb{C}_6$	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1
$\mathbb{C}_7$	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0

We have obtained
$T_{KB} \uparrow 0 = \mathbb{C}_1[1,\ldots,4] = (0,0,0,0)$
$T_{KB} \uparrow 1 = \mathbb{C}_3[1,\ldots,4] = (1,0,0,0)$
$T_{KB} \uparrow 2 = \mathbb{C}_5[1,\ldots,4] = (1,1,0,0)$
$T_{KB} \uparrow 3 = \mathbb{C}_7[1,\ldots,4] = (1,1,1,1)$

4 Conclusions

Biological neurons have a binary behaviour depending on a threshold. If the threshold is reached, the neuron is triggered and it sends a spike to the next neurons. If it is not reached, nothing is sent. This binary behaviour can be exploited in order to design connectionist systems which are able to deal with two-valued logic-based reasoning systems. In this paper, we have proved that SN P systems are able to deal with the semantics of deductive databases. Namely, we have proved that the immediate consequence operator can be iteratively computed with such devices by using an appropriate representation. This pioneer work opens a door for future bridges between SN P systems and logic-based reasoning systems.

References

1. Apt, K.R.: Logic Programming. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 493–574. The MIT Press (1990)
2. Bader, S., Hitzler, P.: Dimensions of neural-symbolic integration - a structured survey. In: Artemov, S., Barringer, H., d'Avila Garcez, A.S., Lamb, L., Woods, J. (eds.) We Will Show Them: Essays in Honour of Dov Gabbay, vol. 1, pp. 167–194. King's College Publications (2005)

3. Besold, T.R., Kühnberger, K.U.: Towards integrated neural-symbolic systems for human-level AI: Two research programs helping to bridge the gaps. Biologically Inspired Cognitive Architectures 14, 97 – 110 (2015)
4. Doets, K.: From logic to logic programming. Foundations of computing, MIT Press, Cambridge (Mass.) (1994)
5. van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. Journal of the ACM 23(4), 733–742 (1976)
6. Gerstner, W., Kistler, W.: Spiking neuron models: single neurons, populations, plasticity. Cambridge University Press (2002)
7. Gutiérrez-Naranjo, M.A., Rogojin, V.: Deductive databases and P systems. Computer Science Journal of Moldova 12(1), 80–88 (2004)
8. Hammer, B., Hitzler, P. (eds.): Perspectives of Neural-Symbolic Integration, Studies in Computational Intelligence, vol. 77. Springer (2007)
9. Hitzler, P., Seda, A.K.: Mathematical Aspects of Logic Programming Semantics. Chapman and Hall / CRC studies in informatics series, CRC Press (2011)
10. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71(2-3), 279–308 (2006)
11. Ivanov, S., Alhazov, A., Rogojin, V., Gutiérrez-Naranjo, M.A.: Forward and backward chaining with P systems. International Journal on Natural Computing Research 2(2), 56–66 (2011)
12. Lloyd, J.: Foundations of Logic Programming. Symbolic computation: Artificial intelligence, Springer (1987)
13. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford, England (2010)

Remarks on the Computational Power of Some Restricted Variants of P Systems with Active Membranes

Zsolt Gazdag, Gábor Kolonits

Department of Algorithms and their Applications
Faculty of Informatics
Eötvös Loránd University, Budapest, Hungary
{gazdagzs,kolomax}@inf.elte.hu

Summary. In this paper we consider three restricted variants of P systems with active membranes: (1) P systems using out communication rules only, (2) P systems using elementary membrane division and dissolution rules only, and (3) polarizationless P systems using dissolution and restricted evolution rules only. We show that every problem in **P** can be solved with uniform families of any of these variants. This, using known results on the upper bound of the computational power of variants (1) and (3) yields new characterizations of the class **P**. In the case of variant (2) we provide a further characterization of **P** by giving a semantic restriction on the computations of P systems of this variant.

1 Introduction

P systems with active membranes were introduced in [19]. These P systems have the possibility of dividing elementary (or even non-elementary) membranes. It was soon discovered that this feature (combined with maximal parallelism) makes this variant a rather powerful computational device, and efficient solutions of problems that are complete in **NP** [10, 19, 24, 30] (or even in **PSPACE** [1, 28]) were given. In order to establish the connection between classical complexity classes and P system families, recognizer P systems were introduced in [23, 25]. Since then recognizer P systems are considered as the natural framework to study the computational power of various classes of P system families. Among the many research lines in Membrane Computing, one is to find efficient solutions of computationally hard problems by various types of recognizer P systems with active membranes (see e.g. [2, 3, 4, 17, 18, 22]).

It is not too surprising that membrane division is necessary in these systems to solve computationally hard problems efficiently [30]. However, in [20] Păun conjectured that polarization is also necessary. More precisely, Păun conjectured that polarizationless P systems working in polynomial time can solve only problems

in **P**. Although this conjecture has not been proven yet, there are some partial results. In [8] it was shown that without dissolution rules these systems can solve exactly the problems in **P**. The conjecture was also confirmed in the following cases: when dissolution rules are allowed, but the P systems can employ only restricted, so-called symmetric, division rules [12], and when the initial membrane structure is a nested sequence of membranes, and the system can employ only dissolution and elementary membrane division rules [29].

It was observed in [13] that the **P** lower bound in the characterization of **P** in [8] comes from the polynomial uniformity of the examined P systems. In fact, according to [11] the used uniformity condition dominates the computational power of uniform families of polarizationless P systems with no dissolution rules. This initiated a sequence of papers where P systems with active membranes under reasonably tight uniformity conditions were examined [15, 16]. Moreover, several solutions of problems in **P** with restricted classes of P systems under tight uniformity conditions were given [5, 9, 14, 15].

In this paper we continue the work in this research line. First we show that uniform families of P systems with active membranes using out communication rules only can solve every problem in **P**. Then we show a similar result when the applicable rules are the elementary membrane division and the dissolution rules. The proofs are given by solving a restricted, but still **P**-complete variant of the well know HornSat problem, the satisfiability problem of Horn formulas.

Finally, we show that uniform families of polarizationless P systems with active membranes using dissolution and restricted evolution rules can simulate polynomial time Turing machines efficiently. The restriction made on the evolution rules is that each rule can introduce at most one object during a computation step. This result is stronger than the one appearing in [6] since there communication and not restricted evolution rules were used too. In [15] a solution of a **P**-complete problem was given using dissolution and restricted evolution rules only, however the presented family of P systems was semi-uniform.

Using the **P** upper bound given in [30], our first and third result give new characterizations of **P** in terms of Membrane Computing techniques. In our second result we use such P systems where the initial membrane structure is a nested sequence of membranes, and during the computation the number of membranes on the deepest level is at most two. It can be seen that the set of those problems that can be solved by those P systems with active membranes which have this semantic restriction during their computations are in **P**. This yields another characterization of the complexity class **P**.

The paper is organized as follows. In the next section the necessary notations and notions are recalled. In Section 3 we give the main result of the paper. Finally, some conclusions are given in the last section.

2 Preliminaries

Here we recall the necessary notions used later. Nevertheless, we assume that the reader is familiar with the basic concepts of formal language theory, propositional logic, and Membrane Computing techniques (for a comprehensive guide to these topics see e.g. [7, 21, 26], respectively). $\mathbb{N}$ denotes the set of natural numbers. For $n, m \in \mathbb{N}$, $n < m$, $[n, m]$ denotes the set $\{n, n+1, \ldots, m\}$. If $n = 1$, then $[n, m]$ is denoted by $[m]$.

Propositional formulas and the HornSat *problem.*

A *propositional variable* is a variable whose value can be either *true* or *false*. If it is not confusing, we will often call propositional variables simply *variables*. We fix an infinite set $Var = \{x_1, x_2, x_3, \ldots\}$ of variables (for the better readability of the paper we will often denote some of these variables by $x, y, z, \ldots$). For a number $n \in \mathbb{N}$, Var_n is the set $\{x_1, \ldots, x_n\}$. An *interpretation* of the variables in Var_n is a function $\mathcal{I} : Var_n \to \{true,\ false\}$.

The propositional variables and their *negations* are called *literals*. l is a *positive* (resp. *negative*) literal, if $l = x$ (resp. $l = \neg x$), for some $x \in Var$, where $\neg$ denotes the operation of *negation*. A *clause* $\mathcal{C}$ is a *disjunction* of finitely many pairwise different literals satisfying that there is no $x \in Var$ such that both x and $\neg x$ occur in $\mathcal{C}$. A clause $\mathcal{C}$ is a *positive unit clause* if $\mathcal{C}$ consists of one positive literal. A formula in *conjunctive normal form* (CNF) is a conjunction of finitely many clauses. Let φ be a formula in CNF with variables in Var_n ($n \in \mathbb{N}$). We will sometimes consider φ as a finite set of clauses, where the clauses are finite sets of literals. φ is satisfiable, if there is an interpretation under which φ evaluates *true*. Moreover, φ is a *Horn formula* if every clause in φ contains at most one positive literal.

The HornSat problem sounds as follows: *given a Horn formula φ, decide if φ is satisfiable*. It is known that HornSat is **P**-complete. Let Horn3Sat be that restriction of HornSat where every clause of the input formula can contain at most three literals. Moreover, let Horn3SatNorm be that restriction of Horn3Sat where the input formula is in the following normal form: every clause of the formula is a positive unit clause or it contains exactly two negative literals. For example, $x \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z \vee u)$ is an instance of Horn3Sat, but not of Horn3SatNorm, since $(\neg x \vee y)$ neither is a positive unite clause nor contains exactly two negative literals.

Next we show that Horn3SatNorm is **P**-complete. The proof resembles to that of the **NP**-completeness of the 3Sat problem (the 3Sat problem is the satisfiability problem of those formulas in CNF which can have only clauses with three literals, see e.g. [27]).

Proposition 1. Horn3SatNorm *is* **P***-complete.*

Proof. Since this problem is a restriction of HornSat, it is in **P**. Thus, it is enough to show that HornSat can be reduced using logarithmic space to

HORN3SATNORM. First we show that HORNSAT reduces to HORN3SAT. Let φ be a Horn formula over the variables in Var_n ($n \in \mathbb{N}$). We construct an instance φ' of HORN3SAT such that φ' is satisfiable if and only if φ is satisfiable. Let $\mathcal{C}$ be a clause in φ. If $\mathcal{C}$ has at most three literals, then let $\mathcal{C}$ be a clause of φ'. Otherwise, assume that $\mathcal{C} = x_1 \vee \neg x_2 \vee \cdots \vee \neg x_k$ for some $k \in [4, n]$. It can be easily seen that $\mathcal{C}$ is satisfiable if and only if $(x_1 \vee \neg x_2 \vee \neg y) \wedge (y \vee \neg x_3 \vee \cdots \vee \neg x_k)$ is satisfiable, where y is a new variable, not included in Var_n. In this way we can construct the formula $(x_1 \vee \neg x_2 \vee \neg y_1) \wedge (y_1 \vee \neg x_3 \vee \neg y_2) \wedge \cdots \wedge (y_{k-3} \vee \neg x_{k-1} \vee \neg x_k)$, which is satisfiable (over $Var_n \cup \{y_1, \ldots, y_{k-3}\}$) if and only if $\mathcal{C}$ is satisfiable (over Var_n). To a clause with no positive literal one can give a very similar construction. Then we add these new clauses to φ'. Clearly, φ' is satisfiable if and only if φ is satisfiable, and the mapping $\varphi \mapsto \varphi'$ can be carried out by a deterministic Turing machine using logarithmic space in the size of φ.

Next we show that HORN3SAT reduces to HORN3SATNORM. To this end let φ be an instance of HORN3SAT over the variables in Var_n. We construct an instance φ' of HORN3SATNORM such that φ' is satisfiable if and only if φ is satisfiable. For every clause $\mathcal{C}$ of φ, if $\mathcal{C}$ corresponds to the restrictions made on the instances of HORN3SATNORM, then let $\mathcal{C}$ be a clause of φ'. Otherwise we replace $\mathcal{C}$ with the set $\mathcal{C}'$ of clauses defined as follows:

- if $\mathcal{C} = \neg x$, then let $\mathcal{C}' = \{\neg x \vee \neg y, y\}$,
- if $\mathcal{C} = x_1 \vee \neg x_2$, then let $\mathcal{C}' = \{x_1 \vee \neg x_2 \vee \neg y, y\}$, and
- if $\mathcal{C} = \neg x_1 \vee \neg x_2 \vee \neg x_3$, then let $\mathcal{C}' = \{\neg x_1 \vee \neg x_2 \vee y, \neg y \vee \neg x_3\}$,

where y is always a new variable not used yet during the construction. Clearly the clauses in $\mathcal{C}'$ always have the desired forms, and φ' is satisfiable if and only if φ is satisfiable. Moreover, the described construction can be carried out by a logarithmic space Turing machine. Thus, since logarithmic space reductions are closed under composition, we have that HORNSAT can be efficiently reduced to HORN3SATNORM, which finishes the proof of the statement.

Turing machines.

In this paper we will use that variant of Turing machines which appears, e.g., in [27]. A *(deterministic) Turing machine* is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where

- Q is the finite set of *states*,
- Σ is the *input alphabet*,
- Γ is the tape alphabet including Σ and a distinguished symbol $\sqcup \notin \Sigma$, called the *blank symbol*,
- $\delta : (Q - \{q_a, q_r\}) \times \Gamma \to Q \times \Gamma \times \{-1, 1\}$ is the *transition function*; the ith component of $\delta(q, X)$ ($i \in [1, 3], q \in Q - \{q_a, q_r\}, X \in \Gamma$) is denoted by $\mathrm{proj}_i(\delta(q, X))$,
- q_0, q_a, and q_r are the *initial*, *accepting*, and *rejecting states*, respectively.

M works on a single infinite tape that is closed on the left-hand side. During the computation of M, the tape contains only finitely many non-blank symbols, and it is blank elsewhere. Let $w \in \Sigma^*$. The initial configuration of M on w is the configuration where w is placed at the beginning of the tape, the head points to the first letter of w, and the current state of M is q_0. A computation step performed by M can be described as follows. If M is in state p and the head of M reads the symbol X, then M changes its state to q and writes X' onto X if and only if $\delta(p, X) = (q, X', d)$, for some $d \in \{-1, 1\}$. Moreover, if $d = 1$ (resp. $d = -1$), then M moves its head one position to the right (resp. to the left) (by definition, M can never move the head off the left-hand end of the tape even if the head points to the first cell and $d = -1$). We say that M accepts (resp. rejects) w, if M can reach from the initial configuration on w the accepting state q_a (resp. the rejecting state q_r). We note here that M can stop only in these states. The language accepted by M is the set $L(M)$ consisting of those words in Σ^* that are accepted by M.

P systems with active membranes.

In this paper we consider several restricted variants of P systems with active membranes. In general, a *P system* with active membranes [19] is a construct of the form $\Pi = (\Gamma, H, \mu, w_1, \ldots, w_m, R)$, where m is the initial *degree* of the system, Γ is the alphabet of *objects*, H is a finite set of *labels* of the membranes; μ is a *membrane structure* consisting of m membranes and labelled with elements of H; $w_1, \ldots, w_m \subseteq \Gamma^*$ are the *initial multisets of objects* placed in the m regions of μ; and R is a finite set of *rules* defined as follows:

(a) $[a \to v]_h^e$, for $e \in \{+, -, 0\}$, $h \in H, a \in \Gamma, v \in \Gamma^*$
(object *evolution* rules, associated with membranes and depending on the label and the charge of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part in the application of these rules nor are they modified by them);

(b) $a[\ ]_h^{e_1} \to [b]_h^{e_2}$, for $e_1, e_2 \in \{+, -, 0\}$, $h \in H$, $a, b \in \Gamma$
(*in communication* rules, sending an object into a membrane, maybe modified during this process; also the polarization of the membrane can be modified, but not its label);

(c) $[a]_h^{e_1} \to [\ ]_h^{e_2} b$, for $e_1, e_2 \in \{+, -, 0\}$, $h \in H$, $a, b \in \Gamma$
(*out communication* rules; an object is sent out of the membrane, maybe modified during this process; also the polarization of the membrane can be modified, but not its label);

(d) $[a]_h^e \to b$, for $e \in \{+, -, 0\}$, $h \in H$, $a, b \in \Gamma$
(*membrane dissolving* rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(e) $[a]_h^{e_1} \to [b]_h^{e_2} [c]_h^{e_3}$, for $e_1, e_2, e_3 \in \{+, -, 0\}$, $h \in H$, $a, b, c \in \Gamma$
(*division* rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with possibly different polarizations; the object a specified in the rule is replaced in the two new membranes by (possibly new) objects b and c respectively, and the remaining objects are duplicated).

As it is usual in membrane computing, P systems with active membranes work in a *maximally parallel* manner:

- In one step, any object of a membrane that can be used by a rule must be used, but one object can be used by only one rule in (a)-(e).
- If an object can be used by two or more different rules, then one of these rules is non-deterministically chosen.
- A membrane can be the subject of only one rule in (b)-(d) during each step.

We say that an evolution rule $[a \to v]_h^e$ is 1-*restricted* if $|v| \leq 1$ (i, the number of objects in v is at most one). A *layer* is a nested membrane structure, that is a layer has the form $[\dots[\ ]_{h_1}\dots]_{h_n}$ $(n \geq 1, h_1, \dots, h_n \in H)$. For two layers $\mu_1 = [\dots[\ ]_{h_1}\dots]_{h_j}$ and $\mu_2 = [\dots[\ ]_{g_1}\dots]_{g_k}$ $(j, k \geq 1, h_1, \dots, h_j, g_1, \dots, g_k \in H)$, the *composition* $\mu_1[\mu_2]$ of μ_1 and μ_2 is the layer $[\dots[[\dots[\ ]_{g_1}\dots]_{g_k}]_{h_1}\dots]_{h_j}$. A *region* is a composition of finitely many layers.

Recognizer P systems.

A *recognizer P system* [23, 25] is a P system Π with a designated *input* membrane and having the following properties. The alphabet Γ of objects has two designated elements *yes* and *no*. Every computation of Π halts and sends to the environment the same object which is either *yes* or *no*, and these objects are sent out in the last step of the computation (if the examined P system model does not have out communication rules, then the output of the systems appears in the skin membrane). The *input* of Π is a multiset over Γ, which is added to the input membrane of the system in the initial configuration.

Uniform families of P systems.

A family $\mathbf{\Pi} = \{\Pi(i)\}_{i\in\mathbb{N}}$ of recognizer P systems *decides a problem* L if, for every instance x of L with length n, starting $\Pi(n)$ with an appropriate encoding of x in its input membrane, $\Pi(n)$ sends to the environment *yes* if and only if $x \in L$.

We will use uniform families of recognizer P systems to solve problems in $\mathbf{P}$. Clearly, we should use such a uniformity condition that is reasonably weak to work with in class $\mathbf{P}$. According to the widely believed fact that Turing machines using logarithmic space are strictly weaker than Turing machines working in polynomial time, we will use logarithmic space uniform families of P systems. We denote by $\mathbf{L}$ the family of functions that can be computed by Turing machines using logarithmic amount of space.

Assume that a family $\mathbf{\Pi} = \{\Pi(i)\}_{i\in\mathbb{N}}$ of recognizer P systems decides a problem L. $\mathbf{\Pi}$ is called **(L,L)**-*uniform* if and only if (i) there are functions $f, cod \in \mathbf{L}$ such that, for every $n \in \mathbb{N}$, $\Pi(n) = f(1^n)$ (i.e., the P system $\Pi(n)$ can be constructed by a logarithmic space Turing machine from the unary representation of n); (ii) for every instance x of L with size n, $cod(x)$ is a multiset encoding x over the alphabet of objects in $\Pi(n)$.

For a type $\mathcal{F}$ of recognizer P systems, we denote by $(\mathbf{L}, \mathbf{L}) - \mathbf{PMC}_{\mathcal{F}}$ the class of those problems that can be decided by $(\mathbf{L}, \mathbf{L})$-uniform families of P systems

of type $\mathcal{F}$ working in polynomial time. $\mathcal{AM}_{+out}$ (resp. $\mathcal{AM}_{+e,+d}$) denotes the family of P systems with active membranes having out communication (resp. division and dissolution) rules only. Similarly, $\mathcal{AM}^0_{+evo(1),+d}$ denotes the family of polarizationless P systems having 1-restricted evolution and dissolution rules only.

3 Results

Here we show that recognizer P systems of type $\mathcal{AM}_{+out}$, $\mathcal{AM}_{+e,+d}$, or $\mathcal{AM}^0_{+evo(1),+d}$ and working in polynomial time are capable to solve every problem in **P**. First we consider two solutions of Horn3SatNorm, then we give an efficient simulation of Turing machines.

3.1 The solution of Horn3SatNorm

By definition, if φ is an instance of Horn3SatNorm, then every clause of φ is either a positive unit clause or it has exactly two negative literals. In the rest of this section by a clause we mean a clause having this property. Using the well known equivalences of propositional logic, a clause having exactly two negative literals $\neg x$ and $\neg y$ can be written in the form $x \wedge y \rightarrow\downarrow$ or $x \wedge y \rightarrow z$, where z is a variable, $\rightarrow$ denotes the operation of implication and $\downarrow$ denotes a formula with constant *false* truth value. We will often use these expressions to denote the corresponding clauses of the input formula (in fact, we will often call these expressions clauses, although strictly speaking they are not clauses). Moreover, for the sake of simplicity, we will not indicate the sign $\wedge$ of conjunction in the left-hand side of these expressions.

Let φ be an instance of Horn3SatNorm. Clearly, if φ is *true* in an interpretation I, then $I(x) = true$ must hold for every positive unite clause $\{x\}$ in φ. Assume now that $\mathcal{C} = xy \rightarrow z$ is a clause of φ, where x, y are variables and z is either a variable or $\downarrow$. We observe that if $I(x) = I(y) = true$, then $\mathcal{C}$ is *true* in I if and only if z is *true* too. That is, if $z =\downarrow$, then x, y, z cannot be all *true* in I. We will use these observations in the following algorithm H3SN, which decides if an instance φ of Horn3SatNorm over variables in Var_n $(n \geq 1)$ is satisfiable or not. Let $\mathcal{N}(n)$ denote the set of those clauses over variables in Var_n which contain exactly two negative literals, and let $m = |\mathcal{N}(n)|$. In the rest of this section we assume a fixed enumeration $c_1, \ldots, c_m$ of clauses in $\mathcal{N}(n)$.

Algorithm H3SN

1. **input**: φ
2. $X := \{x \in Var_n \mid x \in \varphi\}$ // x is a positive unit clause in φ
3. **For** $i = 1 \ldots n$ do
4. **For** $j = 1 \ldots m$ **do**
5. **If** $c_j = xy \rightarrow u \in \varphi$ **and** $x, y \in X$ **then** $X := X \cup \{u\}$
6. **If** $\downarrow$ is in X **then return** *no*
7. **else return** *yes*

To demonstrate the work of H3SN consider the following example. Let $\varphi = x \wedge y \wedge (xy \to z) \wedge (xz \to\downarrow)$. Then, initially, $X = \{x, y\}$. Since $x, y \in X$ and $xy \to z \in \varphi$, X becomes $\{x, y, z\}$. Then, since $x, z \in X$ and $xz \to\downarrow\in \varphi$, X becomes $\{x, y, z, \downarrow\}$. After this the value of X remains the same until H3SN halts. Thus, since $\downarrow\in X$, H3SN outputs *no*. This is correct as φ is unsatisfiable.

In this section we give two families of P systems with rather restricted sets of applicable rules to solve the Horn3SatNorm problem in polynomial time. Both solutions are based on Algorithm $HS3N$. In these solutions the P systems cannot employ evolution and in communication rules. In addition, in the first solution dissolution and membrane division rules, while in the second solution out communication rules are also not allowed.

In both solutions the P systems, roughly, work as follows. Let φ be an instance of Horn3SatNorm over the variables in Var_n. The initial membrane structure consists of n regions, and the innermost membrane contains $cod(\varphi)$ (that is, the encoding of φ). A region r_i corresponds to the ith round of the main loop in Algorithm H3SN.

For an arbitrary clause $\mathcal{C}$ with variables in Var_n, $cod(\varphi)$ contains an object $O^{\exists}_{\mathcal{C}}$ or $O^{\nexists}_{\mathcal{C}}$ (but not both) according to that $\mathcal{C}$ occurs in φ or not. Moreover, for every clause of the form $xy \to u$ ($x, y \in Var_n, u \in Var_n \cup \{\downarrow\}$), r_i has a layer l whose membranes are indexed by this clause. The objects in the inner membrane of l go through l (either by out communication or by dissolution rules, according to the used model), and during this the system performs the following task. It first checks whether all the objects $O^{\exists}_{xy\to u}$, $O^{\exists}_{x}$, $O^{\exists}_{y}$, and $O^{\nexists}_{u}$ were present in the innermost membrane of l. If yes, then the system rewrites $O^{\nexists}_{u}$ to $O^{\exists}_{u}$. In this way the system can determine which variables of φ must be *true* in order to make φ *true* in an interpretation. After performing the above task in all layers of region r_n, the skin contains either $O^{\exists}_{\downarrow}$ or $O^{\nexists}_{\downarrow}$. If $O^{\exists}_{\downarrow}$ occurs in the skin, then φ cannot be satisfied and the system introduces object *no*, otherwise it introduces *yes*.

Formally, we encode an instance φ of Horn3SatNorm with variables in Var_n as follows. First, let

$$\Sigma(n) = \{O^e \mid O \in V(n) \cup C(n), e \in \{\exists, \nexists\}\},$$

where $V(n) = \{V_u \mid u \in Var_n \cup \{\downarrow\}\}$ and $C(n) = \{C_{xy\to u} \mid x, y \in Var_n, u \in Var_n \cup \{\downarrow\}\}$. Then the encoding of φ is $cod(\varphi) = \{O^{\exists}_c \in \Sigma(n) \mid c \in \varphi\} \cup \{O^{\nexists}_c \in \Sigma(n) \mid c \notin \varphi\} \cup \{V^{\nexists}_{\downarrow}\}$. We note here that technically there is no need to distinguish in the notation between positive unite clauses and clauses having two negative literals. Nevertheless, we decided to do so to improve the readability of the constructions. Since the size of φ is clearly polynomial in n, it can be seen that *cod* is a function in **L**.

A solution using out communication rules only.

Here we solve Horn3SatNorm with a family $\mathbf{\Pi} = \{\Pi(n)\}_{n\in\mathbb{N}}$ of recognizer P systems of type $\mathcal{AM}_{+out}$, where $\Pi(n) = (\Gamma(n), H(n), \mu(n), W(n), R(n))$ is defined as follows:

- $\Gamma(n) = \Sigma(n) \cup \{V_x^{\exists +} \mid x \in Var_n \cup \{\downarrow\}\} \cup \{yes, no\}$.
- $H = \{(xy \to u, \alpha) \mid x, y \in Var_n, u \in Var_n \cup \{\downarrow\}, \alpha \in \{a, b, c\}\} \cup \{s_k \mid k \in [m+n]\} \cup \{skin\}$.
- $\mu(n) = S[r_n[r_{n-1}[\ldots r_2[r_1]\ldots]]]$, where $S = [[[\ ]_{s_1} \ldots]_{s_{m+n}}]_{skin}$ and, for every $i \in [n]$, r_i is a region defined as follows. $r_i = l_{c_m}[\ldots l_{c_2}[l_{c_1}]\ldots]$, where, for every $j \in [m]$, the layer l_{c_j} has the form $[[[\ ]_{(c_j,a)}]_{(c_j,b)}]_{(c_j,c)}$.
- The input membrane is the innermost membrane in the initial membrane structure.
- $W(n)$ is the sequence of empty initial multisets.
- R consists of the following subsets of rules, where $x, y \in Var_n$ and $u \in Var_n \cup \{\downarrow\}$:

  (1) $[C^{\exists}_{xy\to u}]^0_{(xy\to u,a)} \to [\ ]^+_{(xy\to u,a)} C^{\exists}_{xy\to u}$,
  $[C^{\exists}_{xy\to u}]^0_{(xy\to u,\beta)} \to [\ ]^0_{(xy\to u,\beta)} C^{\exists}_{xy\to u}$,
  $[C^{\nexists}_{xy\to u}]^0_{(xy\to u,\alpha)} \to [\ ]^-_{(xy\to u,\alpha)} C^{\nexists}_{xy\to u}$ $\quad (\alpha \in \{a, b, c\}, \beta \in \{b, c\})$.
  These rules are used to initialize the layers in the following sense: the first membranes of those layers that are indexed by a clause in φ get positive charges, the second and third membranes keep their neutral charges, while all the membranes of the remaining layers get negative charges.

  (2) $[V^e_v]^-_{(xy\to u,\alpha)} \to [\ ]^-_{(xy\to u,\alpha)} V^e_v$,
  $[C^e_{rs\to v}]^-_{(xy\to u,\alpha)} \to [\ ]^-_{(xy\to u,\alpha)} C^e_{rs\to v}$
  $(e \in \{\exists, \nexists\}, r, s \in Var_n, v \in Var_n \cup \{\downarrow\}, \alpha \in \{a, b, c\})$.
  Every membrane with negative charge lets all of the objects to pass through itself.

  (3) $[V^{\exists}_x]^+_{(xy\to u,a)} \to [\ ]^-_{(xy\to u,a)} V^{\exists +}_x$,
  $[V^{\exists +}_x]^0_{(xy\to u,b)} \to [\ ]^+_{(xy\to u,b)} V^{\exists}_x$.
  If φ has a clause $xy \to u$, that is, the membrane with label $(xy \to u, a)$ has positive charge, and $V^{\exists}_x$ exists in this membrane, then these rules are used to store this information in the positive charge of the membrane with label $(xy \to u, b)$.

  (4) $[V^{\exists}_y]^+_{(xy\to u,b)} \to [\ ]^-_{(xy\to u,b)} V^{\exists +}_y$,
  $[V^{\exists +}_y]^0_{(xy\to u,c)} \to [\ ]^+_{(xy\to u,c)} V^{\exists}_y$.
  If the membrane with label $(xy \to u, b)$ has positive charge and $V^{\exists}_y$ exists in this membrane, then these rules are used to store this information in the positive charge of the membrane with label $(xy \to u, c)$.

  (5) $[V^{\nexists}_u]^+_{(xy\to u,c)} \to [\ ]^-_{(xy\to u,c)} V^{\exists}_u$.
  The positive charge of the membrane with label $(xy \to u, c)$ indicates that $xy \to u$ is a clause of the system and that both variables x and y has to be *true* in an interpretation in order to make φ *true*. Thus, with this rule the system rewrites $V^{\nexists}_u$ to $V^{\exists}_u$ indicating that u must be also *true* to make φ *true*.

(6) $[V_u^{\exists}]^p_{(xy\to u,\alpha)} \to [\ ]^-_{(xy\to u,\alpha)} V_u^{\exists}$ $(p \in \{+,0\}, \alpha \in \{a,b,c\})$.
If the system already knows that u must be *true* to make φ *true*, then the charges of the corresponding membranes are set to negative.

(7) $[V_x^{\nexists}]^p_{(xy\to u,\alpha)} \to [\ ]^-_{(xy\to u,\alpha)} V_x^{\nexists}$,
$[V_y^{\nexists}]^p_{(xy\to u,\alpha)} \to [\ ]^-_{(xy\to u,\alpha)} V_y^{\nexists}$ $(p \in \{+,0\}, \alpha \in \{a,b,c\})$.
If any of the variables on the left-hand side of a clause $xy \to u$ is not considered to be *true* yet, then the charges of membranes of the corresponding layer are set to negative by these rules, and $V_u^{\exists}$ cannot be introduced by this layer.

(8) $[V_{\downarrow}^{e}]^0_{s_k} \to [\ ]^0_{s_k} V_{\downarrow}^{e}$, $[V_{\downarrow}^{\exists}]^0_{skin} \to [\ ]^0_{skin} no$, $[V_{\downarrow}^{\nexists}]^0_{skin} \to [\ ]^0_{skin} yes$
$(k \in [m+n], e \in \{\exists, \nexists\})$.
The first rule is used to move object $V_{\downarrow}^{\exists}$ or $V_{\downarrow}^{\nexists}$ towards the skin membrane. When they arrive at the skin, the system sends to the environment the correct answer.

Correctness, running time, and $(\mathbf{L}, \mathbf{L})$*-uniformity.*

First we observe that during the computation of $\Pi(n)$ the following holds.

1. If all the membranes in a layer l have negative charge, then l does not contribute to the computation, i.e. all objects pass through the membranes of l without any change.
2. For every $\mathcal{C} \in C(n)$, either $C_{\mathcal{C}}^{\exists}$ or $C_{\mathcal{C}}^{\nexists}$ (but not both) occurs in the system (the same object during the whole computation).
3. For every $x \in Var_n \cup \{\downarrow\}$, either $V_x^{\exists}$ or $V_x^{\nexists}$ (but not both) occurs in the system. Indeed, the rules that can change an object of this form are rules in (3)-(5) (not counting the rules that introduce *yes* or *no* at the last step of the computation). Rule in (5) removes $V_u^{\nexists}$ and introduces $V_u^{\exists}$, thus the observation remains true after applying it. Concerning rules in (3)-(4), it is enough to observe that if the first rule can be applied, then the second rule can be applied too in the next step.

Now consider a layer $l_{xy\to u}$ $(x, y \in Var_n, u \in Var_n \cup \{\downarrow\})$. At the beginning of the computation every membrane in $l_{xy\to u}$ has neutral charge. According to the objects that pass through this layer we can distinguish the following cases.

1. All of the objects $C_{xy\to u}^{\exists}$, $V_x^{\exists}$, $V_y^{\exists}$, and $V_u^{\nexists}$ pass through the membranes of $l_{xy\to u}$. Then the system rewrites the object $V_u^{\nexists}$ to $V_u^{\exists}$.
2. Any of the objects $C_{xy\to u}^{\nexists}$, $V_x^{\nexists}$, $V_y^{\nexists}$, or $V_u^{\exists}$ passes through the membranes of $l_{xy\to u}$. Then the charge of every membrane in $l_{xy\to u}$ is set to negative, and thus this layer cannot contribute to the computation. (Notice that in this case the computation is not deterministic but confluent, i.e., all the possible computations in the layer yield the same result.)

It follows that the objects passing trough the layer $l_{xy\to u}$ simulate step 5 of Algorithm H3SN. Thus, sending objects through a region corresponds to performing

steps $4-5$ of this algorithm. Since the algorithm performs steps $4-5$ n times, the work of the P system in the n regions corresponds to the work of the algorithm. Thus, $V_{\downarrow}^{\exists}$ or $V_{\downarrow}^{\nexists}$ eventually appears in membrane s_1. In the next $m+n$ steps this object gets to the skin by rules in (8). There the system computes *yes* or *no* accordingly, which is then sent to the environment. It can bee seen that during this computation all the other objects occurring in the systems arrive to membrane s_1, and the computation halts.

This justifies the correctness of $\Pi(n)$. Since $\Pi(n)$ has polynomial number of objects in the initial configuration and no evolution rules are performed during its work, sending all the objects through a region takes polynomial steps. Thus the running time of $\Pi(n)$ is also polynomial.

It can be seen that all the ingredients of $\Pi(n)$ can be enumerated and written onto the output tape by a logarithmic space Turing machine. Thus, using that HORN3SATNORM is **P**-complete, we get the following result.

Theorem 1. $\mathbf{P} \subseteq (\mathbf{L}, \mathbf{L}) - \mathbf{PMC}_{\mathcal{AM}_{+out}}$.

A solution using elementary membrane division and dissolution rules only.

In this subsection we solve HORN3SATNORM with a family $\mathbf{\Pi} = \{\Pi(n)\}_{n\in\mathbb{N}}$ of recognizer P systems of type $\mathcal{AM}_{+e,+d}$. The solution is similar to the one given in the previous subsection, however, there is a substantial difference: here the presence of the necessary objects to simulate step 5 of Algorithm H3SN are checked by the application of membrane division rules. Consequently, those objects that do not take part in the simulation are duplicated several times. In particular, at certain points of the computation the P system has multiple copies of objects of the form $V_x^{\nexists}$. However, the correctness of the computation requires that at the beginning of the work in a layer there is at most one copy of objects of this form. Therefore we will apply special layers, that will remove those objects that could cause the system to give incorrect results. The following is the formal definition of $\Pi(n) = (\Gamma(n), H(n), \mu(n), W(n), R(n))$:

- $\Gamma(n) = \Sigma(n) \cup \{w, \overline{w}_1, \overline{w}_2, \#, \$\} \cup \{yes, no\}$.
- $H(n) = \{skin, s\} \cup \{(xy \to u, \alpha) \mid x, y \in Var_n, u \in Var_n \cup \{\downarrow\}, \alpha \in \{a, b, c, d\}\} \cup \{d_O \mid O \in V(n) \cup C(n) \cup \{w\}\}$.
- $\mu(n)$ is defined as follows. Let $\mathcal{C} = xy \to u$ be a clause ($x, y \in Var_n, u \in Var_n \cup \{\downarrow\}$) and $l_{\mathcal{C}}$ be the layer $D_{\mathcal{C}}[M_{\mathcal{C}}]$, where $D_{\mathcal{C}}$ and $M_{\mathcal{C}}$ are defined as follows:

  $$M_{\mathcal{C}} = [\,[\,[\,[\,[\,[\,[\,]_{(xy\to u,a)}]_{(xy\to u,b)}]_{d_w}]_{(xy\to u,c)}]_{d_w}]_{(xy\to u,d)}$$

  and $D_{\mathcal{C}}$ is a layer containing, for every $O \in V(n) \cup C(n)$, the membrane $[\,]_{d_O}$ fifteen times if $O \neq V_u$, and once, otherwise. Intuitively, $M_{\mathcal{C}}$ is that part of the layer which is responsible to simulate step 5 in Algorithm H3SN, and layer $D_{\mathcal{C}}$ is used (together with membranes with label d_w in $M_{\mathcal{C}}$) to remove those

objects that are produced by the used division rules, but should be removed in order to keep the behaviour of the system correct.
To finish the construction, let $\mu(n) = S[r_n[r_{n-1}[\dots r_2[r_1]\dots]]$, where $S = [[\ ]_s]_{skin}$ and, for every $i \in [n]$, r_i is the region $l_{c_m}[\dots l_{c_2}[l_{c_1}]\dots]$.

- The input membrane is the innermost membrane in the initial membrane structure.
- $W(n)$ is a sequence of empty initial multisets.
- R consists of the following subsets of rules, where $x, y \in Var_n$ and $u \in Var_n \cup \{\downarrow\}$:

(1) $[V_u^{\nexists}]^0_{(xy\to u,a)} \to [w]^-_{(xy\to u,a)}[\#]^-_{(xy\to u,a)}$,
$[V_u^{\exists}]^0_{(xy\to u,a)} \to [\overline{w}_1]^-_{(xy\to u,a)}[\#]^-_{(xy\to u,a)}$.
These rules are used to decide if $V_u^{\nexists}$ or $V_u^{\exists}$ is present in a membrane with label $(xy \to u, a)$. If $V_u^{\nexists}$ is present, then the system introduces w which indicates that the system should work further to decide if $V_u^{\exists}$ should be introduced or not. Object $\overline{w}_1$ indicates that $V_u^{\exists}$ is present in the system and thus it should not be introduced later. # indicates that the membrane containing it is not used effectively in the computation.

(2) $[w]^-_{(xy\to u,a)} \to w, [\overline{w}_1]^-_{(xy\to u,a)} \to \overline{w}_1$,
$[\#]^-_{(xy\to u,a)} \to \$$.
These rules pass the information computed by rules in (1) to the membrane labelled with $(xy \to u, b)$. \$ is a dummy object not used later.

(3) $[C^{\exists}_{xy\to u}]^0_{(xy\to u,b)} \to [C^{\exists}_{xy\to u}]^+_{(xy\to u,b)}[C^{\exists}_{xy\to u}]^+_{(xy\to u,b)}$,
$[C^{\nexists}_{xy\to u}]^0_{(xy\to u,b)} \to [C^{\nexists}_{xy\to u}]^-_{(xy\to u,b)}[C^{\nexists}_{xy\to u}]^-_{(xy\to u,b)}$.
These rules decide if object $C^{\exists}_{xy\to u}$ or $C^{\nexists}_{xy\to u}$ exists in the system. The result is stored in the polarizations of the new membranes.

(4) $[w]^+_{(xy\to u,b)} \to w, [w]^-_{(xy\to u,b)} \to \overline{w}_2$,
$[\overline{w}_1]^+_{(xy\to u,b)} \to \overline{w}_1, [\overline{w}_1]^-_{(xy\to u,b)} \to \overline{w}_1$.
These rules introduce objects that will control the computation according to the information computed by the previous subsets of rules. For example, if w and $C^{\nexists}_{xy\to u}$ is present in the inner membrane, then $\overline{w}_2$ is introduced. In this case $V_u^{\exists}$ will not be introduced at the end of the computation in this layer (see rules in (8)).

(5) $[V_y^{\exists}]^0_{(xy\to u,c)} \to [V_y^{\exists}]^+_{(xy\to u,c)}[V_y^{\exists}]^+_{(xy\to u,c)}$,
$[V_y^{\nexists}]^0_{(xy\to u,c)} \to [V_y^{\nexists}]^-_{(xy\to u,c)}[V_y^{\nexists}]^-_{(xy\to u,c)}$.
These rules decide if object $V_y^{\exists}$ or $V_y^{\nexists}$ exists in the system. The result is stored in the polarizations of the new membranes.

(6) $[w]^+_{(xy\to u,c)} \to w, [w]^-_{(xy\to u,c)} \to \overline{w}_2$,
$[\overline{w}_1]^+_{(xy\to u,c)} \to \overline{w}_1, [\overline{w}_1]^-_{(xy\to u,c)} \to \overline{w}_1$,
$[\overline{w}_2]^+_{(xy\to u,c)} \to \overline{w}_2, [\overline{w}_2]^-_{(xy\to u,c)} \to \overline{w}_2$.

These rules introduce objects that will control the computation according to the information computed by the previous subset of rules.

(7) $[V_x^{\exists}]^0_{(xy\to u,d)} \to [V_x^{\exists}]^+_{(xy\to u,d)}[V_x^{\exists}]^+_{(xy\to u,d)}$,
$[V_x^{\not\exists}]^0_{(xy\to u,d)} \to [V_x^{\not\exists}]^-_{(xy\to u,d)}[V_x^{\not\exists}]^-_{(xy\to u,d)}$.
These rules decide if object $V_x^{\exists}$ or $V_x^{\not\exists}$ exists in the system. The result is stored in the polarizations of the new membranes.

(8) $[w]^+_{(xy\to u,d)} \to V_u^{\exists}, [w]^-_{(xy\to u,d)} \to V_u^{\not\exists}$,
$[\overline{w}_1]^+_{(xy\to u,d)} \to V_u^{\exists}, [\overline{w}_1]^-_{(xy\to u,d)} \to V_u^{\exists}$
$[\overline{w}_2]^+_{(xy\to u,d)} \to V_u^{\not\exists}, [\overline{w}_2]^-_{(xy\to u,d)} \to V_u^{\not\exists}$.
These rules are used to handle the different cases of possible computations in a layer. For example, w indicates that at the beginning of the computation in a layer the system contained objects $V_u^{\not\exists}$, $C^{\exists}_{xy\to u}$, and $V_y^{\exists}$.

(9) $[O^e]^0_{d_O} \to \$, [w]^0_{d_w} \to \$, [\overline{w}_i]^0_{d_w} \to \$$
$(O \in V(n) \cup C(n), e \in \{\exists, \not\exists\}, i \in [2])$.
These rules are used to remove certain objects from the system.

(10) $[V_{\downarrow}^{\not\exists}]^0_s \to [no]^-_s[\$]^-_s$, $[V_{\downarrow}^{\exists}]^0_s \to [yes]^-_s[\$]^-_s$, $[\kappa]^-_s \to \kappa \quad (\kappa \in \{yes, no\})$.
These rules are used to send out the computed answer to the environment.

Correctness, running time, and $(\mathbf{L}, \mathbf{L})$*-uniformity.*

First we observe that during the computation of $\Pi(n)$ the following holds:

1. The membrane structure has the form $[\ldots[M]_{h_1}\ldots]_{h_k}$ $(h_1, \ldots, h_k \in H(n))$, where M is either a membrane or it is of the form $[\ ]_{g_1}[\ ]_{g_2}$ $(g_1, g_2 \in H(n))$, and
2. objects occur only in the innermost membranes.

The correctness of the system follows from the following lemma.

Lemma 1. *Let* $\mathcal{C} = xy \to u$ $(x, y \in Var_n, u \in Var_n \cup \{\downarrow\})$ *and consider the layer* $l_{\mathcal{C}} = D_{\mathcal{C}}[M_{\mathcal{C}}]$. *Assume that, for every* $O \in C(n) \cup V(n)$, *either one copy of* $O^{\exists}$ *or one copy of* $O^{\not\exists}$ *occurs in* $l_{\mathcal{C}}$. *Let* O *be an object in* $l_{\mathcal{C}}$. *Depending on* O *the following holds:*

1. *If* $O \in \Sigma(n) - \{V_u^{\not\exists}\}$, *then after dissolving all the membranes in* $l_{\mathcal{C}}$, $\Pi(n)$ *contains exactly one copy of* O.
2. *If* $O = V_u^{\not\exists}$ *and* $l_{\mathcal{C}}$ *contains all of the objects* $C_{\mathcal{C}}^{\exists}$, $V_x^{\exists}$, *and* $V_y^{\exists}$, *then after dissolving all the membranes in* $l_{\mathcal{C}}$, $\Pi(n)$ *contains no* $V_u^{\not\exists}$ *and exactly one copy of* $V_u^{\exists}$.
3. *If* $O = V_u^{\not\exists}$ *and* l_c *contains* $C_{\mathcal{C}}^{\not\exists}$, $V_x^{\not\exists}$, *or* $V_y^{\not\exists}$, *then after the work in* $l_{\mathcal{C}}$ $\Pi(n)$ *contains exactly one copy of* $V_u^{\not\exists}$.

Proof. By assumption, $l_{\mathcal{C}}$ contains exactly one copy of O. Then Statement 1 can be seen by distinguishing the following two sub-cases:

Case 1. $O \neq V_u^{\exists}$. Then during the work in $M_{\mathcal{C}}$, O is duplicated by the corresponding rules rules in (1), (3), (5), and (7), and the other rules are not applied to O in $M_{\mathcal{C}}$. This yields sixteen copies of O in $D_{\mathcal{C}}$. Out of these copies fifteen ones are removed during the computation in $D_{\mathcal{C}}$.
Case 2. $O = V_u^{\exists}$. Then the second rule in (1) removes first $V_u^{\exists}$ and introduces one copy of $\overline{w}_1$. After this, membrane $(\mathcal{C}, a)$ is dissolved using rules in (2). In the next two steps, $\overline{w}_1$ is duplicated first due the division of membrane $(\mathcal{C}, b)$ by rules in (3), then the yielded membranes are dissolved by rules in (4). Thus, at this point of the computation two copies of $\overline{w}_1$ are in membrane d_w. However, in the next step one copy is removed due to the corresponding rule in (9). After this, membrane $(\mathcal{C}, c)$ is divided (rules in (5)) and the new membranes are dissolved (rules in (6)). At this point, two copies of $\overline{w}_1$ are in membrane d_w, and one copy is removed by the corresponding rule in (9). Finally, $\overline{w}_1$ is duplicated by rules in (7), and then the two copies of $\overline{w}_1$ introduce two copies of $V_u^{\exists}$. During the dissolution of membranes in $D_{\mathcal{C}}$ one copy of $V_u^{\exists}$ is removed which proves the statement.

Statement 2 can be seen as follows. The computation starts with removing the object $V_u^{\nexists}$ and introducing one w (first rule in (1)). Then the new membranes with label $(\mathcal{C}, a)$ are dissolved by the corresponding rules in (2). In membrane $(\mathcal{C}, b)$ the first rule of (3) is applied and thus w is duplicated. At this point membranes with label $(\mathcal{C}, b)$ have positive charges, thus only the first rule in (4) can be applied. After this the corresponding rule in (9) removes one copy of w. During the next step the first rule in (5) is applied, and then only the first rule in (6) can be used. Again, one copy of w is removed by the corresponding rule in (9). Then the first rule in (7) divides membrane $(\mathcal{C}, d)$, w is again duplicated, and by the first rule in (8) each w introduces one copy of $V_u^{\exists}$. During the work in $D_{\mathcal{C}}$, one copy of $V_u^{\exists}$ is removed.

The system has several different computations in the case of Statement 3. We discuss here only one of them, the remaining ones can be treated similarly. Assume for example that $l_{\mathcal{C}}$ contains $C_{\mathcal{C}}^{\exists}$ and $V_y^{\nexists}$. Then the computation goes in the same way as in the case of Statement 2 until the application of the corresponding dissolution rules in (4). But now the second rule in (5) is applied, and thus, in the next step, only the second rule in (6) can be applied. Therefore here two copies of $\overline{w}_2$ are introduced. Then the computation continues similarly as in Case 2 in the proof of Statement 1. However here, when rules from (8) are applied the system has two copies of $\overline{w}_2$, and thus two copies of $V_u^{\nexists}$ are introduced by the fifth and sixth rules in (8). One of these copies is eliminated during the work in $D_{\mathcal{C}}$.

Clearly, the initial configuration of $\Pi(n)$ satisfies the conditions of Lemma 1. Let $\mathcal{C}$ be a clause having exactly two negative literals. Let moreover $x \in Var_n \cup \{\downarrow\}$. Then at the end of the computation in layer $l_{\mathcal{C}}$ either $V_x^{\exists}$ or $V_x^{\nexists}$ occurs in the system. Therefore the computation of the system in a region corresponds to performing steps $4-5$ of Algorithm H3SN. Since this algorithm performs steps $4-5$ n times, the work of $\Pi(n)$ in the n regions corresponds to the work of the algorithm. This justifies the correctness of $\Pi(n)$.

Since $\Pi(n)$ has polynomial number of membranes in layer $l_{\mathcal{C}}$, and in $l_{\mathcal{C}}$ the number of the applied division rules is constant, we have that dissolving all the membranes in $l_{\mathcal{C}}$ takes polynomial time. As in the initial configuration there are n regions and each region has polynomial number of layers, it follows that the running time of $\Pi(n)$ is also polynomial.

Finally, it can be seen that all the ingredients of $\Pi(n)$ can be enumerated and written onto the output tape by a logarithmic space Turing machine. Using the **P**-completeness of Horn3SatNorm we obtain the following theorem.

Theorem 2. $\mathbf{P} \subseteq (\mathbf{L}, \mathbf{L}) - \mathbf{PMC}_{\mathcal{AM}_{+e,+d}}$.

3.2 Simulating Turing Machines

Here we show that, for every polynomial time Turing machine M, an $(\mathbf{L}, \mathbf{L})$-uniform family $\mathbf{\Pi}$ of polarizationless recognizer P systems can be constructed such that the members of $\mathbf{\Pi}$ can simulate the work of M efficiently using only dissolution and 1-restricted evolution rules.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ be an $f(n)$-time Turing machine, for some polynomial $f(n)$. Notice that M can use at most $f(n)$ cells of its tape during its computations. Let $k = |Q|$ and $m = |\Gamma|$. Assume that $Q = \{s_1, \ldots, s_k\}$, where $s_1 = q_0, s_{k-1} = q_a$ and $s_k = q_r$. Likewise, assume that $\Gamma = \{X_1, \ldots, X_m\}$, where $X_m = \sqcup$. The idea of the simulation is the following. The initial membrane structure μ is a composition of $f(n)$ regions. The input membrane is the innermost membrane. During the simulation of the tth step of M, the objects in the innermost membrane will dissolve all the membranes in the tth region as follows. Assume that after $t-1$ steps M is in state s_i ($i \in [k-2]$), the position of the head is p, and the head scans X_j. Then the innermost membrane of the tth region contains an object O that represents s_i and p, and another object O' representing X_j on the pth position of the tape. The regions are composed from $k \cdot m \cdot f(n)$ membranes (that is, in every region, for every state–tape symbol–position triple there is a corresponding membrane). During the simulation of the tth step of M, O dissolves all the membranes that correspond to a state $s_{i'}$ with $i' < i$ or a position $p' < p$. Meanwhile O' evolves using a counter and at the appropriate time step it starts to dissolve all the membranes corresponding to s_i, p, and tape symbol $X_{j'}$ with $j' < j$. After this the simulation of one step of M is performed using the value $\delta(s_i, X_j)$. Then the remaining membranes in the tth region are dissolved, and the system continues with the simulation of the next step of M.

Construction of the P system.

The uniform family of P systems that will perform the above described simulation is defined as follows. Let $w = a_1 \ldots a_n$ be an input of M ($a_1, \ldots, a_n \in \Sigma$) and $N = f(n) \cdot k \cdot m$. Let $cod(w)$ be a multiset over the alphabet

$$\Sigma(n) = \{(X_j, p, t)^{(c)}, (s_i, p, t)^{(c')} \mid j \in [m], i \in [k], p \in [f(n)], t \in [0, f(n)], c \in [0, N], c' \in [0, N+m]\}$$

defined as follows: $cod(w) = \{(a_1, 1, 0)^{(0)}, \ldots, (a_n, n, 0)^{(0)}\} \cup \{(\sqcup, n+1, 0)^{(0)}, \ldots, (\sqcup, f(n), 0)^{(0)}\} \cup (s_1, 1, 0)^{(0)}$. Intuitively, an object $(X_j, p, t)^{(c)}$ in $\Sigma(n)$ represents the fact that after t steps M has X_j on the pth position of its tape. We call these objects *position objects*. Similarly, an object $(s_i, p, t)^{(c')}$ represents the fact that after t steps M is in state s_i and the head points to the pth position of the tape. We call these objects *state objects*. The indexes c, c' are counters used for technical reasons. It can be seen that $cod \in \mathbf{L}$.

Let $\mathbf{\Pi} = \{\Pi(n)\}_{n\in\mathbb{N}}$ be a uniform family of P systems, where $\Pi(n) = (\Gamma(n), H(n), \mu(n), W(n), R(n))$ is defined as follows:

- $\Gamma(n) = \Sigma(n) \cup \{yes, no\}$.
- $H(n) = \{(s_i, p, X_j, t) \mid i \in [k], p, t \in [f(n)], j \in [m]\}$.
  Intuitively, a label (s_i, p, X_j, t) corresponds to the following configuration of M after t steps on w: the current state is s_i, the position of the head is p, and the scanned symbol is X_j. We will often call s_i, p, and t the *state*, *position*, and *time* labels of the corresponding membrane, respectively.
- $\mu(n)$ is a composition $S[r_{f(n)}[\ldots[r_1]]]$ of regions, where $S = [\ ]_{skin}$, and a region r_t ($t \in [f(n)]$) is a composition of layers defined as follows. For every $i \in [k]$ and $p \in [f(n)]$, let $l_{s_i,p,t} = [\ldots[\ ]_{(s_i,p,X_1,t)}\ldots]_{(s_i,p,X_m,t)}$, and let $r_t = l_{s_k,f(n),t}[\ldots[l_{s_k,1,t}[\ldots[l_{s_1,f(n),t}[\ldots[l_{s_1,1,t}]\ldots]]\ldots]]\ldots]$.
- The input membrane is the innermost membrane in $\mu(n)$.
- $W(n)$ is a sequence of empty initial multisets.
- R consists of the following sets of rules:

  (1) $[(s_i, p, t)^{(0)}]_{(s_{i'}, p', X_j, t+1)} \to (s_i, p, t)^{(0)}$
  ($j \in [m]$, $i \in [k-2], i' \in [k]$, $p, p' \in [f(n)], t \in [0, f(n)-1]$, and $i' < i$ or $p' < p$).
  These rules are used to find the first such membrane whose state and position labels correspond to the state and position stored in the state object.

  (2) $[(X_j, p, t)^{(c)} \to (X_j, p, t)^{(c+1)}]_{(s_i, p', X_{j'}, t+1)}$
  ($j, j' \in [m], i \in [k], p, p' \in [f(n)], t \in [0, f(n)-1], c \in [0, N-1]$).
  These rules are used to increment the counter c in the position objects. When this counter equals to N, the system can start to use rules in (3).

  (3) $[(X_j, p, t)^{(N)}]_{(s_i, p, X_l, t+1)} \to (X_j, p, t)^{(N)}$,
  $[(X_j, p, t)^{(N)} \to (X_{j'}, p, t+1)^{(0)}]_{(s_i, p, X_j, t+1)}$,
  $[(X_j, p', t)^{(N)} \to (X_j, p', t+1)^{(0)}]_{(s_i, p, X_1, t+1)}$
  ($j, l \in [m]$, $l < j$, $i \in [k-2]$, $p, p' \in [f(n)]$, $p \neq p'$, $t \in [0, f(n)-1]$, and $X_{j'} = \mathrm{proj}_2(\delta(s_i, X_j))$).
  If the position stored in an object $(X_j, p, t)^{(N)}$ corresponds to the position label of the current membrane, then this object starts to dissolve the membranes until a membrane whose label stores X_j is found. When this membrane is found, $(X_j, p, t)^{(N)}$ evolves according to the value of $\delta(s_i, X_j)$, its counter is reset, and its component t is incremented. Those position objects that store a different position than the position label of the current

membrane evolve immediately such that their counter is reset and their component t is incremented. Notice that after performing the computations by these rules, the position objects have no impact on the computation in region r_{t+1}.

(4) $[(s_i, p, t)^{(c)} \to (s_i, p, t)^{(c+1)}]_{(s_i,p,X_l,t+1)}$,
$[(s_i, p, t)^{(N+m)} \to (s_{i'}, p', t+1)^{(0)}]_{(s_i,p,X_l,t+1)}$
$(i \in [k-2], i' \in [k]$, $p \in [f(n)]$, $t \in [0, f(n)-1]$, $c \in [N+m-1]$, $l \in [m]$, $s_{i'} = \text{proj}_1(\delta(s_i, X_l))$, $p' = \max\{p + \text{proj}_3(\delta(s_i, X_l)), 1\})$.
The counter of the state object is incremented using the first rule. Until the counter reaches $N+m$, the appropriate position object can find the corresponding membrane using rules in (3). Then the state object evolves according to the value of the transition function of M. Moreover, its counter is reset and its component t is incremented.

(5) $[(s_i, p, t+1)^{(0)}]_{(s_{i'},p',X_j,t+1)} \to (s_i, p, t+1)^{(0)}$
$(i \in [k-2], i' \in [k], p, p' \in [f(n)], j \in [m], t \in [0, f(n)-1])$.
After simulating a step of M using rules in (1)-(4), the remaining membranes in region r_{t+1} are dissolved by these rules.

(6) $[(s_{k-1}, p, t)^{(0)} \to yes]_h$, $[(s_k, p, t)^{(0)} \to no]_h$,
$[yes]_{h'} \to yes$, and $[no]_{h'} \to no$
$(p, t \in [f(n)], h \in H(n), h' \in H(n) - \{skin\})$.
These rules are used to produce the answer of $\Pi(n)$ according to which halting state is reached by M on the input.

Correctness, running time, and $(\mathbf{L}, \mathbf{L})$*-uniformity.*

Let $w = a_1 \dots a_n$ be an input of M ($a_1, \dots, a_n \in \Sigma$). We show that $\Pi(n)$ produces *yes* started with $cod(w)$ in its input membrane if and only if $w \in L(M)$. The work of $\Pi(n)$ can be described as follows. Initially, the object $(s_1, 1, 0)^{(0)}$ (representing that M starts its work in its initial state and the head is positioned to the first letter of the input) is in the innermost membrane of region r_1. Now assume that $\Pi(n)$ has already simulated t steps of M, that is, the innermost membrane of $\Pi(n)$ is the most deeply nested membrane of region r_{t+1}, and this membrane contains an object $(s_i, p, t)^{(0)}$, for some $i \in [k]$ and $p \in [f(n)]$. If $i \in [k-1, k]$, i.e., M has reached one of its halting states, then $\Pi(n)$, using rules in (6) computes the answer of the system *yes* or *no* accordingly. Otherwise, rules from (1) are applied until a membrane with label $(s_i, p, X_1, t+1)$ is reached. Meanwhile, the counter c in position objects is incremented using rules in (2). By the time this counter becomes N, the corresponding membrane is reached by the rules in (1).

Now those position objects that store different positions than p evolve by the third rule in (3) to such objects that will be used next time only in the next region r_{t+2} (i.e., in the simulation of the next step of M). Concerning the position object storing p, assume that this object is $(X_j, p, t)^{(N)}$. Then $(X_j, p, t)^{(N)}$ is used to find that membrane in layer $l_{s_i,p,t+1}$ whose label contains X_j. When this membrane is

found, $(X_j, p, t)^{(N)}$ evolves according to the transition function of M. Moreover, its counter is reset and its time component is incremented. Thus this object is not used any more in this region.

Meanwhile, rules in (4) are used to increment the counter c of $(s_i, p, t)^{(c)}$. By the time this counter becomes $N+m$, the position object $(X_j, p, t)^{(N)}$ has reached the membrane it searched for. Now the second rule in (4) is used to produce object $(s_{i'}, p', t+1)^{(0)}$ where $s_{i'}$ and p' are calculated according to the transition function of M. Finally, $(s_{i'}, p', t+1)^{(0)}$ is used to dissolve the remaining membranes of r_{t+1}. If this is done, the system is ready to simulate the next step of M. With this we have seen that $\Pi(n)$ simulates correctly the computation of M on w.

It can be seen that dissolving a region in the membrane structure takes $O(N)$ steps and $N = O(f(n))$. Moreover, $\Pi(n)$ has $f(n)$ regions. Thus the running time of the system is $O(f^2(n))$, that is, polynomial in n. The $(\mathbf{L}, \mathbf{L})$-uniformity of $\mathbf{\Pi}$ follows from the observation that the size of $\Pi(n)$ is also polynomial in n. Thus we have the following result.

Theorem 3. $\mathbf{P} \subseteq (\mathbf{L}, \mathbf{L}) - \mathbf{PMC}_{\mathcal{AM}^0_{+evo(1),+d}}$.

As we have observed on page 197, our solution of Horn3SatNorm by P systems of type $\mathcal{AM}_{+e,+d}$ is such that every membrane has at most two child membranes in every configuration of each computation of the system. Let $k \geq 1$. We say that a P system Π is k-bounded, if every membrane has at most k child membranes in every configuration of each computation of Π. For a type $\mathcal{F}$ of P systems, denote $\mathbf{PMC}_{\mathcal{F}_{\leq k}}$ the set of those problems that can be decided by such polynomially uniform families of P systems of type $\mathcal{F}$ which have k-bounded members only. Denote $\mathcal{AM}_{-e}$ those P systems with active membranes that do not employ membrane division rules. It can be seen using the generalization of the proof of $\mathbf{PMC}_{\mathcal{AM}_{-e}} \subseteq \mathbf{P}$ in [30] that $\mathbf{PMC}_{\mathcal{AM}_{\leq 2}} \subseteq \mathbf{P}$ also holds. Using the results obtained in the paper we can give the following new characterizations of $\mathbf{P}$.

Corollary 1. $\mathbf{P} = (\mathbf{L}, \mathbf{L}) - \mathbf{PMC}_{\mathcal{AM}_{+out}} = (\mathbf{L}, \mathbf{L}) - \mathbf{PMC}_{\mathcal{AM}_{+e,+d,\leq 2}} = (\mathbf{L}, \mathbf{L}) - \mathbf{PMC}_{\mathcal{AM}^0_{+evo(1),+d}}$.

4 Conclusions

In this paper we have shown that uniform families of the following restricted variants of P systems with active membranes can solve all problems in $\mathbf{P}$: (1) P systems where only out communication rules are used, (2) P systems where only elementary membrane division and dissolution rules are used, and (3) polarizationless P systems where only dissolution and 1-restricted evolution rules are used. Using the obtained results concerning variants (1) and (3), and known results about the upper bound on the power of these variants we could give new characterizations of $\mathbf{P}$ in terms of Membrane Computing techniques.

It remained an open question if the variant (2) could solve problems outside of **P**. It is known that without polarizations of the membranes this is not possible [29]. It is also an open question if these systems can solve all problems in **P** when polarizations of the membranes are not allowed. Nevertheless, we could give another characterization of **P** using variant (2) when we made a simple semantic restriction on the computations of this variant.

References

1. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-Complete Problem by P Systems with Restricted Active Membranes. Fundamenta Informaticae **58** (2003) 67–77
2. Alhazov, A., Pan, L., Păun, G.: Trading polarizations for labels in P systems with active membranes. Acta Inf. **41**(2-3) (2004) 111–144
3. Gazdag, Z,: Solving SAT by P Systems with Active Membranes in Linear Time in the Number of Variables. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing: 14th International Conference, LNCS vol. 8340 (2014) 189–205
4. Gazdag, Z., Kolonits, G.: A new approach for solving SAT by P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) Membrane Computing: 13th International Conference, LNCS vol. 7762 (2013) 195–207
5. Gazdag, Z., Gutiérrez-Naranjo, M.A.: Solving the ST-connectivity problem with pure membrane computing techniques. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing: 15th International Conference, LNCS vol. 8961 (2014) 215–228
6. 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, LNCS vol. 8961 (2014) 229–240
7. Gensler, H.J.: Introduction to Logic, Routledge, London (2002)
8. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: On the Power of Dissolution in P Systems with Active Membranes. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing: 6th International Workshop, LNCS vol. 3850 (2006) 224–240
9. Kolonits, G.: A Solution of Horn-SAT with P Systems Using Antimatter. In: Membrane Computing: 16th International Conference, LNCS vol. 9504 (2015) 236–250
10. Krishna, S.N., Rama, R.: A variant of P systems with active membranes: Solving NP-complete problems. Romanian J. of Information Science and Technology, 2, 4 (1999) 357–367
11. Murphy, N.: Uniformity conditions for membrane systems: uncovering complexity below P. Ph.D. thesis, National University of Ireland, Maynooth (2010)
12. Murphy, N., Woods, D.: Active Membrane Systems Without Charges and Using Only Symmetric Elementary Division Characterise P. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing: 8th International Workshop, LNCS vol. 4860 (2007) 367–384

13. Murphy, N., Woods, D.: A Characterisation of NL Using Membrane Systems without Charges and Dissolution. In: Calude, C.S. et al (eds.) Unconventional Computing: 7th International Conference, LNCS vol. 5204 (2008) 164–176
14. Murphy, N., Woods, D.: On acceptance conditions for membrane systems: characterisations of **L** and **NL**. In Neary, T., Woods, D., Seda, T., Murphy, N., eds.: Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, Volume 1 of Electronic Proceedings in Theoretical Computer Science., Open Publishing Association (2009) 172–184
15. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing **10**(1) (2011) 613–632
16. Murphy, N., Woods, D.: Uniformity is weaker than semi-uniformity for some membrane systems. Fundam. Inf. **134**(1-2) (2014) 129–152
17. Pan, L., Alhazov, A., Isdorj, T.-O.: Further remarks on P systems with active membranes, separation, merging, and release rules. Soft Computing **9**(9) (2004) 686–690
18. Pan, L., Ishdorj, T.-O.: P systems with active membranes and separation rules. Journal of Universal Computer Science 10(5) (2004) 630649
19. Păun, Gh.: P Systems with Active Membranes: Attacking NP-Complete Problems. Journal of Automata, Languages and Combinatorics **6**(1) (2001) 75–90
20. Păun, Gh.: Further twenty six open problems in membrane computing. In: Third Brainstorming Week on Membrane Computing. Fénix Editora, Sevilla (2005) 249–262
21. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford, England (2010)
22. Pérez-Jiménez, M.J., Romero-Campero, F.J.: Trading Polarization for Bi-stable Catalysts in P Systems with Active Membranes. In: Mauri, G., Păun, G., Prez-Jimnez, M.J., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing: 5th International Workshop, LNCS vol. 3365 (2005) 373–388
23. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. In: Csuhaj-Varjú, E., Kintala, C., Wotschke, D., Vaszil, G. (eds.) Proceeding of the 5th Workshop on Descriptional Complexity of Formal Systems. DCFS 2003 (2003) 284–294
24. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing **2**(3) (2003) 265–285
25. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. Journal of Automata, Languages and Combinatorics **11**(4) (2006) 423–434
26. Salomaa, A.: Formal Languages. Academic Press, New York, London (1973)
27. Sipser, M.: Introduction to the Theory of Computation. 3rd edn. Cengage Learning (2012)
28. Sosík, P.: The computational power of cell division in P systems. Nat. Comput. **2**(3) (2003) 287–298
29. Woods, D., Murphy, N., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Membrane Dissolution and Division in P. In: Calude, C.S., da Costa, J.F.G., Dershowitz, N., Freire, E., Rozenberg, G. (eds.) Unconventional Computation: 8th International Conference, LNCS vol. 5715 (2009) 262–276
30. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-Complete Problems Using P Systems with Active Membranes. In: Unconventional Models of Computation, UMC2K: Proceedings of the Second International Conference on Unconventional Models of Computation. Springer London, London (2001) 289–301

Kernel P Systems Modelling, Testing and Verification

Marian Gheorghe[1], Rodica Ceterchi[2], Florentin Ipate[2,3] and Savas Konur[1]

[1] School of Electrical Engineering and Computer Science, University of Bradford
Bradford BD7 1DP, UK
{m.gheorghe, s.konur}@bradford.ac.uk
[2] Department of Computer Science, University of Bucharest
Str. Academiei nr. 14, 010014, Bucharest, Romania
florentin.ipate@ifsoft.ro, rceterchi@gmail.com
[3] Department of Computer Science, University of Piteşti
Str Targul din Vale, nr 1, 110040, Argeş, Romania

Summary. A kernel P system (kP system, for short) integrates in a coherent and elegant manner many of the P system features most successfully used for modelling various applications and, consequently, it provides a framework for analyzing these models. In this paper, we illustrate the modeling capabilities of kernel P systems by showing how other classes of P systems can be represented with this formalism and providing a number of kP system models for sorting algorithms. Furthermore, the problem of testing systems modelled as kP systems is also discussed and a test generation method based on automata is proposed. We also demonstrate how formal verification can be used to validate that the given models work as desired.

1 Introduction

Membrane systems were introduced in [27] as a new natural computing paradigm inspired by the structure and distribution of the compartments of living cells, as well as by the main bio-chemical interactions occurring within compartments and at the inter-cellular level. They were later also called *P systems*. An account of the basic fundamental results can be found in [28] and a comprehensive description of the main research developments in this area is provided in [29]. The key challenges of the membrane systems area and a discussion on some future research directions, are available in a more recent survey paper [20].

In recent years, significant progress has been made in using P systems to model and simulate systems and problems from various areas. However, in order to facilitate the modelling, in many cases various features have been added in an ad-hoc manner to these classes of P systems. This has led to a multitude of P systems variants, without a coherent integrating view. The newly introduced concept of

kernel P systems (kP systems) [16, 17] provides a response to this problem. A kP system integrates in a coherent and elegant manner many of the P system features most successfully used for modelling various applications and, consequently, it provides a framework for analyzing these models. Furthermore, the expressive power of these systems has been illustrated by a number of representative case studies [19, 17]. The kP system model is supported by a modelling language, called kP-Lingua, capable of mapping a kP system specification into a machine readable representation. Furthermore, kP systems are supported by a software framework, kPWorkbench [21], which integrates a set of related simulation and verification tools and techniques.

Another complementary method to simulation and verification is testing, a major activity in the lifecycle of software systems. In practice, software products are almost always validated through testing. Testing has been discussed for cell-like P systems and various strategies, such as rule coverage based and automata based techniques, have been proposed [15, 24]. Until now, however, testing has not been discussed in the context of kP systems.

In this paper we further illustrate the modeling capabilities of kernel P systems by showing that other classes of P systems can be represented with this formalisms and by providing a number of kP system models for sorting algorithms. We present in this paper the relationship between kP systems and active membrane systems with electrical charges, whereas in [16, 17, 18] we have also investigated the relationship with neural-like P systems. We also study here the relationship between kP systems and P systems with symport/antiport rules. Furthermore, the problem of testing systems modelled as kP systems is also discussed and a test generation method based on automata is proposed. We also demonstrate how formal verification can be used to validate that the given models work as desired.

2 kP Systems - Main Concepts and Definitions

We consider that standard P system concepts such as strings, multisets, rewriting rules, and computation are well-known and refer to [28] for their formal notations and precise definitions. The kP system concepts and definitions introduced below are from [16, 17]; some are slightly changed and this will be mentioned.

Definition 1. *T is a* set of compartment types, *$T = \{t_1, \ldots, t_s\}$, where $t_i = (R_i, \sigma_i)$, $1 \leq i \leq s$, consists of a set of rules, R_i, and an execution strategy, σ_i, defined over $Lab(R_i)$, the labels of the rules of R_i.*

Remark 1. The compartments that appear in the definition of the kP systems will be instantiated from these compartment types. The types of rules and the execution strategies will be discussed later.

Definition 2. *A* kernel P (kP) system *of degree n is a tuple*

$$k\Pi = (A, \mu, C_1, \ldots, C_n, i_0),$$

where A is a finite set of elements called objects*; μ defines the* initial membrane structure*, which is a graph, (V, E), where V are vertices indicating components, and E edges; $C_i = (t_i, w_i)$, $1 \leq i \leq n$, is a* compartment *of the system consisting of a compartment type from T and an* initial multiset*, w_i over A; i_o is the* output compartment *where the result is obtained.*

2.1 kP System Rules

The discussion below assumes that the rules we refer to belong to the same compartment, C_i.

Each rule r may have a **guard** g which refers to the multiset where the rule is applied to. Its generic form is $r\ \{g\}$. The rule r is applicable to a multiset w when its left hand side is contained into w and g is true for w.

The guards are constructed using multisets over A, as operands, and relational and Boolean operators. Let us first introduce some notations.

For a multiset w over A and an element $a \in A$, we denote by $|w|_a$ the number of objects a occurring in w. Let us denote $Rel = \{<, \leq, =, \neq, \geq, >\}$, the set of relational operators, $\gamma \in Rel$, a relational operator, a^n a multiset and $r\ \{g\}$ a rule with guard g. We first introduce an *abstract relational expression* which is evaluated for any multiset where the rule is applied to.

Definition 3. *If g is the* abstract relational expression γa^n *and w is the multiset it refers to, then the guard denotes the* relational expression $|w|_a \gamma n$*. The guard g is true for the multiset w if $|w|_a \gamma n$ is true.*

One can consider the Boolean operators $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction), listed with respect to the decreasing precedence order. *Abstract Boolean expressions* are obtained by connecting abstract relational expressions by Boolean operators.

Definition 4. *If g is the* abstract Boolean expression *and the current multiset is w, then the guard denotes the* Boolean expression *for w, obtained by replacing abstract relational expressions with relational expressions for w. The guard g is true for the multiset w when the Boolean expression for w is true.*

Definition 5. *A guard is: (i) one of the Boolean constants true or $false$; (ii) an abstract relational expression; or (iii) an abstract Boolean expression.*

Example 1. If g is the guard $\geq a^5 \wedge \geq b^3 \vee \neg > c$ and w a multiset it refers to, then g is true in w if it has at least 5 a's and 3 b's or no more than one c.

Definition 6. *A rule from a compartment $C_{l_i} = (t_{l_i}, w_{l_i})$ can have one of the following types:*

- *(a)* **rewriting and communication** *rule: $x \to y\ \{g\}$,*
  where $x \in A^+$ and y has the form $y = (a_1, t_1) \ldots (a_h, t_h)$, $h \geq 0$, $a_j \in A$ and t_j indicates a compartment type from T – see Definition 2 – with instance

compartments linked to the current compartment; t_j might also indicate the type of the current compartment, t_{l_i}, (in this case it is not present on the right hand side of the rule); if a link does not exist (i.e., there is no link between the two compartments in E) then the rule is not applied; if a target, t_j, refers to a compartment type that has more than one instance connected to C_{l_i}, then one of them will be non-deterministically chosen;

- *(b)* **structure changing rules***; the following types of rules are considered:*
  - *(b1)* **membrane division** *rule:* $[x]_{t_{l_i}} \to [y_1]_{t_{i_1}} \dots [y_p]_{t_{i_p}}$ $\{g\}$,
    where $x \in A^+$ and $y_j \in A^*$; the compartment C_{l_i} will be replaced by p compartments; the j-th compartment, instantiated from the compartment type t_{i_j} contains the same objects as C_{l_i}, but x, which will be replaced by y_j; all the links of C_{l_i} are inherited by each of the newly created compartments;
  - *(b2)* **membrane dissolution** *rule:* $[]_{t_{l_i}} \to \lambda$ $\{g\}$*;*
    the compartment C_{l_i} will be destroyed together with its links;
  - *(b3)* **link creation** *rule:* $[x]_{t_{l_i}}; []_{t_{l_j}} \to [y]_{t_{l_i}} - []_{t_{l_j}}$ $\{g\}$*;*
    the current compartment is linked to a compartment of type t_{l_j} and x is transformed into y; if more than one instance of the compartment type t_{l_j} exists then one of them will be non-deterministically picked up; g is a guard that refers to the compartment instantiated from the compartment type t_{l_1};
  - *(b4)* **link destruction** *rule:* $[x]_{t_{l_i}} - []_{t_{l_j}} \to [y]_{t_{l_i}}; []_{t_{l_j}}$ $\{g\}$*;*
    is the opposite of link creation and means that the compartments are disconnected.

The membrane division is defined slightly differently here compared to [16, 17]. Currently, the right hand side of the rule uses simple multisets with no target compartments, as they were initially introduced in [16, 17].

2.2 kP System Execution Strategies

In kP systems the way in which rules are executed is defined for each compartment type t from T – see Definition 1 and Remark 1. As in Definition 1, $Lab(R)$ is the set of labels of the rules R.

Definition 7. *For a compartment type $t = (R, \sigma)$ from T and $r \in Lab(R)$, $r_1, \dots, r_s \in Lab(R)$, the* execution strategy*, σ, is defined by the following*

- *$\sigma = \lambda$, means no rule from the current compartment will be executed;*
- *$\sigma = \{r\}$ – the rule r is executed;*
- *$\sigma = \{r_1, \dots, r_s\}$ – one of the rules labelled $r_1, \dots, r_s$ will be chosen non-deterministically and executed; if none is applicable then none is executed; this is called* alternative *or* choice*;*
- *$\sigma = \{r_1, \dots, r_s\}^*$ – the rules are applied an arbitrary number of times (*arbitrary parallelism*);*
- *$\sigma = \{r_1, \dots, r_s\}^\top$ – the rules are executed according to* maximal parallelism *strategy x;*

- $\sigma = \sigma_1 \& \ldots \& \sigma_s$, *means executing sequentially* $\sigma_1, \ldots, \sigma_s$, *where* σ_i, $1 \leq i \leq s$, *describes any of the above cases, namely* λ, *one rule, a choice, arbitrary parallelism or maximal parallelism; if one of* σ_i *fails to be executed then the rest is no longer executed;*
- *for any of the above* σ *strategy only one single structure changing rule is allowed.*

Arbitrary parallelism and maximal parallelism for rewriting and communication rules, as well as for structure changing rules (cell division, dissolution), are discussed in [29].

Remark 2. In certain cases the operator & will be ignored and the sequential execution will be denoted as $\sigma = \sigma_1 \ldots \sigma_s$.

Remark 3. A computation, as usual in membrane computing, is defined as a sequence of finite steps starting from the initial configuration, with the initial multisets distributed in compartments. In each step the rules are selected according to the execution strategy and this is given by the execution strategy in each compartment. The result of a computation will be the number of objects collected in the output compartment. For a kP systems $k\Pi$, the set of all these numbers will be denoted by $M(k\Pi)$.

Remark 4. When a terminal alphabet, F, is considered, the result of a computation will be the number of objects from F collected in the output compartment and this will be denoted by $M_t(k\Pi)$

3 kP Systems and Other Classes of P Systems

In this section we will investigate the relationship between kP systems and P systems with active membranes, but other relevant classes of P systems will be also considered, especially those with various applications, such as symport/antiport P systems. In [17, 18] neural-like P systems have been also considered.

3.1 P Systems with Active Membranes versus kP Systems

We study how P systems with active membranes are simulated by kP systems. In this case we are dealing with a cell-like system, so the underlying structure is a tree and we have a set of labels (types) for the compartments of the system. The way the relationship between these P systems is presented in the sequel is a natural extension of the method proposed in [16, 17, 18]. In the previous investigations the set of objects from the output compartment has been mixed up with the rest of the objects of the system. In this investigation we separate the objects corresponding to the output compartment and provide a more consistent notation for the kP system involved. We also deal in this investigation with active membrane systems with an upper bound for the number of active components

Definition 8. *A P system with active membranes of initial degree n is a tuple (see [29], Chapter 11)* $\Pi = (O, H, \mu, w_{1,0}, \ldots, w_{n,0}, R, i_0)$ *where:*

- O *is an alphabet of objects,* $w_{1,0}, \ldots, w_{n,0}$ *are the initial strings in the n initial compartments and* i_0 *is the output compartment;*
- H *is the set of labels for compartments;*
- μ *defines the tree structure associated with the system;*
- R *consists of rules of the following types:*
  - *(a)* rewriting rules*:* $[u \to v]_h^e$*, for* $h \in H$*,* $e \in \{+, -, 0\}$ *(set of electrical charges),* $u \in O^+$*,* $v \in O^*$*;*
  - *(b)* in communication rules*:* $u[]_h^{e_1} \to [v]_h^{e_2}$*, for* $h \in H$*,* $e_1, e_2 \in \{+, -, 0\}$*,* $u \in O^+$*,* $v \in O^*$*;*
  - *(c)* out communication rules*:* $[u]_h^{e_1} \to []_h^{e_2} v$*, for* $h \in H$*,* $e_1, e_2 \in \{+, -, 0\}$*,* $u \in O^+$*,* $v \in O^*$*;*
  - *(d)* dissolution rules*:* $[u]_h^e \to v$*, for* $h \in H \backslash \{s\}$*, s denotes the skin membrane (the outmost one),* $e \in \{+, -, 0\}$*,* $u \in O^+$*,* $v \in O^*$*;*
  - *(e)* division rules for elementary membranes*:* $[u]_h^{e_1} \to [v]_h^{e_2}[w]_h^{e_3}$*, for* $h \in H$*,* $e_1, e_2, e_3 \in \{+, -, 0\}$*,* $u \in O^+$*,* $v, w \in O^*$*;*

The rules are executed in accordance with the maxim parallelism, but in each compartment only one of the rules (b)-(e) is executed. In the sequel we assume that the output compartment is neither dissolved nor divided. The result of a computation, obtained in i_0 is denoted by $M(\Pi)$.

The following result shows how the computation of a P system with active membranes starting with n_1 compartments and an upper bound to the number of active compartments can be performed by a kP system using only rewriting and communication rules. A first idea of this result has been given in [16, 17, 18].

Theorem 1. *If* Π *is a P system with active membranes having* n_1 *initial compartments and an upper bound to the number of active compartments in any computation, then there exists a kP system,* $k\Pi$*, of degree* 2 *and using only rewriting and communication rules, such that* $M(\Pi) = M(k\Pi)$*.*

Proof. Let $\Pi = (O, H, \mu, w_{1,0}, \ldots, w_{n_1,0}, R, i_0)$ be a P system with active membranes of initial degree n_1. Initially, the polarizations of the n_1 compartments are all 0, i.e., $e_1 = \ldots = e_{n_1} = 0$.

We will build a kP system with two compartments. Compartment C_1 will capture the contents and rules of all the compartments of Π. The other compartment, C_2 will be associated to i_0 and this will collect the result.

We will need to keep track of a dynamic system of membranes, since we have dissolution and division of elementary membranes. We will identify a membrane by a pair (i, h) where $i \in I$ is an index associated with an instance of the membrane and $h \in H$ is its label. We use the index in addition to the label as the same label might appear several times in the system, especially after a membrane division rule has been applied. We work under the assumption that I is finite. Its cardinal is equal to the maximum number of active membranes that may appear in any

computation – this is assumed to have an upper limit. We let $i_0 \in I$ and $i_0 \in H$. We will denote by $(I \times H)_c$ the currently used pairs (i, h). We assume that for any $(i, h) \in (I \times H)_c$ and $(j, h') \in (I \times H)_c$, we have $i \neq j$. This way we make sure that the cardinal of $(I \times H)_c$ is always at most the cardinal of I. Whenever a membrane dissolution takes place, its index and label are removed from $(I \times H)_c$. When a membrane division rule is applied the index and label of the divided compartment are removed from $(I \times H)_c$ and two new values of indices with the same label are selected and added to the set $(I \times H)_c$. The tuple (i_0, i_0) is always in $(I \times H)_c$

We will codify a compartment $[w]_h^e$ by two tuples $< e, i, h >$ and $< w, i, h >$, with $(i, h) \in (I \times H)_c$, and where, for a multiset $w = a_1 \dots a_m$, $< w, i, h >$ denotes $< a_1, i, h > \dots < a_m, i, h >$. These tuples appear in C_1. When $h = i_0$ then in addition to the tuples present in C_1, in C_2, for $[w]_{i_0}^e$ we have e and w. For a compartment with label h and electrical charge e in Π there is only one tuple $< e, i, h >$ in C_1, when $h \neq i_0$, or an e in C_2, otherwise.

By $p(i, h)$ we denote the parent of the membrane with label h and of index i. If $p(i, h) = (i', h')$ it means that the membrane with label h' and index i' is the parent of the membrane with label h and index i. By $< x, p(i, h) >$ and $< e, p(i, h) >$ we denote the tuples $< x, i', h' >$ and $< e, i', h' >$, respectively.

A new symbol, δ, will be used for the membrane dissolution and division to control the transfer of objects after these rules have been applied. Hence, we will use the guard

$$\overline{= \delta_{all}} := \bigwedge (\neg =< \delta, i, h > |\ i \in I, h \in H).$$

We also introduce a guard checking that the symbols γ_1 ans γ_2, related with the communication with the output compartment, i_0, do not appear in the current multiset:

$$\overline{= \gamma_{all}} := (\neg = \gamma_1) \wedge (\neg = \gamma_2).$$

We construct $k\Pi$ using $T = \{t_1, t_2\}$, where $t_j = (R'_j, \sigma_j)$ (where R'_j and σ_j will be defined later), $1 \leq j \leq 2$, as follows: $k\Pi = (A, \mu', C_1, C_2, 2)$, where the elements of the system are given below.

- μ' is the graph with nodes C_1, C_2 and the edge linking them;
- The alphabet is

$$A = O \cup \{0, 0', +, +', -, -', \gamma_1, \gamma_2\} \\ \cup (\bigcup_{(i,h) \in I \times H} (\{< a, i, h > |\ a \in O \cup \{\delta\}\} \cup \{< e, i, h > |\ e \in \{0, +, -\}\}))$$

- $C_j = (t_j, w'_{j,0} w''_{j,0})$, $1 \leq j \leq 2$ and C_2 is the output compartment.
  - The initial multiset, $w'_{1,0} w''_{1,0}$, is given by

$$w'_{1,0} =< w_{1,0}, 1, h_1 > \dots < w_{n_1,0}, n_1, h_{n_1} > \overline{w_{i_0,0}}$$

where $\overline{w_{i_0,0}}$ means that $w_{i_0,0}$ does not appear in the initial multiset of C_1 (it will appear in C_2).

$$w''_{1,0} = \{< e_1, 1, h_1 > \dots < e_{n_1}, n_1, h_{n_1} > \overline{e_{i_0}}$$

where $e_1 = \ldots = e_{n_1} = 0$, for all the initial multisets and initial membranes of Π, and, similar to the above case, $\overline{e_{i_0}}$ means that e_{i_0} does not appear in the initial multiset. The initial multiset $w'_{2,0}w''_{2,0}$, is given by

$$w'_{2,0} = w_{i_0,0}, w''_{2,0} = e_{i_0}.$$

Initially, the indices $(I \times H)_1 = \{(1,h_1)\ldots(n_1,h_{n_1})\} \setminus \{(i_0,i_0)\}$ are used in association with compartment C_1 and (i_0,i_0) for C_2. The currently used indices are $(I \times H)_c = (I \times H)_1 \cup \{(i_0,i_0)\}$.

– R'_1 and R'_2 contain the rules below.

(a.1) For each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$ and each rule $[u \to v]^e_h \in R$, $e \in \{+,-,0\}$, we add to R'_1 the rule $< u,i,h > \to < v,i,h >$ $\{=< e,i,h > \wedge \overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$; these rules are applied only when the polarization e appears in the compartment with index i and label h and none of the (δ,j,h'), γ_1, γ_2 appears, i.e., no dissolution or division has started and no communication with the output compartment, i_0, takes place – see below.

(a.2) For $(i,h) = (i_0,i_0)$, we add to R'_1 the rule $< u,i_0,i_0 > \to < v,i_0,i_0 >$ $\{=< e,i_0,i_0 > \wedge \overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$ and the rule $u \to v$ $\{= e \wedge \overline{= \gamma_{all}}\}$ to R'_2.

(b.1) For each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$, such that $p(i,h) \neq (i_0,i_0)$, and each rule $u[]^{e_1}_h \to [v]^{e_2}_h \in R$, $e_1,e_2 \in \{+,-,0\}$, we add to R'_1 the rule $< u,p(i,h) >< e_1,i,h > \to < v,i,h >< e_2,i,h >$ $\{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$; these rules will transform $< u,p(i,h) >$ corresponding to u from the parent compartment to $< v,i,h >$ corresponding to v from the compartment with index i and label h; the polarization is changed; as there is only one object $< e_1,i,h >$, it follows that only one single rule corresponding to the compartment can be applied at any moment of the computation.

(b.2) When $(i,h) = (i_0,i_0)$, then the rules added to R'_1 are $< u,p(i_0,i_0) >< e_1,i_0,i_0 > \to < v,i_0,i_0 >< e_2,i_0,i_0 > (ve'_2\gamma_1,2)\gamma_1$ $\{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$ and $\gamma_1 \to \lambda$; and the rules added to R'_2 are $e'_2 \to e_2\{= \gamma_1\}$ and $\gamma_1 e \to \lambda$, $e \in \{0,+,-\}$. The first rule apart from simulating the communication rule, also introduces γ_1 in both compartments. In C_2 it helps changing the polarization of it and in C_1 it helps with the synchronisation of the computation. Then the symbol disappears.

(b.3) When $p(i,h) = (i_0,i_0)$, then we add to R'_1 the rules $< u,i_0,i_0 >< e_1,i,h > \to < v,i,h >< e_2,i,h > (\gamma_2,2)\gamma_2$ $\{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$ and $\gamma_2 \to \lambda$. The rule $u\gamma_2 \to \lambda$ is added to R'_2. Similar to $(b.2)$, γ_2 is introduced in both compartments and in C_2 it helps removing u.

(c.1) For each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$, such that $p(i,h) \neq (i_0,i_0)$, and each rule $[u]^{e_1}_h \to []^{e_2}_h v \in R$, $e_1,e_2 \in \{+,-,0\}$, we add the rule $< u,i,h >< e_1,i,h > \to < v,p(i,h) >< e_2,i,h >$ $\{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$.

(c.2) When $(i,h) = (i_0,i_0)$, then we add to R'_1 the rule $< u,i_0,i_0 >< e_1,i_0,i_0 > \to < v,p(i_0,i_0) >< e_2,i_0,i_0 > (e'_2\gamma_1,2)\gamma_1$ $\{\overline{= \delta_{all}} \wedge \overline{= \gamma_{all}}\}$. As in $(b.2)$, we use $\gamma_1 \to \lambda$ in R'_1 and $e'_2 \to e_2\{= \gamma_1\}$ in R'_2. We need to

add to R'_2 the rule $u\gamma_1 e \to \lambda$. The rules make sure that in C_1 we simulate the communication rule and in C_2 u disappears and the polarization is changed to e_2.

(c.3) When $p(i,h) = (i_0, i_0)$, then the rule added to R'_1 is $< u,i,h >< e_1,i,h > \to < v,i_0,i_0 >< e_2,i,h > (v,2)\ \{\overline{= \delta_{all}}\ \wedge \overline{= \gamma_{all}}\}$. This rule simulates the communication rule and introduces v into C_2.

(d.1) for each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$, such that $p(i,h) \neq (i_0,i_0)$, and each rule $[u]^e_h \to v \in R$, $e \in \{+,-,0\}$, we add to R'_1 the rule $< u,i,h >< e,i,h > \to < v,p(i,h) >< \delta,i,h >\ \{\overline{= \delta_{all}}\ \wedge \overline{= \gamma_{all}}\}$; all the objects corresponding to those from the compartment of index i and label h must be moved to the parent compartment - this will happen in the presence of (δ,i,h) when no other transformation will take place; this is obtained by using in R'_1 rules $< a,i,h > \to < a,p(i,h) >\ \{=< \delta,i,h >\}$, $a \in O$ and $< \delta,i,h > \to \lambda$; the set $(I \times H)_c$ will change now by removing the pair (i,h) from it.

(d.2) When $p(i,h) = (i_0,i_0)$, then the rules above will become $< u,i,h >< e,i,h > \to < v,i_0,i_0 >< \delta,i,h > (v,2)\ \{\overline{= \delta_{all}}\ \wedge \overline{= \gamma_{all}}\}$ and $< a,i,h > \to < a,i_0,i_0 > (a,2)\ \{= (\delta,i,h)\}$, $a \in O$.

(e) For each $(i,h) \in I \times H \setminus \{(i_0,i_0)\}$ and each rule $[u]^{e_1}_h \to [v]^{e_2}_h [w]^{e_3}_h \in R$, $e_1,e_2,e_3 \in \{+,-,0\}$; we add to R'_1 the rule $< u,i,h >< e_1,i,h > \to < v,j_1,h >< e_2,j_1,h >< w,j_2,h >< e_3,j_2,h >< \delta,i,h >\ \{\overline{= \delta_{all}}\ \wedge \overline{= \gamma_{all}}\}$ – the pair (i,h) is removed from $(I \times H)_c$ and two new pairs (j_1,h) and (j_2,h), existing in $I \times H$, with $j_1 \neq j_2$, are added to $(I \times H)_c$ and one $< u,i,h >$ is transformed into $< v,j_1,h >$ and $< w,j_2,h >$ and their associated electrical charges; then the content corresponding to compartment of index i and label h will be moved to those of index j_1 and j_2 and the same label h, hence rules $< a,i,h > \to < a,j_1,h >< a,j_2,h >\ \{=< \delta,i,h >\}$, $a \in O$ are added to R'_1; finally, $< \delta,i,h > \to \lambda$ is also included in the set of rules of C_1; it is clear that only one division rule for the same compartment is applied in any step of the computation.

We note that in C'_2 there are no rules for dissolution and division as the output compartment is not afected by these rules.

The execution strategy in both compartments, C_1 and C_2 is maximal parallelism.

For a sequence of rules applied in Π, we have a corresponding sequence of rules in $k\Pi$. Obviously the objects obtained in the output compartment of Π are the same with those obtained in C_2 of $k\Pi$.

3.2 P Systems with Symport/Antiport versus kP Systems

The following definition is from [29].

Definition 9. *A P system (of degree $d \geq 1$) with* antiport *and/or* symport *rules is a construct*

$$\Pi = (O, F, E, \mu, w_{1,0}, \cdots, w_{d,0}, R_1, \cdots, R_d, i_0) \quad \textit{where}$$

*O is the alphabet of objects; $F \subseteq O$ is the alphabet of terminal objects; $E \subseteq O$ is the set of objects occurring in an unbounded number in the environment; μ is a membrane structure consisting of d membranes (usually labelled with i and represented by corresponding brackets $[_i$ and $]_i$, $1 \leq i \leq d$); w_i, $1 \leq i \leq d$, are strings over O associated with regions $1, \cdots, d$ of μ, representing the initial multisets of objects present in the regions of μ; R_i, $1 \leq i \leq d$, are finite sets of rules of the form $(u, out; v; in)$, with $u \neq \lambda$ and $v \neq \lambda$ (*antiport rule*) and/or (x, out) or (x, in), with $x \neq \lambda$ (*symport rules*); i_0, $1 \leq i_0 \leq d$, specifies the output membrane of Π.*

We will show now that one can construct for any symport/antiport P system a kernel P system, such that they compute the same result. We will adopt a slightly different way of computing the result of the kP systems by allowing it to use a set of terminal objects. In this case, according to Remark 4, the result will be given by the number of terminal objects from the output compartment. We can now state the main result of this section.

Theorem 2. *For any P system with symport/antiport rules, Π, there is a kP system, $k\Pi$, using only rewriting and communication rules and having a terminal set of objects, such that $M(\Pi) = M_t(k\Pi)$.*

Proof. Let $\Pi = (O, F, E, \mu, w_{1,0}, \cdots, w_{d,0}, R_1, \cdots, R_d, i_0)$ be a P system, of degree d, with symport and antiport rules as given by Definition 9.

We construct a kP system $k\Pi$ of degree one in the following manner. We take one unique compartment C_1. Apart from the d membranes in system Π, numbered by $1, 2, \cdots, d$, we think of the environment as a new membrane, with label 0.

The kP system we build is $k\Pi = (A, F', \mu', C_1, 1)$. The alphabet, A, of $k\Pi$ will consist of objects given by pairs $< x, i > \in O \times \{0, 1, \cdots, d\}$. For a multiset $w = a_1 \cdots a_m$ in membrane i we use the notation $< w, i >$ for $< a_1, i >, \cdots < a_m, i >$.

The initial multiset is

$$w'_{1,0} =< w_{1,0}, 1 > \cdots < w_{d,0}, d >$$

i.e., it contains all the pairs having the first element the initial multiset of membrane i and the second one i, $1 \leq i \leq d$. Initially, the environment associated with Π does not have any other objects apart from those in E. The set of rules, R'_1, of the kP system, includes the rules below.

- If a rule $(u, out; v, in)$, $u \neq \lambda$, $v \neq \lambda$, is in membrane i with parent j and $j \neq 0$, then we add the rule
  $< u, i >< v, j > \rightarrow < u, j >< v, i >$.
- If a rule $(u, out; v, in)$, $u \neq \lambda$, $v \neq \lambda$, is in membrane i with parent $j, j = 0$, then we decompose $u = u_1 u_2$ and $v = v_1 v_2$, such that $u_1, v_1 \in (O \setminus E)^*$ and $u_2, v_2 \in E^*$ and add the rule
  $< u, i >< v_1, 0 > \rightarrow < u_1, 0 >< v, i >$.

If $u_1 = \lambda$ or $v_1 = \lambda$ we interpret $< \lambda, 0 >$ as λ, i.e. for $v_1 = \lambda$ and $u_1 \neq \lambda$ the rule becomes $< u, i > \rightarrow < u_1, 0 >< v, i >$.

- If a rule (u, out), $u \neq \lambda$, is in membrane i with parent j and $j \neq 0$, then we add the rule
  $< u, i > \rightarrow < u, j >$.
- If a rule (u, out), , $u \neq \lambda$, is in membrane i with parent $j, j = 0$, we add the rule
  $< u, i > \rightarrow < u_1, 0 >$,
  where $u = u_1 u_2$ with $u_1 \in (O \setminus E)^*$ and $u_2 \in E^*$.
  If $u_1 = \lambda$, then again $< \lambda, 0 >$ is λ, and the rule becomes $< u, i > \rightarrow \lambda$.
- If a rule (v, in), $v \neq \lambda$, is in membrane i with parent j, and $j \neq 0$, then we add the rule
  $< v, j > \rightarrow < v, i >$.
- If a rule (v, in), $v \neq \lambda$, is in membrane i with parent $j, j = 0$, then we add the rule
  $< v_1, 0 > \rightarrow < v, i >$,
  where $v = v_1 v_2$ with $v_1 \in (O \setminus E)^+$ and $v_2 \in E^*$.
  Note that in this last case $v_1 \neq \lambda$.

Note that the environment (membrane 0) is treated differently by the above rules. We do not keep track of elements over E in the environment, which are in an unbounded number, but we must keep track of elements over $O \setminus E$ in the environment. If an u must go into the environment, then we decompose $u = u_1 u_2$ such that $u_1 \in (O \setminus E)^*$ and $u_2 \in E^*$, and only $< u_1, 0 >$ will appear in the right-hand side of the rule. Similarly, if a v comes from the environment, we have $v = v_1 v_2$ with $v_1 \in (O \setminus E)^+$ and $v_2 \in E^*$, and $< v_1, 0 >$ must be consumed by the rule.

The execution strategy of $k\Pi$ will be maximal parallelism.

The terminal alphabet is $F' = \{< a, i_0 > | \, a \in F\}$. Note that multisets over F' obtained in $k\Pi$ will correspond to multisets over F obtained in membrane i_0 by Π.

Remark 5. It remains an open problem to devise a kP system with two compartments, where C_1 reflects the functioning of the entire system, while C_2 simulates membrane i_0.

4 Sorting with kP Systems

Sorting is a central topic in Computer Science (see [25]). A variety of approaches to sorting have been investigated, for different algorithms, and with different P system models. A first approach was [3], in which a BeadSort algorithm was implemented with tissue P systems. Another approach was [6], in which algorithms inspired from sorting networks were implemented using P systems with communication. Other papers ([1], [30]) use different types of P systems, and refine the sorting problem

to sorting by ranking. A first overview of sorting algorithms implemented with P systems was [2]. A dynamic sorting algorithm was proposed in [7]. The bitonic sort was implemented with P systems [8], spiking neural P systems were used for sorting [10], other network algorithms were implemented using P systems [9]. Another overview of sorting algorithms implemented with P systems is provided by [11]. First implementations of sorting with kP systems were proposed in [16, 17].

The problem can be stated as follows: suppose we want to sort $x_1, \cdots, x_n$, $n \geq 1$, in ascending order, where x_i, $1 \leq i \leq n$, are positive integer values. Each such number, x_i, $1 \leq i \leq n$, will be represented as a multiset $a_i^{x_i}$, $1 \leq i \leq n$, where a_i is an object from a given set. In the next sections we will present two sorting algorithms using different representations of the sequence of positive integer numbers. More precisely, we start with an algorithm already studied in several other papers, [6, 2] for various types of P systems. Here we implement it using kP systems, by representing each element x_i by a^{x_i}, $1 \leq i \leq n$. The multisets a^{x_i}, $1 \leq i \leq n$, are stored in separate compartments, C_i, $1 \leq i \leq n$ (Section 4.1). In Section 4.2 these positive integer numbers are represented by $a_i^{x_i}$, $1 \leq i \leq n$, and stored in one compartment C_1; an additional one, C_2, is used for implementation purposes. In Section 4.3 it is used again the representation $a_i^{x_i}$, $1 \leq i \leq n$, but a more complex structure of compartments is provided in order to maximise the parallel behaviour of the system implementing the sorting algorithm. The algorithm used in Section 4.1 and Section 4.2 makes comparisons of adjacent compartments by employing a two stage process. In the first stage all pairs "odd-even" are compared (C_{2i-1} with C_{2i}, $i \geq 1$) and in the second stage all pairs "even-odd" are involved (C_{2i} with C_{2i+1}, $i \geq 1$).

4.1 Sorting Using kP Systems with an Element per Compartment

The approach presented below follows [16, 17], but stopping conditions have been also considered and the sequence of numbers is obtained in ascending order.

Let us consider a kP system, $k\Pi_1$, having n compartments $C_i = (t_i, w_{i,0})$, where $t_i = (R_i, \sigma_i)$, $1 \leq i \leq n$, and a set of objects $A = \{a, b, c, p, p'\}$. In each compartment, C_i, the initial multiset, $w_{i,0}$, $1 \leq i \leq n$, includes the representation of the positive integer number x_i, i.e., a^{x_i}, the multiset $c^{2(n-1)}$ and the object p for all odd index values, when n is an even number, and for all odd index values, but the last, when n is odd. The objects p stored initially in compartments indexed by odd values indicate that one starts with stage one, whereby "odd-even" compartment pairs are compared first. The multiset $c^{2(n-1)}$ will be used in a counting process, in each of the compartments, that will help stoping the algorithm when the sorting is complete.

Let us consider for $n = 6$ the sequence 3, 6, 9, 5, 7, 8. Then the initial multisets are:
$w_{1,0} = a^3c^{10}p; w_{2,0} = a^6c^{10}; w_{3,0} = a^9c^{10}p; w_{4,0} = a^5c^{10}; w_{5,0} = a^7c^{10}p; w_{6,0} = a^8c^{10}$. As n is even, p appears in all compartments indexed by odd values, i.e., C_1, C_3, and C_5.

In each compartment C_i, t_i contains the following set of rules, denoted R_i, $1 \leq i \leq n$,
$r_{1,i} : a \to (b, i+1)\ \{\geq p\}$, $i < n$;
$r_{2,i} : p \to p'$;
$r_{3,i} : p' \to (p, i+1)$, for i odd and $i < n$, and $r'_{3,i} : p' \to (p, i-1)$, for i even and $i > 1$;
$r_{4,i} : ab \to a(a, i-1)$, $i > 1$;
$r_{5,i} : b \to a$, $i > 1$.

We also consider the rule $r : c \to \lambda$. This rule is used for implementing the counting process mentioned above. By using the two stage process of comparing "odd-even" pair of compartments and then "even-odd" ones, one needs at most $n-1$ stages to complete the sorting. As it will be explained below, each stage will involve two steps and consequently after $2(n-1)$ steps one expects to stop the sorting process.

In each compartment C_i, the execution strategy is given by

$$\sigma_i = \{r\}\{r_{1,i}, r_{2,i}, r_{3,i}, r_{4,i}\}^{\top}\ \{r_{5,i}\}^{\top},$$

if i is odd; for even values of i, $r_{3,i}$ is replaced by $r'_{3,i}$. The execution strategy, σ_i, tells us that a sequence of three sets of rules are executed in each step. The first one indicates that one single rule is applied and then two sets of rules are used, each of them applied in a maximal parallel manner.

We assume that any two compartments, C_i, C_{i+1}, $1 \leq i < n$, are connected.

In the first step, of the "odd-even" stage, in every compartment one c is removed by applying $r : c \to \lambda$; then the only applicable rules are $r_{1,i}, r_{2,i}$ in all compartments indexed by an odd value. Given the presence of p in these compartments, rules $r_{1,i}$ move all objects a from each compartment with an odd index value, i, $i < n$, to the compartment C_{i+1} by transforming them into bs and rules $r_{2,i}$ transforming p into p'. In the next step, another c is removed from every compartment and rules $r_{3,i}, r_{4,i}, r_{5,i}$ are then applied. The rules $r_{3,i}$ are applied in compartments with an odd index value and $r_{4,i}$ are applied in compartments with an even index value, this means p' is moved as p from each C_i, i an odd value and $i < n$, to compartment C_{i+1} and every ab, in each C_j, j an even value and $j > 1$, is transformed into an a kept in the compartment and another a moved to C_{j-1}. At the end of the step, in each compartment C_j, j an even value and $j > 1$, and in accordance with the execution strategy, the remaining b objects, if any, are transformed into as. These two steps implement comparators between two adjacent compartments, in this case "odd-even" pairs. If a^{x_i} from C_i and $a^{x_{i+1}}$ from C_{i+1}, $i < n$, are such that $x_i > x_{i+1}$ then the multisets a^{x_i} is moved to C_{i+1} and $a^{x_{i+1}}$ to C_i. In the next step, the first of the second stage, ps appear in even compartments and the comparators are now acting between pairs of compartments C_i, C_{i+1}, where i is even and $i < n$.

Given that the algorithm must stop in maximum $2(n-1)$ steps, one can notice that in step $2(n-1)$ the counter, c, disappears, i.e., becomes λ, and the first rule from the execution strategy, r, is no longer applicable and then the next sets of

rules are not executed either. Hence, the process stops with the multisets codifying for positive integer values in asceding order.

The table below presents the first four steps of the sorting process.

Compartments - Step	C_1	C_2	C_3	C_4	C_5	C_6
0	$a^3c^{10}p$	a^6c^{10}	$a^9c^{10}p$	a^5c^{10}	$a^7c^{10}p$	a^8c^{10}
1	c^9p'	$a^6b^3c^9$	c^9p'	$a^5b^9c^9$	c^9p'	$a^8b^7c^9$
2	a^3c^8	a^6c^8p	a^5c^8	a^9c^8p	a^7c^8	a^8c^8p
3	a^3c^7	c^7p'	$a^5b^6c^7$	c^7p'	$a^7b^9c^7$	a^8c^7p'
4	a^3c^6p	a^5c^6	a^6c^6p	a^7c^6	a^9c^6p	a^8c^6

Now, one can state the result of the algorithm presented above and the number of steps involved.

Theorem 3. *The above algorithm sorts in ascending order a sequence of* $n, n \geq 1$, *positive integer numbers in* $2(n-1)$ *steps.*

4.2 Sorting Using kP Systems with Two Compartments

In this section we use a representation of the positive integer numbers $x_1, \cdots, x_n$ as multisets $a_1^{x_1}, \cdots, a_n^{x_n}$, where $a_1, \cdots, a_n$ are from a given set of objects. We consider a kP system, $k\Pi_2$, with two compartments $C_j = (t_j, w_{j,0})$, $1 \leq j \leq 2$, which are linked and $A = \{a_1, \ldots, a_n, c\}$. The initial multisets are $w_{1,0} = a_1^{x_1} \cdots a_n^{x_n} c^{n-1}$ and $w_{2,0} = c^{n-1}$.

Finally, the kP system $k\Pi_2$ will lead to a multiset $a_1^{x_{i_1}} \cdots a_n^{x_{i_n}}$ in compartment C_1, such that $x_{i_1} \leq \cdots \leq x_{i_n}$.

In compartments C_1 the rules are
$R_{1,1} = \{a_i a_{i+1} \to (a_i, 2)(a_{i+1}, 2) \mid 1 \leq i < n \ \& \ i = 1, 3, ...\}$;
$R_{2,1} = \{a_i \to (a_{i+1}, 2) \mid 1 \leq i < n \ \& \ i = 1, 3, ...\}$;
$R_{3,1} = \{a_i \to (a_i, 2) \mid 1 \leq i \leq n\}$.

We also consider the rule $r : c \to \lambda$, like in the previous section.

Compartment C_2 has the rules
$R_{1,2} = \{a_i a_{i+1} \to (a_i, 1)(a_{i+1}, 1) \mid 1 \leq i < n \ \& \ i = 2, 4, ...\}$;
$R_{2,2} = \{a_i \to (a_{i+1}, 1) \mid 1 \leq i < n \ \& \ i = 2, 4, ...\}$;
$R_{3,2} = \{a_i \to (a_i, 1) \mid 1 \leq i \leq n\}$;
and the rule r defined above.

The execution strategies of these compartments are

$$\sigma_j = \{r\} Lab(R_{1,j})^\top Lab(R_{2,j})^\top Lab(R_{3,j})^\top, \ j = 1, 2.$$

In compartment C_1 one implements "odd-even" comparison steps and in C_2 "even-odd" steps. The process starts with compartment C_1. The execution strategy in each compartment starts by decrementing the counter (using r), then the comparators are implemented by executing first $R_{1,j}$ and then $R_{2,j}$, $j = 1, 2$, both in maximally parallel manner. After that all the pairs a_i, a_{i+1} are sent to the other compartment and when $a_i^{x_i}$ and $a_{i+1}^{x_{i+1}}$ are such that $x_i > x_{i+1}$ then a_i is transformed into a_{i+1} and sent to the other compratment, i.e., a_i and a_{i+1} are swapped

and sent to the other compartment. In the last part, are moved to the other compartment all the objects a_i, $1 \leq i \leq n$, that remained there after comparisons. This is the case when a pair a_i and a_{i+1} has its objects with their multiplicities, x_i and x_{i+1}, respectively, in the right order, i.e., $x_i \leq x_{i+1}$.

Clearly after at most $n-1$ steps the objects $a_1, \cdots, a_n$ have their multiplicities in the ascending order and the sorting process stops as r is no longer applicable and the execution strategy is not applicable any more.

Theorem 4. *The above algorithm sorts in ascending order a sequence of $n, n \geq 1$, positive integer numbers in $n-1$ steps.*

One can produce a similar implementation whereby the comparison of two neighbours is made more directly and with simpler rules, but with more complex guards.

In this case we extend the definition of a guard, by allowing θa^n to be of the form $\theta a^{f(z)}$, where $f(z)$ is a function over the multisets of objects returning a positive integer value. For the current multiset z, one can define, for instance, $f_b(z) = |z|_b$, Then a rule $a \to b\{> a^{f_b(\cdot)}\}$ is applicable to z if the guard is true, i.e., $|z|_a > |z|_b$.

The *extended definition of the guard* allows us to implement a comparator with simpler rules than in the previous case. We have the pair of integers x_1, x_2 represented as ${a_1}^{x_1}, {a_2}^{x_2}$. Consider the pair of guarded rewriting rules

$$a_1 \to a_2\{> {a_1}^{f_{a_2}(\cdot)}\} \text{ and } a_2 \to a_1\{< {a_2}^{f_{a_1}(\cdot)}\}$$

where $f_{a_2}(w) = |w|_{a_2}$ and $f_{a_1}(w) = |w|_{a_1}$. Then both guards codify the condition $x_1 > x_2$.

If $x_1 \leq x_2$ the rules are not applicable, while if $x_1 > x_2$, then the x_1 copies of a_1 are rewritten as a_2, and x_2 copies of a_2 are rewritten as a_1, interchanging the values and achieving eventually $x_1 \leq x_2$.

A kP system, $k\Pi_3$, is defined now for sorting the sequence of $n, n \geq 1$, positive integer numbers. It consists of two compartment C_1 and C_2 which are linked. They have the same initial multisets like $k\Pi_2$. The sets of rules associated with these compartments are

- R_1 consisting of three subsets of rules (R_1 is responsible for "odd-even" stages):
  - $\{r \mid r : c \to \lambda\}$;
  - $R_{1,1} = \{a_i \to (a_{i+1}, 2)\{> a_i^{f_{a_{i+1}}(\cdot)}\} \mid i = 1, 3 \cdots \ \& \ i < n\}$;
  - $R_{2,1} = \{a_{i+1} \to (a_i, 2)\{< a_{i+1}^{f_{a_i}(\cdot)}\} \mid i = 1, 3 \cdots \ \& \ i < n\}$;
  - $R_{3,1} = \{a_i \to (a_i, 2) \mid i = 1, \cdots, n\}$.

The function f_{a_i} is defined $f_{a_i}(z) = |z|_{a_i}$, $1 \leq i \leq n$, for any multiset z.

Similarly, one defines R_2 in compartment C_2, which is used to implement the "even-odd" stage. The execution strategy is given by $\sigma_j = \{r\}Lab(R_{1,j} \cup R_{2,j})^\top Lab(R_{3,j})^\top, j = 1, 2$.

Theorem 5. *The above algorithm sorts in ascending order a sequence of* $n, n \geq 1$, *positive integer numbers in* $n-1$ *steps.*

Remark 6. 1. The kP system $k\Pi_3$ has simpler rules (non-cooperative) than $k\Pi_2$ (cooperative rules), but the guards of the rules in $k\Pi_2$ are simpler than those belonging to $k\Pi_3$.
2. The number of rules applied in each step to interchange $a_i^{x_i}$ and $a_{i+1}^{x_{i+1}}$ is $max\{x_i, x_{i+1}\}$ for $k\Pi_2$ and $x_i + x_{i+1}$ for $k\Pi_3$. Hence, $k\Pi_2$ uses less rules than $k\Pi_3$ in each one of the $n-1$ steps.

4.3 A kP System for Sorting in Constant Time

We suppose the integers to be sorted $x_1, \cdots x_n$ distinct.

We use a total of $n^2 + 2n$ compartments:

- $C_{i,j}$, $1 \leq i, j \leq n$, where each $C_{i,j}$ will be responsible for a comparison;
- C_i, $1 \leq i \leq 2n$, where each C_i, $1 \leq i \leq n$, will collect the results of comparing x_i to the rest; and C_i, $n+1 \leq i \leq 2n$, will collect the sorted result.

The connections between compartments are given by the set of edges

$$E = \cup_{i=1}^{n} E_i$$

where

$$E_i = \{(C_i, C_{i,j}) \mid 1 \leq j \leq n\} \cup \{(C_i, C_k) \mid n+1 \leq k \leq 2n\}, 1 \leq i \leq n.$$

Each $C_{i,j}$, $1 \leq i, j \leq n$, will contain the initial multiset $w_{i,j,0} = {a_i}^{x_i} {a_j}^{x_j} a$ and the rules
$r'_{i,j} : a_i \to a_j F\{> {a_i}^{f_j(\cdot)}\}$; $r''_{i,j} : a_j \to a_i\{< {a_j}^{f_i(\cdot)}\}$; $r'''_{i,j} : a \to a'$;
$r_{i,j} : a' \to (F, i)\{\geq F\}$,
where $f_i(z) = |z|_{a_i}$ and $f_j(z) = |z|_{a_j}$.
The execution strategy is $\sigma_{i,j} = \{r'_{i,j}, r''_{i,j}, r'''_{i,j}, r_{i,j}\}^{\top}$.

Note that the rules $r'_{i,j}, r''_{i,j}$ implement a comparator between x_i and x_j, similar to the one of the previous section. The modified comparator produces also a symbol F (False) when $x_i > x_j$, signifying that $x_i \leq x_j$ is false. If the rewriting rules $r'_{i,j}, r''_{i,j}$ and $r'''_{i,j}$ have acted, then a single F will be sent to compartment C_i (by using the rule $r_{i,j}$).

In compartment $C_i, 1 \leq i \leq n$, we have the initial multiset $w_{i,0} = {a_i}^{x_i} a$ and the rules
$r'_i : a \to a'$; $r''_i : a' \to a''$;
$r_{i,0} : a_i \to (a, n+1)\{< F \wedge = a''\}$; $r_{i,k} : a_i \to (a, n+k+1)\{= F^k \wedge = a''\}$, $1 \leq k \leq n-1$.
The execution strategy is $\sigma_i = \{r'_i, r''_i, r_{i,0}, \cdots, r_{i,n-1}\}^{\top}$.
Compartments $C_i, n+1 \leq i \leq 2n$, are initially empty and contain no rules.

The functioning of the system is as follows. Initially, in compartments $C_{i,j}$, $1 \leq i, j \leq n$, the rules $r'_{i,j}$, $r''_{i,j}$, and $r'''_{i,j}$ act. If $x_i > x_j$ the values will be interchanged

and some Fs will be produced (rules $r'_{i,j}$, $r''_{i,j}$ are used), signifying that $x_i \leq x_j$ is false. Also $r'''_{i,j}$ is used to transform a in a'. If at least one F is produced in $C_{i,j}$, then a single F will be sent to C_i, using rule $r_{i,j}$. In parallel, in each compartment C_i, $1 \leq i \leq n$, in the first two steps the rules r'_i and r''_i are applied.

After these two steps, no rules are applicable in $C_{i,j}$, $1 \leq i, j \leq n$, and in C_i, $1 \leq i \leq n$, the rules $r_{i,k}$, $0 \leq k \leq n-1$, might be applicable, depending on the number of Fs collected.The number of Fs tells us how many comparisons $x_i \leq x_j$, $1 \leq j \leq n$, are false. If we have k such Fs in C_i, it means that x_i is greater than exactly k other values, which means that in the sorted order it must be the $(k+1)$-th component. This is accomplished by sending a^{x_i} in C_{n+k+1}. The maximum number of Fs in C_i is $n-1$ because $C_{i,i}$ will never produce an F. If there are no Fs in C_i, this means that x_i is the minimum, and a^{x_i} will be sent to C_{n+1}. Compartments C_{n+i}, $1 \leq i \leq n$, collect the result of sorting. Each such C_{n+i} will contain at the end of the computation the string $a^{x_{k_i}}$, x_{k_i} being the i-th value in the sorted order. The computation has three steps, the first two ones in which $C_{i,j}$, $1 \leq i, j \leq n$, work, and a third one in which C_i, $1 \leq i \leq n$, work.

Theorem 6. *The above kP system sorts n integers in 3 steps.*

4.4 Sorting in Constant Time with Membrane Division

The algorithm in the previous section uses only rewriting and communication rules. This solution, although computationally efficient, requires an initial, quite complex, arrangement of compartments and multisets. We present here an algorithm which creates its working space by using membrane division rules.

We want to sort n distinct integers, $x_1, \cdots x_n$, represented as ${a_1}^{x_1}, \cdots, {a_n}^{x_n}$.

We start with a total of $3n$ compartments:

- C_i, $1 \leq i \leq n$, where each C_i, will collect the results of comparing x_i to the rest;
- C_i, $n+1 \leq i \leq 2n$, will collect the sorted result;
- C_k, $2n+1 \leq k \leq 3n$, such that $C_{2n+i} \subset C_i$, responsible for creating the comparator compartments.

The connections between compartments are given by the set of edges

$$E = \cup_{i=1}^{n} E_i$$

where

$$E_i = \{(C_i, C_{2n+i})\} \cup \{(C_i, C_k) \mid n+1 \leq k \leq 2n\}, 1 \leq i \leq n.$$

In compartment $C_i, 1 \leq i \leq n$, we have the initial multiset $w_{i,0} = {a_i}^{x_i} a$ and the rules

$r'_i : a \to a'$; $r''_i : a' \to a''$; $r'''_i : a'' \to a'''$;

$r_{i,0} : a_i \to (a, n+1)\{< F \wedge = a'''\}$; $r_{i,k} : a_i \to (a, n+k+1)\{= F^k \wedge = a'''\}$, $1 \leq k \leq n-1$.

The execution strategy is $\sigma_i = \{r'_i, r''_i, r'''_i, r_{i,0}, \cdots, r_{i,n-1}\}^{\top}$.

Compartments $C_i, n+1 \leq i \leq 2n$, are initially empty and contain no rules.

Compartments C_{2n+i}, $1 \leq i \leq n$, contain an initial multiset s, where s is a new object, and the membrane division rules

$$[s]_{2n+i} \rightarrow [{a_1}^{x_1} {a_i}^{x_i} a]_{i,1} \cdots [{a_j}^{x_j} {a_i}^{x_i} a]_{i,j} \cdots [{a_n}^{x_n} {a_i}^{x_i} a]_{i,n}, \;\; 1 \leq i \leq n.$$

These rules will generate in each C_i the compartments $C_{i,j}$, $1 \leq j \leq n$. In each $C_{i,j}$ we will have the multiset ${a_j}^{x_j} {a_i}^{x_i} a$, and the rules

$r'_{i,j} : a_i \rightarrow a_j F\{> {a_i}^{f_j(\cdot)}\}$; $r''_{i,j} : a_j \rightarrow a_i\{< {a_j}^{f_i(\cdot)}\}$; $r'''_{i,j} : a \rightarrow a'$;
$r_{i,j} : a' \rightarrow (F, i)\{\geq F\}$,

where $f_i(z) = |z|_{a_i}$ and $f_j(z) = |z|_{a_j}$.

The execution strategy is $\sigma_{i,j} = \{r'_{i,j}, r''_{i,j}, r'''_{i,j}, r_{i,j}\}^{\top}$.

Note that this is the comparator of the previous section, which sends a single F in C_i if $x_i \leq x_j$ is false.

During the first step, in compartment C_i rule r'_i is executed, while in C_{2n+i} the membrane division rule is applied, generating the $C_{i,j}$, $1 \leq j \leq n$. The next three steps are identical to the ones of the previous algorithm.

Theorem 7. *The above kP system sorts n integers in 4 steps.*

5 Simulating and Verifying kP Systems

In Section 4, we have illustrated that kP systems provide a coherent and expressive language that allow us to model various systems that were originally implemented by different P system variants. In addition to the modelling aspect, there has been a significant progress on analysing kP systems using various simulation and verification methodologies. The methods and tools developed in this respect have been integrated into a software platform, called kPWorkbench, to support the modelling and analysis of kP systems.

The ability of simulating kernel P systems is an important feature of this tool. Currently, there are two different simulation approaches, kPWorkbench Simulator and Flame (Flexible Large-Scale Agent Modelling Environment). Both simulators receive as input a kP system model written in kP–Lingua and outputs a trace of the execution, which is mainly used for checking the evolution of a system and for extracting various results out of the simulation. The simulators provide traces of execution for a kP system model, and an interface displaying the current configuration (the content of each compartment) at each step. It is useful for checking the temporal evolution of a kP system and for inferring various information from the simulation results.

Another important analysis method that kPWorkbench features is formal verification, requiring an exhaustive analysis of system models against some queries to be verified. The automatic verification of kP systems brings in some challenges as they feature a dynamic structure by preserving the structure changing rules such as membrane division, dissolution and link creation/destruction. kPWorkbench

Prop.	Pattern	(i) Informal query, (ii) Formal query using patterns
1	Existence	**(i)** *The numbers will be eventually sorted, i.e. the multisets representing the numbers will be in ascending order in the compartments*
		(ii) eventually (c_1.a <= c_2.a & c_2.a <= c_3.a & c_3.a <= c_4.a & c_4.a <= c_5.a & c_5.a <= c_6.a)
2	Universality	**(i)** *Counters in different compartments are always sync'ed*
		(ii) always (c_1.c = c_2.c & c_2.c = c_3.c & c_3.c = c_4.c & c_4.c = c_5.c & c_5.c = c_6.c)
3	Steady-state	**(i)** *In the state-state, the numbers are sorted*
		(ii) steady-state (c_1.a <= c_2.a & c_2.a <= c_3.a & c_3.a <= c_4.a & c_4.a <= c_5.a & c_5.a <= c_6.a)
4	Existence	**(i)** *The algorithm will eventually stop*
		(ii) eventually (c_i.c = 0)
5	Response	**(i)** *An unsorted state of two adjacent compartments will always be followed by a sorted one*
		(ii) (c_i.a >c_{i+1}.a) **followed-by** (c_i.a <= c_{i+1}.a)

Table 1: List of properties derived from the property language and their representations in different formats.

employs different verification strategies to allieviate these issues. The framework supports both *Linear Temporal Logic (LTL)* and *Computation Tree Logic (CTL)* properties by making use of the Spin [22] and NuSMV [13] model checkers.

In order to facilitate the formal specification, kPWorkbench features a property language, called *kP-Queries*, comprising a list of natural language statements representing formal property patterns, from which the formal syntax of the Spin and NuSMV formulas are automatically generated. The property language editor interacts with the kP-Lingua model in question and allows users to directly access the native elements in the model, which results in less verbose and shorter state expressions, and hence more comprehensible formulas. *kP-Queries* also features a grammar for the most common property patterns. These features and the natural language like syntax of the language make the property construction much easier.

Some of the commonly used patterns are "next", "existence", "absence", "universality", "recurrence", "steady-state", "until", "response" and "precedence". The details can be found in [21].

We now illustrate the usage of the query patters on the sorting algorithm given in Section 4.1. The other algorithms can be considered in a similar manner. In order to verify that the algorithm works as desired, we have constructed a set of properties specified in kP-Queries, listed in Table 1. The applied pattern types are given in the second column of the table. For each property we provide the following information; **(i)** informal description of each kP-Query, and **(ii)** the formal kP-Query using the patterns. The queries given in Table 1 capture that the algorithm given in Section 4.1 works as desired.

We note that both kP–Lingua model and the queries are automatically converted into the languages required by the corresponding model checkers. So, the verification process in kPWorkbench is carried out in automatic manner.

6 Testing kP Systems Using Automata Based Techniques

In this section we outline how the kP systems obtained in the previous sections can be tested using automata based testing methods. The approach presented here follows the blueprint presented in [24] and [15] for cell-like P systems. We illustrate our approach on $k\Pi_1$, the application of our approach on the other kP system modeling sorting algorithms is similar.

Naturally, in order to apply an automata based testing method to a kP model, a finite automata needs to be obtained first. In general, the computation of a kP system cannot be fully modelled by a finite automaton and so an *approximate* automaton will be sought. The problem will be addressed in two steps.

- Firstly, the computation tree of a P system will be represented as a deterministic finite automaton. In order to guarantee the finiteness of this process, an upper bound k on the length of any computation will be set and only computations of maximum k transitions will be considered at a time.
- Secondly, a *minimal* model, that preserves the required behaviour, will be defined on the basis of the aforementioned derivation tree.

Let $M_k = (A_k, Q_k, q_{0,k}, F_k, h_k)$ be the finite automaton representation of the computation tree, where A_k is the finite input alphabet, Q_k is the finite set of states, $q_{0,k} \in Q_k$ is the initial state, $F_k \subseteq Q_k$ is the set of final states, and $h_k : Q_k \times A_k \longrightarrow Q_k$ is the next-state function. A_k is composed of the tuples of multisets that label the transition of the computation tree. The states of T_k correspond to the nodes of the tree. For testing purposes we will consider all the states as final. It is implicitly assumed that a non-final "sink" state q_{sink} that receives all "rejected" transitions, also exists.

Consider $k\Pi_1$, the kP system in section 4.1, $n = 6$ and the sequence to be sorted 3, 6, 9, 5, 7, 8. Then the initial multisets are:
$w_{1,0} = a^3c^{10}p; w_{2,0} = a^6c^{10}; w_{3,0} = a^9c^{10}p; w_{4,0} = a^5c^{10}; w_{5,0} = a^7c^{10}p; w_{6,0} = a^8c^{10}$. As $k\Pi_1$ is a deterministic kP system, there are no ramification in the computation tree. For $k = 3$, this is represented below.

Compartments - Step	C_1	C_2	C_3	C_4	C_5	C_6
0	$rr^3_{1,1}r_{2,1}$	r	$rr^9_{1,3}r_{2,3}$	r	$rr^7_{1,5}r_{2,5}$	r
1	$rr_{3,1}$	$rr^3_{4,2}$	$rr_{3,3}$	$rr^5_{4,4}r^4_{5,4}$	$rr_{3,5}$	$rr^7_{4,6}$
2	r	$rr^6_{1,2}r_{2,2}$	r	$rr^9_{1,4}r_{2,4}$	r	$rr_{2,6}$
3	r	$rr'_{3,2}$	$rr^5_{1,3}r_{5,3}$	$rr'_{3,4}$	$rr^7_{1,5}r^2_{5,5}$	$rr'_{3,6}$

Let us denote
$\alpha_1 = (rr^3_{1,1}r_{2,1}, r, rr^9_{1,3}r_{2,3}, r, rr^7_{1,5}r_{2,5}, r)$,
$\alpha_2 = (rr_{3,1}, rr^3_{4,2}, rr_{3,3}, rr^5_{4,4}r^4_{5,4}, rr_{3,5}, rr^7_{4,6})$,
$\alpha_3 = (r, rr^6_{1,2}r_{2,2}, r, rr^9_{1,4}r_{2,4}, r, rr_{2,6})$,
$\alpha_4 = (r, rr'_{3,2}, rr^5_{1,3}r_{5,3}, rr'_{3,4}, rr^7_{1,5}r^2_{5,5}, rr'_{3,6})$.

Then, for $k = 3$, $M_k = (A_k, Q_k, q_{0,k}, F_k, h_k)$, where
$A_k = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, $Q_k = \{q_{0,k}, q_{1,k}, q_{2,k}, q_{3,k}, q_{4,k}\}$, $F_k = Q_k$, and h_k, the next-state function, is defined by: $h_k(q_{i-1,k}, \alpha_i) = q_{i,k})$, $1 \leq i \leq 4$.

As M_k is a deterministic finite automaton over A_k, one can find the minimal deterministic finite automaton that accepts *exactly* the language defined by M_k. However, as only sequences of at most k transitions are considered, it is irrelevant how the constructed automaton will behave for longer sequences. Consequently, a deterministic finite cover automaton of the language defined by M_k will be sufficient.

A *deterministic finite cover automaton* (*DFCA*) of a finite language U is a deterministic finite automaton that accepts all sequences in U and possibly other sequences that are longer than any sequence in U [4], [5]. A *minimal DFCA* of U is a DFCA of U having the least possible states. A minimal DFCA may not be unique (up to a renaming of its states). The great advantage of using a minimal DFCA instead of the minimal deterministic automaton that accepts precisely the language U is that the size (number of states) of the minimal DFCA may be much less than that of the minimal deterministic automaton that accepts U. Several algorithms for constructing a minimal DFCA (starting from the deterministic automaton that accepts the language U) exist, the best known algorithm [26] requires $O(n\ log\ n)$ time, where n denotes the number of states of the original automaton. For details about the construction of a minimal DFCA we refer the reader to [24] and [26].

A minimal DFCA of the language defined by M_k, $k = 3$, is $M = (A, Q, q_0, F, h)$, where $A = A_k$, $Q = \{q_0, q_1, q_2, q_3\}$, $F = Q$ and h defined by: $h(q_{i-1}, \alpha_i) = q_i$, $1 \leq i \leq 3$ and $h(q_3, \alpha_4) = q_0$.

Now, suppose we have a finite state model (automaton) of the system we want to test. In *conformance testing* one constructs a finite set of input sequences, called *test suite*, such that the implementation passes all tests in the test suite if and only if it behaves identically to the specification on any input sequence. Naturally, the implementation under test can also be modelled by an unknown deterministic finite automaton, say M'. This is not known, but one can make assumptions about it (e.g. that may have a number of incorrect transitions, missing or extra states). One of the least restrictive assumptions refers to its size (number of states). The *W-method* [12] assumes that the difference between the number of states of the implementation model and that of the specification has to be at most β, a non-negative integer estimated by the tester. The W-method involves the selection of two sets of input sequences, a state cover S and a characterization set W [12].

In our case, we have constructed a DFCA model of the system and we are only interested of the behavior of the system for sequences of length up to an upper bound k. Then, the set suite will only contain sequences of up to length k and its successful application to the implementation under test will establish that the implementation will behave identically to the specification for any sequence of length less then or equal to k. This situation is called conformance testing for bounded sequences. Recently, it was shown that the underlying idea of the W-method can also be applied in the case of bounded sequences, provided that

the sets S and W used in the construction of the test suite satisfy some further requirements; these are called a proper state cover and strong characterization set, respectively [23]. In what follows we informally define these two concepts and illustrate them on our working example. For formal definitions we refer the reader to [23] or [24].

A *proper state cover* of a deterministic finite automaton $M = (A, Q, q_0, F, h)$ is a set of sequences $S \subseteq A^*$ such that for every state $q \in Q$, S contains a sequence of *minimum length* that reaches q. Consider M the DFCA in our example. Then λ is the sequence of minimum length that reaches q_0, σ_1 is a sequence of minimum length that reaches q_1, $\alpha_1\alpha_2$ is a sequence of minimum length that reaches q_2, $\alpha_1\alpha_2\alpha_3$ is a sequence of minimum length that reaches q_3. Furthermore, we can use any input symbol in $A \setminus \{\alpha_1\}$ to reach the (implicit) "sink" state, for example α_2. Thus, $S = \{\lambda, \alpha_1, \alpha_1\alpha_2, \alpha_1\alpha_2\alpha_3, \alpha_2\}$ is a proper state cover of M.

A *strong characterization set* of a minimal deterministic finite automaton $M = (A, Q, q_0, F, h)$ is a set of sequences $W \subseteq A^*$ such that for every two distinct states $q_1, q_2 \in Q$, W contains a sequence of minimum length that distinguishes between q_1 and q_2. Consider again our running example. λ distinguishes between the (non-final) "sink" state and all the other (final) states. A transition labelled α_1 is defined from q_0, but not from q_1, q_2 or q_3, so α_1 is a sequence of minimum length that distinguishes q_0 from q_1, q_2 and q_3. Similarly, α_2 is a sequence of minimum length that distinguishes q_1 from q_2 and q_3 and α_3 is a sequence of minimum length that distinguishes between q_2 and q_3. Thus $W = \{\lambda, \alpha_1, \alpha_2, \alpha_3\}$ is a strong characterization set of M,

Once we have established the sets S and W and the maximum number β of extra states that the implementation under test may have, a test suite is constructed by extracting all sequences of length up to k from the set

$$S(A^0 \cup A^1 \cup \ldots \cup A^\beta)W,$$

where A^i denotes the set of input sequences of length $i \geq 0$.

Note that some test sequences may be accepted by the DFCA model - these are called *positive tests* - but some others may not be accepted (they end up in the (non-final) "sink" state) - these are called *negative tests*.

7 Conclusions

In this paper, we have investigated the relationships between kP systems, on the one hand, and active membrane systems with polarization and symport/antiport membrane systems, on the other hand. We have also illustrated the modeling power of kP systems by providing a number of kP system models for sorting algorithms. We have also discussed the problem of testing systems modelled as kernel P systems and proposed a test generation method based on automata. Namely, we have outlined how the kP systems can be tested using automata based

testing methods. Furthermore, we have shown how formal verification can be used to validate that the given models work as desired.

We have also begun a study on the ability of kP aytems to simulate other particular classes of P systems, We have presented here the case of P systems with active membranes, and P systems with symport/antiport rules.

In future studies we aim to connect kP systems with other classes of P systems, especially those utilised in various applications, and to show how other problems can be solved, tested and verified by using kP systems.

Acknowledgements.

MG and SK acknowledge the support provided for synthetic biology research by EPSRC ROADBLOCK (project number: EP/I031812/1). The work of FI and MG were supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI (project number: PN-II-ID-PCE-2011-3-0688).

References

1. A. Alhazov, D. Sburlan, Static Sorting Algorithms for P Systems, *Pre-Proc. 4^{th} Workshop on Membrane Computing* (A. Alhazov et al., eds.), *GRLMC Rep.* 28/03, Tarragona, 17 – 40, 2003.
2. A. Alhazov, D. Sburlan, Static Sorting P Systems. In [14], 215 – 252, 2006.
3. J.J. Arulanandham, Implementing Bead-Sort with P Systems, *Unconventional Models of Computation* (C.S. Calude et al., eds.), *Lecture Notes in Computer Science*, 2509, 115–125, 2002.
4. C. Câmpeanu, N. Santean, S. Yu. Minimal Cover-Automata for Finite Languages, *Workshop on Implementing Automata* (J.-M. Champarnaud et al., eds.), *Lecture Notes in Computer Science*, 1660, 43 – 56, 1998.
5. C. Câmpeanu, N. Santean, S. Yu. Minimal Cover-Automata for Finite Languages. *Theoretical Computer Science*, 267(1-2), 3 – 16, 2001.
6. R. Ceterchi, C. Martín-Vide, P Systems with Communication for Static Sorting. In *Pre-Proc. 1^{st} Brainstorming Week on Membrane Computing* (M. Cavaliere et al., eds.), *Technical Report* no 26, Rovira i Virgili Univ., Tarragona, 101 – 117, 2003.
7. R. Ceterchi , C. Martín-Vide, Dynamic P Systems, *Proc. 4^{th} Workshop on Membrane Computing* (Gh. Păun et al., eds.), *Lecture Notes in Computer Science*, 2597, 146 – 186, 2003.
8. R. Ceterchi, M.J. Pérez-Jiménez, A.I. Tomescu, Simulating the Bitonic Sort Using P Systems, *Proc. 8^{th} Workshop on Membrane Computing* (G. Eleftherakis et al., eds.), *Lecture Notes in Computer Science*, 4860, 172 – 192, 2007.
9. R. Ceterchi, M.J. Pérez-Jiménez, A.I. Tomescu, Sorting Omega Networks Simulated With P Systems: Optimal Data Layouts, (D. Diaz-Pernil et al., eds.), *Pre-Proc. 6^{th} Brainstorming Week on Membrane Computing*, *RGNC Rep.* 01/08, Fénix Editora, pp. 79 – 92, 2008.
10. R. Ceterchi, A. I. Tomescu, Implementing Sorting Networks with Spiking Neural P Systems, *Fundamenta Informaticae*, 87(1), 35 – 48, 2008.

11. R. Ceterchi, D. Sburlan, Membrane Computing and Computer Science, Chapter 22 of [29], 553–583, 2010.
12. T. S. Chow Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engineering*, 4(3), 178 – 187, 1978.
13. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV Version 2: An Open Source Tool for Symbolic Model Checking, *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, (W.A. Hunt, Jr and F. Somenzi, eds.), *Lecture Notes in Computer Science*, 2404, 359 – 364, 2002.
14. G. Ciobanu, Gh. Păun, M. J. Pérez-Jiménez, eds., *Applications of Membrane Computing*, Springer, 2006.
15. M. Gheorghe, F. Ipate. On Testing P Systems, *Proc. 9^{th} Workshop on Membrane Computing*, (D.W. Corne et al., eds.), *Lecture Notes in Computer Science*, 5391, 204 – 216, 2009.
16. M. Gheorghe, F. Ipate, C. Dragomir, Kernel P Systems, *Pre-proc. 10^{th} Brainstorming Week on Membrane Computing*, (M. A. Martínez-del-Amor et al., eds.), Fénix Editora, Universidad de Sevilla, 153 – 170, 2012.
17. M. Gheorghe, F. Ipate, C. Dragomir, L. Mierlă, L. Valencia-Cabrera, M. García-Quismondo, M.J. Pérez-Jiménez, Kernel P Systems – Version 1, *Pre-Proc. 11th Brainstorming Week on Membrane Computing*, (L. Valencia-Cabrera et al., eds.), Fénix Editora, Universidad de Sevilla, 97 – 124, 2013.
18. M. Gheorghe, F. Ipate, S. Konur, Kernel P Systems and Relationships with other Classes of P Systems, *Multidisciplinary Creativity*, (M. Gheorghe et al., eds.), Spandugino Publishing House, 64 – 76, 2015.
19. M. Gheorghe, F. Ipate, R. Lefticaru, M.J. Pérez-Jiménez, A. Ţurcanu, L. Valencia-Cabrera, M. García-Quismondo, L. Mierlă, 3-COL Problem Modelling Using Simple kernel P Systems, *International Journal of Computer Mathematics*, 90(4), 816 – 830, 2013.
20. M. Gheorghe, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, Research Frontiers of Membrane Computing: Open Problems and Research Topics, *International Journal of Foundations of Computer Scienve*, 24, 547 – 624, 2013.
21. M. Gheorghe, S. Konur, F. Ipate, L. Mierlă, M. E. Bakir, M. Stannett, An Integrated Model Checking Toolset for Kernel P Systems, *Proc. 16^{th} Conference on Membrane Computing*, (G. Rozenberg et al., eds.), *Lecture Notes in Computer Science*, 9504, 153 – 170, 2015.
22. G. J. Holzmann, The Model Checker SPIN, *IEEE Transactions on Software Engineering*, 23(5), 275 – 295, 1997.
23. F. Ipate, Bounded Sequence Testing from Deterministic Finite State Machines, *Theoretical Computer Science*, 411(16-18), 1770 – 1784, 2010.
24. F. Ipate, M. Gheorghe. Finite State Based Testing of P Systems, *Natural Computing*, 8(4), 833 – 846, 2009.
25. D.E. Knuth, *The Art of Computer Programming*, vol. 3, *Sorting and Searching*, Addison-Wesley, 1973.
26. H. Körner. On Minimizing Cover Automata for Finite Languages in O(n log n) Time, *Proc. 7^{th} Conference on Implementation and Application of Automata*, (J.-M. Champarnaud and D. Morel, eds.), *Lecture Notes in Computer Science*, 2608, 117 – 127, 2002.
27. Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences*, 61(1), 108 – 143, 2000.

28. Gh. Păun, Membrane Computing - An Introduction, Springer, 2002.
29. Gh. Păun, G. Rozenberg, A. Salomaa, eds., *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
30. D. Sburlan, A Static Sorting Algorithm for P Systems with Mobile Catalysts, *Analele Ştiinţifice Universitatea Ovidius Constanţa*, 11(1), 195 – 205, 2003.

On the Classes of Languages Characterized by Generalized P Colony Automata

Kristóf Kántor, György Vaszil

Department of Computer Science, Faculty of Informatics
University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
{kantor.kristof, vaszil.gyorgy}@inf.unideb.hu

Summary. We study the computational power of generalized P colony automata and show how it is influenced by the capacity of the system (the number of objects inside the cells of the colony) and the types of programs which are allowed to be used (restricted and unrestricted com-tape and all-tape programs, or programs allowing any kinds of rules).

1 Introduction

P colonies are variants of very simple membrane systems, which are similar to so-called colonies of simple grammars, a model in the theory of grammar systems, launched by the introduction of cooperating, distributed systems of grammars in [4]. One of the grammatical models of the field is the colony of grammars, see [12], which is a collection of very simple generative grammars, but as a system, they are able to generate complicated languages. For more on grammar systems and colonies the interested reader is referred to the monograph [5].

Similarly to the grammar systems variant, P colonies also consist of a collection of very simple computing agents which interact in a shared environment, see [13, 14]. The environment and the computing agents are both described by multisets of objects which are processed by the colony members using rules which enable the transformation of the objects and the exchange of objects between the colony members and the environment. The rules are grouped into programs, which execute the rules they contain in parallel. A computation consists of a sequence of computational steps during which the colony members execute their programs in parallel, until the system reaches a halting configuration.

P colony automata, a variant of P colonies characterizing string languages instead of multiset collections were introduced in [3] where several of its variants were shown to be computationally complete in. The power of some of those left open there was further examined in [1].

Generalized P colony automata were introduced in [11] in order to make the model resemble more to the standard models of membrane computing, in particu-

lar, to the model of P automata, introduced in [7]. In this case, the computation of the colony defines an accepted multiset sequence, which is turned into an accepted string by a non-erasing mapping (as in P automata). In [11] some basic variants of the model were introduced and studied from the point of view of their computational power. Here we continue the investigations by examining generalized P colony automata of capacity one, two, and three, and also take the initial steps in the study of the relationship of their languages and the languages accepted by P automata.

2 Preliminaries and Definitions

Let V be a finite alphabet, let the set of all words over V be denoted by V^*, and let ε be the empty word. We denote the number of occurrences of a symbol $a \in V$ in w by $|w|_a$.

A *two-counter machine*, see [9], $M = (\Sigma \cup \{Z, B\}, Q, q_0, q_F, Tr)$ is a 3-tape Turing machine where Σ is an *alphabet*, Q is a set of *internal states* with $q_0, q_F \in Q$ being the initial and the final states, and Tr is a set of *transition rules*. The machine has a read-only input tape and two semi-infinite storage tapes which are used as counters. The alphabet of the storage tapes contains only two symbols, Z and B (blank), while the alphabet of the input tape is $\Sigma \cup \{B\}$. The symbol Z is written on the first, leftmost cells of the storage tapes which are scanned initially by the tape heads. An integer t can be stored by moving a tape head t cells to the right of Z. A stored number can be incremented or decremented by moving the tape head right or left. The machine is capable of checking whether a stored value is *zero* or not by looking at the symbol scanned by the tape heads. If the scanned symbol is Z, then the value stored in the corresponding counter is *zero*.

Without the loss of generality, we assume that two-counter machines check and modify only one of their counters during any transition, thus, the rule set Tr contains transition rules of the form $(q, x, i, \alpha) \to (q', \beta)$ where $x \in \Sigma \cup \{B\} \cup \{\varepsilon\}$ corresponds to the symbol scanned on the input tape in state $q \in Q$, and $\alpha \in \{Z, B\}$, $i \in \{1, 2\}$ correspond to the symbols scanned on the i-th storage tape. By a rule of the above form, M enters state $q' \in Q$, and the i-th counter is modified according to $\beta \in \{-1, 0, +1\}$. If $x \in \Sigma \cup \{B\}$, then the machine was scanning x on the input tape, and the head moves one cell to the right; if $x = \varepsilon$, then the machine performs the transition irrespective of the scanned input symbol, and the reading head does not move.

A word $w \in \Sigma^*$ is accepted by the two-counter machine if starting in the initial state q_0, the input head reaches and reads the rightmost non-blank symbol on the input tape, and the machine is in the accepting state q_F. Two-counter machines are computationally complete; they are just as powerful as Turing machines (see [9] for more details).

We will also need the notion of a *register machine*, and we also consider a variant: a *register machine with input tape*. Such a machine consists of a given

number of registers each of which can hold an arbitrarily large non-negative integer number (we say that the register is empty if it holds the value zero), and a set of labeled instructions which specify how the numbers stored in registers can be manipulated (see [15] for more information).

Formally, a *register machine* is a construct $M = (m, H, l_0, l_h, R)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halting label, and R is the set of instructions; each label from H labels only one instruction from R. There are several types of instructions which can be used. For $l_i, l_j, l_k \in H$ and $r \in \{1, \ldots, m\}$ we have

- $l_i : (\texttt{ADD}(r), l_j)$ - *add*: Add 1 to register r and then go to the instruction with label l_j.
- $l_i : (\texttt{CHECKSUB}(r), l_j, l_k)$ - *zero check and subtract*: If the value of register r is not zero, subtract one from it and go to instruction l_j, otherwise leave it unchanged and go to l_k.
- $l_h : \texttt{HALT}$ - *halt*: Stop the machine.

A register machine accepts a number m if starting the computation with the instruction labeled by l_0 while having m in the first register (and all other registers empty), it reaches the halting instruction. This way a register machine computes a set of numbers.

To be able to accept strings, we might also add an input tape to a register machine, together with a new type of instruction

- $l_i : (\texttt{READ}(a), l_j)$ for a symbol $a \in \Sigma$ of some input alphabet Σ.

Such an instruction can be applied if the reading head scans a symbol $a \in \Sigma$ on the input tape, and the head moves to the next tape cell after the application of the instruction.

It is not difficult to see that register machines with input tape characterize the class of recursively enumerable languages, as they can simulate two-counter machines. To see this, consider the following. For each transition $t : (q, x, i, \alpha) \to (q', \beta)$ of a two-counter machine M_{2c}, construct the instructions for a register machine M_R as follows.

Let M_R have a register r_q for each state $q \in Q$, and a register r_i for each counter c_i of M_{2c}. Initially the register for the initial state contains the value one, and all other registers are empty. The transition t is simulated by several instructions of M_R.

The simulation starts with $l_t : (\texttt{READ}(x), l_{t,1})$, and then continues with $l_{t,1} : (\texttt{CHECKSUB}(q), l_{t,2}, l_{trap})$ where l_{trap} is a "trap" label with a "trap" instruction $l_{trap} : (\texttt{ADD}(q), l_{trap})$. Then $l_{t,2} : (\texttt{ADD}(q'), l_{t,3})$ follows for the new state q'. Now, if $\alpha = B$, then the next instructions are $l_{t,3} : (\texttt{CHECKSUB}(i), l_{t,4}, l_{trap})$, and $l_{t,4} : (\texttt{ADD}(i), l_{t,5})$, if $\alpha = 0$ then $l_{t,3} : (\texttt{CHECKSUB}(i), l_{trap}, l_{t,5})$. These instructions check the required state of the ith register. If $\beta = 0$, then $l_{t,5}$ can be replaced by the label $l_{t'}$ for a new transition t' (starting with the state q') of M_{2c}. If $\beta = -1$, then $l_{t,5} : (\texttt{CHECKSUB}(i), l_{t'}, l_{trap})$, if $\beta = +1$, then $l_{t,5} : (\texttt{ADD}(i), l_{t'})$.

If we define the instructions of M_R in such a way that each accepting transition of M_{2c} can also lead to the halting instruction, then M_R accepts an input word if and only if M_{2c} does.

Now we define the notions related to multisets as follows. If the set of non-negative integers is denoted by $\mathbb{N}$, then a multiset over a set V is a mapping $M : V \to \mathbb{N}$ which assigns to each object $a \in V$ its multiplicity $M(a)$ in M. The support of M is the set $supp(M) = \{a \mid M(a) \geq 1\}$. If V is a finite set, then M is called a finite multiset. A multiset M is empty if its support is empty, $supp(M) = \emptyset$. We will represent a finite multiset M over V by a string w over the alphabet V with $|w|_a = M(a)$, $a \in V$, and ε will represent the empty multiset which is also denoted by $\emptyset$.

We say that $a \in M$ if $M(a) \geq 1$, and the cardinality of M, $card(M)$ is defined as $card(M) = \Sigma_{a \in M} M(a)$. For two multisets $M_1, M_2 : V \to \mathbb{N}$, $M_1 \subseteq M_2$ holds, if for all $a \in V$, $M_1(a) \leq M_2(a)$. The union of M_1 and M_2 is defined as $(M_1 \cup M_2) : V \to \mathbb{N}$ with $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$ for all $a \in V$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) : V \to \mathbb{N}$ with $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$.

A P system, see [17], is a structure of hierarchically embedded membranes (a rooted tree), each having a unique label and enclosing a region containing a multiset of objects. The outmost membrane is called the skin membrane.

An antiport rule is of the form $(u, in; v, out)$, where $u, v \in V^*$ are finite multisets over V. If such a rule is applied in a region, then the objects of u enter from the parent region and, in the same step, objects of v leave to the parent region.

A *P automaton*, see [6] $\Pi = (V, \mu, w_1, \ldots, w_k, P_1, \ldots, P_k)$ is a membrane system with object alphabet V, membrane structure μ, initial contents (multisets) of the ith region $w_i \in V^*$, $1 \leq i \leq k$, and sets of antiport rules P_i, $1 \leq i \leq k$.

The configurations of the P automaton can be changed by transitions in the sequential mode (seq) or in the non-deterministic maximally parallel mode (par). In the first case one rule is applied in each region in every step, in the second case as many rules are applied simultaneously in the regions at the same step as possible. Thus, a transition in the P automaton Π is $(v_1, \ldots, v_m) \in \delta_{\Pi,X}(u_0, u_1, \ldots, u_m)$, where $\delta_{\Pi,X}$ denotes the transition relation, $X \in \{seq, par\}$, $u_1, \ldots, u_k$ are the contents of the k regions, u_0 is the multiset entering the system from the environment, and $v_1, \ldots, v_k$, respectively, are the contents of the k regions after performing the transition in the working mode.

In this way, there is a sequence of multisets which enter the system from the environment during the steps of its computations. If the computation is accepting, that is, if it halts, then this multiset sequence is called an accepted multiset sequence, and denoted by $A(\Pi)$ for a P automaton Π.

Before giving the definition of the accepted string languages of P automata, we define the notion of a generalized P colony automaton (genPCol automaton in short).

Definition 1. A *genPCol automaton* of capacity k and with n cells, $k, n \geq 1$, is a construct $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ where

- V is an *alphabet*, the alphabet of the automaton, its elements are called *objects*;
- $e \in V$ is the *environmental object* of the automaton;
- $w_E \in (V - \{e\})^*$ is a string representing the multiset of objects different from e which is found in the environment initially;
- $(w_i, P_i), 1 \leq i \leq n$, specifies the i-th *cell* where w_i is a multiset over V, it determines the initial contents of the cell, and its cardinality $|w_i| = k$ is called the *capacity* of the system. The sets P_i of *programs* are formed from k rules of the following types:
  - *tape rules* of the form $a \xrightarrow{T} b$, or $a \overset{T}{\leftrightarrow} b$, called rewriting tape rules and communication tape rules, respectively; or
  - *nontape rules* of the form $a \to b$, or $c \leftrightarrow d$, called rewriting (nontape) rules and communication (nontape) rules, respectively.

  A program is called a *tape program* if it contains at least one tape rule.
- F is a set of *accepting configurations* of the automaton which we will specify in more detail below.

A genPCol automaton reads an input word during a computation. A part of the input (possibly consisting of more than one symbols) is read during each configuration change: the processed part of the input corresponds to the multiset of symbols introduced by the tape rules of the system. This process is defined more precisely as follows.

A *configuration* of a genPCol automaton is an $(n+1)$-tuple $(u_E, u_1, \ldots, u_n)$, where $u_E \in (V - \{e\})^*$ represents the multiset of objects different from e in the environment, and $u_i \in V^*$, $1 \leq i \leq n$, represent the contents of the i-th cell. The *initial configuration* is given by $(w_E, w_1, \ldots, w_n)$, the initial contents of the environment and the cells. The elements of the set F of *accepting configurations* are given as configurations of the form $(v_E, v_1, \ldots, v_n)$, where

- $v_E \subseteq (V - \{e\})^*$ represents a multiset of objects different from e being in the environment, and each
- $v_i \in V^*$, $1 \leq i \leq n$, is the contents of the i-th cell.

To describe the computation process formally, for any rule r we define the following multisets. Let $X \in \{T, \varepsilon\}$, and if $r = a \xrightarrow{X} b$, or $r = a \overset{X}{\leftrightarrow} b$, then let $left(r) = a$, $right(r) = b$. Let us extend this notation also for programs. For $\alpha \in \{left, right\}$ and for any program p, let $\alpha(p) = \bigcup_{r \in p} \alpha(r)$ where the union denotes multiset union (as defined above), and for a rule r and program $p = \langle r_1, \ldots, r_k \rangle$, the notation $r \in p$ denotes the fact that $r = r_j$ for some j, $1 \leq j \leq k$. Moreover, for any tape program p we also define $read(p)$ as the multiset of symbols (different from e) on the right side of rewriting tape rules and on the left side of communication tape rules, that is, $read(p) = \bigcup_{r \in p, r = a \xrightarrow{T} b, b \neq e} right(r) \cup \bigcup_{r \in p, r = a \overset{T}{\leftrightarrow} b, a \neq e} left(r)$. Thus, $left(r)$ and $right(r)$ are the multisets consisting of the symbol on the left or right side of the rule r. For a program p, $left(p)$ and $right(p)$ are the collection (multiset) of symbols on the left or right sides of the rules in the program p. Finally,

$read(p)$ is the multiset (collection) of symbols (different from e) on the right side of rewriting tape rules or the left side of communication tape rules.

We also denote by $export(p)$ and by $import(p)$ the multisets $export(p) = \bigcup_{r\in p, r=a\overset{X}{\leftrightarrow}b, a\neq e} a$ and $import(p) = \bigcup_{r\in p, r=a\overset{X}{\leftrightarrow}b} b$, and by $create(p)$ the multiset $create(p) = \bigcup_{r\in p, r=a\overset{X}{\rightarrow}b} b$. So by $export(p)$ and $import(p)$, that were defined for communication rules of a given program p, we indicate the objects that are sent out to the environment and brought inside the cell, respectively. Whereas $create(p)$ is the multiset of symbols produced by the rewriting rules of program p.

Let the programs of each P_i be labeled in a one-to-one manner by labels from the set $lab(P_i)$, $1 \leq i \leq n$, $lab(P_i) \cap lab(P_j) = \emptyset$ for $i \neq j$. In the following, for the sake of brevity, if no confusion arises, we designate programs and their labels with the same letters, thus, for a label $p \in lab(P_i)$, we also write $p \in P_i$.

Let $c = (u_E, u_1, \ldots, u_n)$ be a configuration of a genPCol automaton Π, and let $U_E = u_E \cup \{e, e, \ldots\}$, thus, the multiset of objects found in the environment (together with the infinite number of es which are always present). We call a *set of programs*, P_c, *applicable in configuration* c, if the following conditions hold.

- At most one program is selected for each cell, that is, if $p, p' \in P_c, p \neq p'$ and $p \in P_i, p' \in P_j$, then $i \neq j$;
- the selected programs are applicable in the cells (the left sides of the rules contain the same symbols that are present in the cell), that is, for each $p \in P_c$, if $p \in P_i$ then $left(p) = u_i$;
- the symbols which are brought inside the cells by the programs are present in the environment, that is, $\bigcup_{p\in P_c} import(p) \subseteq U_E$;
- P_c is maximal, that is, if any other program is added to it, then some of the above conditions are not satisfied.

A configuration $c = (u_E, u_1, \ldots, u_n)$ is *changed* to a configuration $c' = (u'_E, u'_1, \ldots, u'_n)$ and is denoted by $c \Longrightarrow c'$ by applying the set P_c of applicable programs if the following properties hold:

- If there is a $p \in P_c$ such that $p \in P_i$, then $u'_i = create(p) \cup import(p)$, otherwise $u'_i = u_i$, $1 \leq i \leq n$; and
- $U'_E = U_E - \bigcup_{p\in P_c} import(p) \cup \bigcup_{p\in P_c} export(p)$ (where U'_E again denotes $u'_E \cup \{e, e, \ldots\}$ with an infinite number of es).

We denote the reflexive and transitive closure of $\Longrightarrow$ by $\Longrightarrow^*$.

The general idea behind the above definitions is that instead of the different computational modes used in [3], we have a system with programs and we apply the programs in the maximally parallel way as usual in P colonies, that is, in each computational step, every component cell must non-deterministically choose and apply one of its applicable programs. Then we look at those rules which were tape rules (in the applied set of programs) and collect all the symbols that they "read": this multiset (of the collected symbols) is the multiset read by the system in the given computational step. A successful computation defines this way an accepted

sequence of multisets: the sequence of multisets entering the system during the steps of the computation.

Definition 2. Let $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ be a genPCol automaton. The *set of input sequences accepted by* Π is defined as

$$A(\Pi) = \{u_1 u_2 \ldots u_s \mid u_i \in (V - \{e\})^*,\ 1 \leq i \leq s, \text{ and there is a configuration sequence } c_0, \ldots, c_s, \text{ with } c_0 = (w_E, w_1, \ldots, w_n),\ c_s \in F, \text{ and } c_i \Longrightarrow c_{i+1} \text{ with } \bigcup_{p \in P_{c_i}} read(p) = u_{i+1} \text{ for all } 0 \leq i \leq s-1\}.$$

Now we define the accepted string languages for both genPCol automata, and "ordinary" P automata.

Definition 3. Let Π be a genPCol automaton or a P automaton, and let $f : (V - \{e\})^* \to 2^{\Sigma^*}$ be a mapping, such that $f(u) = \varepsilon$ if and only if u is the empty multiset.

The *language accepted by* Π with respect to f is defined as

$$L(\Pi, f) = \{f(u_1)f(u_2)\ldots f(u_s) \in \Sigma^* \mid\ u_1 u_2 \ldots u_s \in A(\Pi)\}.$$

From now on, we are going to consider the mapping: f_{perm} defined for any multiset $x \in (V - \{e\})^*$ as

$$f(x) = \{y \in (V - \{e\})^* \mid y \in perm(x)\}$$

where $perm(x) \subseteq V^*$ denotes the set of strings representing the multiset composed of the symbols of x, or in other words, $perm(x)$ is the set of strings obtained by a permutation of the symbols of the multiset x.

Concerning the power of P automata with the mapping f_{perm}, the reader is referred to [8] and [10]. In general, they characterize a language class that is strictly included in the class of languages that can be accepted by logarithmic space bounded Turing machines that read their input tape from left to right only once.

For genPCol automata, their working modes, or in other words, the types of programs that they are allowed to use, greatly influence their computational power. Let us refine and extend the definition of the program types defined in [11] as follows.

Definition 4.

- $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{com-tape}(k))$ is the class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ where all the communication rules are tape rules,
- $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{all-tape}(k))$ is the class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ where all the programs must have at least one tape rule,

- $\mathcal{L}(\text{genPCol}, \mathcal{F}, *(k))$ is the class of languages accepted by generalized PCol automata with capacity k and with mappings from the class $\mathcal{F}$ where programs with any kinds of rules are allowed.

For all-tape and com-tape languages we also define their *restricted* variants, $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{restricted all-tape}(k))$ and $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{restricted com-tape}(k))$, respectively. These are accepted by systems with programs not having any rules of types

$$e \xrightarrow{T} e, \ a \xrightarrow{T} e, \text{ and } e \xleftrightarrow{T} e, \ e \xleftrightarrow{T} a,$$

for arbitrary $a \in V$, where e is the special environmental object. Note that systems which accept languages of these restricted classes must read nonempty multisets in each computational step.

In the following, we will be considering systems with the permutation mapping f_{perm} defined above. For the sake of easier readability, we denote the languages of systems with this type of mapping as

- $\mathcal{L}_{perm}(\text{genPCol}, X(k))$, where $X \in \{\text{com-tape}, \text{all-tape}, *\}$.

3 Languages Accepted by genPCol Automata

The following are immediate consequences of the definitions.

Proposition 1 *For any class of mappings $\mathcal{F}$, we have*

1. $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{com-tape}(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, *(k))$ *and* $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{all-tape}(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, *(k)$ *for any* $k \geq 1$*;*
2. $\mathcal{L}(\text{genPCol}, \mathcal{F}, \text{restricted } X(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, X(k))$ *for any* $k \geq 1$ *and* $X \in \{\text{com-tape}, \text{all-tape}, *\}$*; and*
3. $\mathcal{L}(\text{genPCol}, \mathcal{F}, X(k)) \subseteq \mathcal{L}(\text{genPCol}, \mathcal{F}, X(k+1))$ *for any* $k \geq 1$ *and* $X \in \{\text{com-tape}, \text{all-tape}, *\}$.

Proof. The first inclusions hold, as com-tape systems are special cases of all-tape systems, which are both special cases of the unrestricted variant. The second inclusion holds for a similar reason, while the third inclusion can be seen to hold if we consider that adding the of object e to the initial cell contents, and a rule $e \to e$ to the programs of all cells in a system, does not change the accepted language. □

3.1 The Capacity of genPCol Automata

First we consider genPCol automata of capacity one. In the case of P colonies, all recursively enumerable sets of integers can be characterized by systems of capacity one, see [2]. This is also true for genPCol automata with languages obtained by permutation mappings, if programs with any kind of rules are allowed.

Theorem 2. $\mathcal{L}_{perm}(\text{genPCol}, *(1)) = \mathcal{L}(RE)$.

Proof. In Theorem 1 of [2] P colonies of capacity one are shown to be able to simulate register machines. The idea of the simulation is to have an object in the environment corresponding to the label of the instruction which is to be simulated next. The cells of the system "process" the instruction label in such a way that the necessary modifications of the configuration are implemented, and the label of the next instruction is sent to the environment.

Based on this construction, we can show that genPCol automata can simulate register machines with input tape (see section 2 for the definitions), and thus, characterize the class of recursively enumerable languages. In addition to the construction in Theorem 1 of [2], we need to simulate the instructions of type $l_i : (\texttt{READ}(a), l_j)$. To do this, we add one cell (e, P_{l_i}) to the system for each such instruction of the register machine with the programs

$$P_{l_i} = \{\langle e \leftrightarrow l_i \rangle,\ \langle l_i \xrightarrow{T} a \rangle,\ \langle a \rightarrow l_j \rangle, \{\langle l_j \leftrightarrow e \rangle\}.$$

These programs can be applied when l_i appears in the environment. They read an input symbol a while exchanging l_i for l_j in the environment. □

The power of systems with capacity one decreases considerably if not all kinds of programs are allowed. The next theorem examines the relationship of regular languages and languages of genPCol automata with all-tape programs.

Theorem 3. $\mathcal{L}_{perm}(\text{genPCol}, \text{all-tape}(1))$ *is incomparable with the class of regular languages.*

Proof. First we show that there is a nonregular language in $\mathcal{L}_{perm}(\text{genPCol}, *(1))$. Let $L_1 = \{\{a, \$\}^{2n}\{c, \$\}^2\{b, \$\}^{2n+2} \mid n \geq 0\}$ be the non-regular language over $\Sigma = \{a, b, c, \$\}$, where by $\{x, y\}^m$ we denote the string $w_1 w_2 \dots w_m$ with w_i being either xy or yx, $1 \leq i \leq m$.

Consider $\Pi = (\Sigma \cup \{e\}, e, w_E, (e, P_1), (e, P_2), (e, P_3), F)$, the genPCol automaton with the sets of programs as

$$\begin{aligned} P_1 &= \{\langle e \xrightarrow{T} \$ \rangle, \langle \$ \xleftrightarrow{T} e \rangle\}, \\ P_2 &= \{\langle e \xrightarrow{T} a \rangle, \langle a \xleftrightarrow{T} e \rangle, \langle e \xrightarrow{T} c \rangle, \langle c \xleftrightarrow{T} \$ \rangle\}, \\ P_3 &= \{\langle b \xleftrightarrow{T} c \rangle, \langle c \xrightarrow{T} b \rangle, \langle b \xleftrightarrow{T} a \rangle, \langle a \xrightarrow{T} b \rangle\}, \end{aligned}$$

and set of accepting configurations: $F = \{(u, e, \$, b) \mid u \in (\Sigma \setminus \{a\})^*\}$.

It is easy to see, that the first cell starts producing \$ objects indefinitely, while the second cell reads $2n$ $(n \geq 0)$ as, while sending as in the environment n times. After stopping, the third cell starts to work, eliminating every a in the environment while reading two bs.

Next, we show that $L_2 = \{bbc, c\}$ cannot be accepted by any Π genPCol automaton with capacity one, working in all-tape mode, using the f_{perm} mapping.

Let $V = \{b, c\} \cup \{e, e'\}$ be the alphabet of a genPCol automaton. Note that these are the only symbols that might appear in the programs, and e' can only serve as the initial cell contents. It is clear, that the automaton must accept *bbc* and *c*. We show that it is impossible to accept these and only these strings. Let us examine the cases:

1. There is only one cell: $\Pi = (V, e, w_E, (w_0, P_0), F)$. In this case there are three subcases:

1.(a) $w_0 = e'$. In order to decide whether to read *c* or *bbc*, Π must create two pathways. To do this, P_0 must contain the following programs: $\langle e' \overset{T}{\to} e\rangle, \langle e \overset{T}{\to} c\rangle, \langle e \overset{T}{\to} b\rangle$ or $\langle e' \overset{T}{\to} c\rangle, \langle e' \overset{T}{\to} b\rangle$. If nondeterministically Π decides to read *b*, it should be able to read one more *b*. To do this, we can either add $\langle b \overset{T}{\to} b\rangle$ or $\langle b \overset{T}{\leftrightarrow} e\rangle$ program to P_0, but it would create a nondeterminism, so that the automaton could read strings other than *bbc* or *c*.

1.(b) $w_0 = b$ or $w_0 = c$. In this case we would need to have the program $\langle b \overset{T}{\to} b\rangle$ again, or P_0 would contain $\langle b \overset{T}{\leftrightarrow} e\rangle$ or $\langle c \overset{T}{\leftrightarrow} e\rangle$. Since we have one cell, one of these rules automatically decides which string we would like to start reading, therefore it is impossible to accept both *bbc* and *c* strings.

1.(c) The only remaining option in this case is $w_0 = e$. Here P_0 must contain $\langle e \overset{T}{\to} b\rangle$ and $\langle e \overset{T}{\to} c\rangle$. If Π nondeterministically chooses $\langle e \overset{T}{\to} b\rangle$, then it would be still left to read *bc*. There are three different rules that could be used at the moment: $\langle b \overset{T}{\to} b\rangle, \langle b \overset{T}{\to} e\rangle, \langle b \overset{T}{\leftrightarrow} e\rangle$, however these cases lead to nondeterminism, where Π could read strings other than *bbc* or *c*.

2. There are $n \geq 2$ cells: $\Pi = (V, e, w_E, (w_0, P_0), \ldots, (w_n, P_n), F)$.We are able to define the ith $(0 \leq i \leq n)$ cell in three different ways. Please note that in order to decide whether to read *c* or *bbc*, Π must create two pathways. In these cases Π would create the pathways using two or more cells:

2.(a) $w_i = b$ or $w_i = c$. Hence the maximal parallelism, the ith cell would immediately read *c* or *b*, therefore restricting to accept only *bbc* or *c*.

2.(b) $w_i = e$. P_i could contain $\langle e \overset{T}{\leftrightarrow} b\rangle$ or $\langle e \overset{T}{\leftrightarrow} c\rangle$, but these cases require to have *b* or *c* in the environment, which is impossible because of the previous case. Thus P_i must contain one or more of these programs: $\langle e \overset{T}{\to} e\rangle$, $\langle e \overset{T}{\leftrightarrow} e\rangle$, $\langle e \overset{T}{\to} b\rangle$ or $\langle e \overset{T}{\to} c\rangle$. Please note that in all of these cases, Π would be able to accept strings other than *bbc* or *c*, therefore it would be impossible to accept *bbc* and *c*.

2.(c) $w_i = e'$. In this last subcase, P_i could contain $\langle e' \overset{T}{\to} e\rangle$, $\langle e' \overset{T}{\to} b\rangle$ or $\langle e' \overset{T}{\to} c\rangle$. Choosing $\langle e' \overset{T}{\to} e\rangle$ would lead to the previous case, where $w_i = e$. The two remaining programs would immediately read strings other than *bbc* or *c*.

3. The last case that is left to be examined is when there are $n \geq 2$ cells, and Π creates the nondeterministic pathways in one cell. Let the 0th cell be the one that creates the nondeterministic pathways. Hence case (1), $w_0 = e$ and P_0 must contain $\langle e \overset{T}{\to} b\rangle$ and $\langle e \overset{T}{\to} c\rangle$. If the 0th cell decides to read *b*, we must ensure that Π would read *bc* and then stop. If an other cell would continue to read, the only

logical scenario would be to have b or c in the cell at start and the cell might only use a program in the following form: $\langle k_1 \in \{b, c\} \overset{T}{\leftrightarrow} k_2 \in (\{b, c\} \cup \{e'\})\rangle$, but it is impossible to have k_2 in the environment without reading k_2. Thus the 0th cell is to continue to read b, which can happen by one of the following programs: $\langle b \overset{T}{\rightarrow} b\rangle$ or $\langle b \overset{T}{\leftrightarrow} e\rangle$, however both of them lead to unnecessary nondeterminism.

We have covered the possible ways to construct Π. It is now easy to see, that it is impossible to construct Π in any way to accept $\{bbc, c\}$. □

Now we show that for systems with capacity at least three, their all-tape and com-tape languages include any recursively enumerable language. Given a recursively enumerable language L, the idea is to take a system of capacity two which, when any kind of programs are allowed, accept L (we refer to [11] for such a system), and transform it to a system of capacity three having a communication tape rule in each program by adding "dummy" tape rules which do not interfere with the work of the rest of the system.

Proposition 4 $\mathcal{L}_{perm}(\text{genPCol}, X(3)) = \mathcal{L}(\text{RE})$ *for* $X \in \{\text{com-tape}, \text{all-tape}\}$.

Proof. The construction is based on the proof of Theorem 3 in [11] where a genPCol automaton of capacity two with no restriction on the type of programs is presented. Modifying such a system, we can easily construct a genPCol automaton of capacity three with all-tape or even com-tape type of programs by simply putting one more e object into each cell, and add the rule $e \overset{T}{\leftrightarrow} e$ to every program. □

3.2 Variants of genPCol Automata of with Capacity Two

In [11] we have started the study of genPCol automata languages that can be accepted by systems of capacity two with the mapping f_{perm}. We have shown that if they use restricted all-tape or restricted com-tape programs, then similarly to "ordinary" P automata, they characterize a language class that is strictly included in the class of languages that can be accepted by logarithmic space bounded Turing machines that read their input tape from left to right only once.

On the other hand, even genPCol automata with restricted all-tape or restricted com-tape programs are more powerful than P automata using the mapping f_{perm}.

If we denote by $\mathcal{L}_X(f_{perm}, PA)$ the class of languages characterized by P automata with $X \in \{seq, par\}$ for parallel or sequential rule application, then we have the following.

Theorem 5. $\mathcal{L}_{perm}(\text{genPCol}, \text{restricted all-tape}(2)) \setminus \mathcal{L}_X(f_{perm}, PA) \neq \emptyset$ *for* $X \in \{seq, par\}$.

Proof. Consider the language $L = \{(ab)^n(cd)^n \mid n \geq 1\}$ which, according to [10] cannot be accepted by any P automaton using the mapping f_{perm}. The following genPCol automaton accepts L with f_{perm}.

Let $\Pi = (\{a, b, c, d\}, e, \emptyset, (ee, P), F)$ with $F = \{(u, ad) \mid u \in d^*\}$, and

$$P = \{\langle e \xrightarrow{T} a, e{\leftrightarrow}e\rangle, \langle e \xrightarrow{T} b, a{\leftrightarrow}e\rangle, \langle b \xrightarrow{T} a, e{\leftrightarrow}e\rangle, \langle b \xrightarrow{T} c, e{\leftrightarrow}e\rangle,$$
$$\langle c \xrightarrow{T} d, e{\leftrightarrow}a\rangle, \langle a \xrightarrow{T} c, d{\leftrightarrow}e\rangle\}$$

In the first phase of its functioning, the system above reads a string $(ab)^n$ while sending n copies of a into the environment. Then in the second phase, as many cds are read, as the number of as that can be found in the environment. □

Next we show that if we only require that all programs contain at least one tape rule (but unlike in the restricted case, they can also use the environmental symbol e), then any recursively enumerable language can be accepted also with systems of capacity two.

Theorem 6. $\mathcal{L}(\text{genPCol}, f_{perm}, \text{all-tape}(2)) = \mathcal{L}(\text{RE})$.

Proof. Let $L \subseteq \Sigma^*$ be an arbitrary recursively enumerable language, and let $M = (\Sigma \cup \{Z, B\}, Q, q_0, q_f, Tr)$ be a two-counter machine with $L = L(M)$, as defined in section 2.

Construct the genPCol automaton $\Pi = (V, e, w_E, (w_0, P_0), \ldots, (w_n, P_n), F)$ of capacity two, where $V = \Sigma \cup Q \cup \{t, t', t'', t''' \mid t \in Tr\} \cup \{c_1, c_2, A\}$, the initial contents of the cells are $w_0 = q_0 e$, $w_i \in \{ee, te, t''e\}$, $1 \leq i \leq n$, as we will specify later, and $F = \{(u, q_f e, w_1, \ldots, w_n) \mid u \in V^*\}$.

For any $\alpha \in \{B, Z\}$, $\beta \in \{-1, 0, +1\}$, we define the disjoint sets of transitions $Tr_{\alpha,\beta} \subseteq Tr$ as follows: $t \in Tr_{\alpha,\beta}$, if and only if, $t : (q, x, i, \alpha) \to (q', \beta)$, $x \in \Sigma \cup \{\varepsilon\}$, $i \in \{1, 2\}$. Thus, $Tr = Tr_{B,-1} \cup Tr_{B,0} \cup Tr_{B,+1} \cup Tr_{Z,0} \cup Tr_{Z,+1}$.

For every $t \in Tr_{B,+1}$ the proof will need three cells each, whereas for each $t \in (Tr_{B,-1} \cup Tr_{B,0} \cup Tr_{Z,0} \cup Tr_{Z,+1})$ only two cells are required, thus $n = 3k_1 + 2k_2$, where $k_1 = |Tr_{B,+1}|$ and $k_2 = |(Tr_{B,-1} \cup Tr_{B,0} \cup Tr_{Z,0} \cup Tr_{Z,+1})|$.

As the cells in the constructed system correspond to transitions of the simulated two-counter machine, in the following we will index the cells $C_i = (w_i, P_i)$ (except C_0) with two indices: the transition and an integer (the integer will be 1, 2, or 3, depending on how many cells the simulation of the given transition requires). The sets of programs are defined as follows:

Let $w_0 = q_0 e$, and let

$$P_0 = \{\langle q_0 \leftrightarrow e; e \xrightarrow{T} e\rangle, \langle e \xrightarrow{T} e; e \leftrightarrow q_f\rangle\}.$$

For every $t \in (Tr_{B,0} \cup Tr_{B,-1})$ we will have two cells. The initial contents of the first cell is $w_{t,1} = ee$, whereas the set of programs is the following:

$$P_{t,1} = \{p_{t_1} : \langle e \leftrightarrow q; r_{t_1}\rangle, p_{t_2} : \langle q \xrightarrow{T} e; r_{t_2}\rangle, p_{t_3} : \langle e \to t'; c_i \xrightarrow{T} e\rangle,$$
$$p_{t_4} : \langle t' \leftrightarrow e; e \xrightarrow{T} e\rangle, p_{t_5} : \langle e \xrightarrow{T} e; e \leftrightarrow t'''\rangle,$$
$$p_{t_6} : \langle t''' \to q'; e \xrightarrow{T} e\rangle, p_{t_7} : \langle q' \leftrightarrow e; e \xrightarrow{T} e\rangle\},$$

where r_{t_1} and r_{t_2} are the rules $e \xrightarrow{T} a$ and $a \leftrightarrow c_i$, respectively, if the transition is such that the input symbol is $x = a \in \Sigma$, otherwise if $x = \varepsilon$, then $r_{t_1} = e \xrightarrow{T} e$ and $r_{t_2} = e \leftrightarrow c_i$.

For every $t \in Tr_{B,-1}$ the initial contents of the second cell is still $w_{t,2} = ee$, but the set of programs is different:

$$P_{t,2} = \{p_{t_8} : \langle e \leftrightarrow t'; e \xrightarrow{T} e\rangle, p_{t_9} : \langle t' \rightarrow t'''; e \xrightarrow{T} e\rangle, p_{t_{10}} : \langle t''' \leftrightarrow e; e \xrightarrow{T} e\rangle\}.$$

Next, the initial contents of the first cell for every $t \in Tr_{B,+1}$ is $w_{t,1} = te$, and the set of programs is the following:

$$P_{t,1} = \{p_{t_1} : \langle t \leftrightarrow q; r_{t_1}\rangle, p_{t_2} : \langle q \xrightarrow{T} e; r_{t_2}\rangle, p_{t_3} : \langle c_i \leftrightarrow e; e \xrightarrow{T} e\rangle,$$
$$p_{t_4} : \langle e \rightarrow t'; e \xrightarrow{T} e\rangle, p_{t_5} : \langle e \overset{T}{\leftrightarrow} t; t' \leftrightarrow e\rangle\},$$

where r_{t_1} and r_{t_2} are the rules $e \xrightarrow{T} a$ and $a \leftrightarrow c_i$, respectively, if the transition is such that the input symbol is $x = a \in \Sigma$, otherwise if $x = \varepsilon$, then $r_{t_1} = e \xrightarrow{T} e$ and $r_{t_2} = e \leftrightarrow c_i$.

For every $t \in (Tr_{B,0} \cup Tr_{B,+1})$ the initial contents of the second cell is $w_{t,2} = t''e$ and the set of programs is as follows:

$$P_{t,2} = \{p_{t_8} : \langle t'' \leftrightarrow t'; e \xrightarrow{T} e\rangle, p_{t_9} : \langle t' \rightarrow c_i; e \xrightarrow{T} e\rangle, p_{t_{10}} : \langle c_i \leftrightarrow e; e \xrightarrow{T} e\rangle,$$
$$p_{t_{11}} : \langle e \rightarrow t'''; e \xrightarrow{T} e\rangle, p_{t_{12}} : \langle e \overset{T}{\leftrightarrow} t''; t''' \leftrightarrow e\rangle\}.$$

The initial contents of the third cell for every $t \in Tr_{B,+1}$ is $w_{t,3} = ee$, and the set of programs is the following:

$$P_{t,3} = \{p_{t_{13}} : \langle e \leftrightarrow t'''; e \xrightarrow{T} e\rangle, p_{t_{14}} : \langle t''' \rightarrow q'; e \xrightarrow{T} e\rangle, p_{t_{15}} : \langle q' \leftrightarrow e; e \xrightarrow{T} e\rangle\}.$$

For every $t \in (Tr_{Z,0} \cup Tr_{Z,+1})$ the initial contents of the first cell is $w_{t,1} = ee$ and the set of programs is as follows:

$$P_{t,1} = \{p_{t_1} : \langle e \leftrightarrow q; r_{t_1}\rangle, p_{t_2} : \langle q \rightarrow t'; r_{t_2}\rangle, p_{t_3} : \langle t' \leftrightarrow e; e \xrightarrow{T} t\rangle,$$
$$p_{t_4} : \langle e \overset{T}{\leftrightarrow} c_i; t \rightarrow A\rangle, p_{t_5} : \langle e \overset{T}{\leftrightarrow} t'''; t \rightarrow q'\rangle, p_{t_6} : \langle t''' \xrightarrow{T} e; q' \leftrightarrow e\rangle\},$$

where r_{t_1} and r_{t_2} are the rules $e \xrightarrow{T} a$ and $a \xrightarrow{T} e$, respectively, if the transition is such that the input symbol is $x = a \in \Sigma$, otherwise if $x = \varepsilon$, then $r_{t_1} = e \xrightarrow{T} e$ and $r_{t_2} = e \xrightarrow{T} e$.

For every $t \in Tr_{Z,0}$ the initial contents of the second cell is $w_{t,2} = ee$ and the set of programs is as follows:

$$P_{t,2} = \{p_7 : \langle e \overset{T}{\leftrightarrow} t'; e \rightarrow t'''\rangle, p_8 : \langle t''' \leftrightarrow e; t' \xrightarrow{T} e\rangle\}.$$

Last, but not least, for every $t \in Tr_{Z,+1}$ the initial contents of the second cell is $w_{t,2} = t''e$ and the set of programs is as follows:

$$P_{t,2} = \{p_{t_8} : \langle t'' \leftrightarrow t'; e \xrightarrow{T} e\rangle, p_{t_9} : \langle t' \rightarrow t'''; e \xrightarrow{T} e\rangle, p_{t_{10}} : \langle t''' \leftrightarrow e; e \xrightarrow{T} e\rangle,$$
$$p_{t_{11}} : \langle e \rightarrow c_i; e \xrightarrow{T} e\rangle, p_{t_{12}} : \langle e \xleftrightarrow{T} t''; c_i \leftrightarrow e\rangle\}.$$

The genPCol automaton Π simulates the work of the two-counter machine M by reading the input symbols with its tape programs and keeping track of the contents of the i-th counter as the number of c_i, $i \in \{1, 2\}$ objects present in the environment.

Each transition of M is simulated separately. At the first step, the 0th cell contains and object that corresponds to $q_0 \in Q$, which is then sent to the environment. The environment keeps track of the current internal state of M. One transition rule is simulated by the interplay of programs of two or three cells from Π, if and only if M changes its state from q to q' while the counter contents are also checked and modified accordingly.

A transition $t : (q, x, i, B) \rightarrow (q', -1) \in Tr_{B,-1}$ is simulated by two cells $C_{t,1}$, $C_{t,2}$ with the sets of programs $P_{t,1}$ and $P_{t,2}$. First, q enters the first cell C_{t_1} from the environment by program p_{t_1}, activating the simulation of the transition. Then c_i enters the first cell (program p_{t_2}), and by programs p_{t_3} and p_{t_4}, object t' is sent to the environment. Now the programs of the second cell C_{t_2} are activated, t' is changed to t''' and sent to the environment by the programs p_{t_8}, p_{t_9}, and $p_{t_{10}}$. Now by p_{t_5}, p_{t_6} of the first cell, t''' is changed to q' denoting the next state of M, and it is sent to the environment by p_{t_7}.

A transition $t : (q, x, i, B) \rightarrow (q', 0) \in Tr_{B,0}$ is also simulated by two cells with the sets of programs $P_{t,1}$ and $P_{t,2}$. Now the initial contents of the second cell is $t''e$. The first four steps are identical with the previous case described above. When t' appears in the environment, the programs of the second cell exchange it with t'' and send c_i and t''' to the environment by the programs p_{t_i}, $8 \leq i \leq 12$, then t''' is exchanged with the object q' (denoting the next state of M) in the environment by p_{t_5}, p_{t_6}, and p_{t_7} of the first cell.

A transition $t : (q, x, i, B) \rightarrow (q', +1) \in Tr_{B,+1}$ is simulated by three cells with the sets of programs $P_{t,1}$, $P_{t,2}$ and $P_{t,3}$. The initial contents of the three cells are te, $t''e$, and ee, respectively. In the first five steps the programs of the first cell are active, but besides the object t', this time the cell also sends c_i to the environment. When t' appears in the environment (after the application of $P_{t,5}$, the programs of the second cell take over. They are identical to the previous case, they exchange t' with t'' and send c_i and t''' to the environment as above. Finally the third cell becomes active, and t''' is exchanged with the object q' (denoting the next state of M) in the environment by $p_{t_{13}}$, $p_{t_{14}}$, and $p_{t_{15}}$.

A transition $t : (q, x, i, Z) \rightarrow (q', 0) \in Tr_{Z,0}$ is simulated by two cells with the sets of programs $P_{t,1}$ and $P_{t,2}$. The initial contents of both cells are ee. The first three steps, the first cell exchanges q to t' in the environment while the second cell remains inactive. When t' appears in the environment, the programs of the second cell exchange it with t''' in the next computational step. During this step, the first cell is either inactive, or imports an object c_i from the environment, if there is at least one such object is present there. In this later case, the transition cannot be

applied in a simulation of the two counter machine M, as the value stored the ith counter is not zero. This is reflected by program $p_{t,4}$ which introduces a a "trap object" A. If the transition is applicable in M, that is, if there is no c_i present in the environment. Then after two inactive steps, t''' is exchanged with the object q' (denoting the next state of M) in the environment by p_{t_5} and p_{t_6} of the first cell.

Finally, a transition $t : (q, x, i, Z) \to (q', +1) \in Tr_{Z,+1}$ is simulated by two cells with the sets of programs $P_{t,1}$ and $P_{t,2}$. The initial contents of the two cells are ee and $t''e$, respectively. The first three steps is identical with those in the previous case. When t' appears in the environment, the programs of the second cell exchange it with t'', send an object t''' and then an object c_i (in exchange with t'') to the environment. These are done by programs p_{t_5} and p_{t_6}. Similarly to the previous case, during the fourth computational step, the first cell is either inactive, or imports an object c_i from the environment (by program p_{t_4}, if there is at least one such object is present there). In the later case, the transition cannot be applied in a simulation as the value stored the ith counter of M is not zero (so the "trap object" A is introduced). If there is no c_i present in the environment (the transition is applicable in a simulation), then after three steps (in which the first cell is inactive), t''' is imported into the first cell, and then the object q' (denoting the next state of M) is sent to the environment by p_{t_5} and p_{t_6}, respectively.

According to these considerations, we have seen that having the object q in the environment, the genPCol automaton Π replaces it with q', and either simulates a transition of the two-counter machine M from state q to state q' (checking and adjusting the multiplicity of the objects corresponding to the counter contents accordingly), or its computation is not successful. Thus, starting with q_0 in the environment (q_0 is sent to the environment by the first program in P_0 in the very first step of Π) the genPCol automaton produces q_{acc}, the accepting state of the two-counter machine M in the environment if and only if its computation corresponds to an accepting computation of M. Having q_{acc} in the environment, Π can reach its final configuration by importing it into the cell C_0 by using the second program of P_0. □

4 Conclusions

We have studied the effect of the capacity of generalized P colony automata on their computational power using the all-tape and com-tape variants of programs used. We have shown that even with capacity one, if we do not place additional restrictions on the types of programs allowed to be used by the system, genPCol automata characterize the class of recursively enumerable languages. On the other hand, for systems with capacity three, even the use of most restrictive program types does not result in any decrease of the computational power. The most interesting cases are the ones in between these two: the restricted variants of capacity one and capacity two. These require further study, especially interesting would be

to refine the relationship of the model with P automata, as they are very closely related, but not as similar as one might expect at the first glance.

References

1. L. Cienciala, L. Ciencialová, P Colonies and Their Extensions. In: J. Kelemen, A. Kelemenová (eds.), *Computation, Cooperation, and Life. Essays Dedicated to Gheorghe Păun on the Occasion of His 60th Birthday.* LNCS 6610, Springer Berlin Heidelberg, 2011, 158–169.
2. L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová, Gy. Vaszil, Variants of P colonies with very simple cell structure. *International Journal of Computers Communication and Control*, **4** (2009), 224–233.
3. L. Cienciala, L. Ciencialová, E. Csuhaj-Varjú, Gy. Vaszil, *PCol automata: Recognizing strings with P colonies.* In: M. A. Martínez del Amor, Gh. Păun, I. Pérez Hurtado, A. Riscos Nuñez (eds.), Eighth Brainstorming Week on Membrane Computing, Sevilla, February 1-5, 2010, Fénix Editora, 2010, 65–76.
4. E. Csuhaj-Varjú, J. Dassow, On cooperating/distributed grammar systems. *Journal of Information Processing and Cybernetics* EIK, **26** (1990), 49–63.
5. E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems – A Grammatical Approach to Distribution and Cooperation.* Gordon and Breach, London, 1994.
6. E. Csuhaj-Varjú, M. Oswald, Gy. Vaszil, P automata. In [17], chapter 6, 144–167.
7. E. Csuhaj-Varjú, Gy. Vaszil, P automata or purely communicating accepting P systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (eds.), *Membrane Computing. International Worskhop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers.* LNCS 2597, Springer Berlin Heidelberg, 2003, 219–233.
8. E. Csuhaj-Varjú, Gy. Vaszil, P automata with restricted power. *International Journal of Foundations of Computer Science*, **25** (2014), 391–408.
9. P.C. Fischer, Turing machines with restricted memory access. *Information and Control*, **9** (1966), 364–379.
10. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez, On the power of P and dP automata. *Annals of Bucharest University, Mathematics-Informatics Series* **63** (2009), 5-22.
11. K. Kántor and Gy. Vaszil Generalized P Colony Automata *Journal of Automata, Languages and Combinatorics* **19** (2014), 145–156.
12. J. Kelemen, A. Kelemenová, A grammar-theoretic treatment of multiagent systems. *Cybernetics and Systems*, **23** (1992), 621–633.
13. J. Kelemen, A. Kelemenová, Gh. Păun, Preview of P colonies: A biochemically inspired computing model. In: M. Bedau et al. (eds.), *Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX).* Boston Mass., 2004, 82–86.
14. A. Kelemenová, P Colonies. In [17], chapter 23.1, 584–593.
15. M.L. Minsky, *Computation: Finite and Infinite Machines.* Prentice Hall, 1967.
16. A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport. *New Generation Computing* **20**(3) (2002), 295-306.
17. Gh. Păun, G. Rozenberg, A. Salomaa, editors, *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.

A Toolbox for Simpler Active Membrane Algorithms*

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

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{leporati,luca.manzoni,mauri,porreca,zandron}@disco.unimib.it

Summary. We show that recogniser P systems with active membranes can be augmented with a priority over their set of rules and any number of membrane charges without loss of generality, as they can be simulated by standard P systems with active membranes, in particular using only *two* charges. Furthermore, we show that more general accepting conditions, such as sending out several, possibly contradictory results and keeping only the first one, or rejecting by halting without output, are also equivalent to the standard accepting conditions. The simulations we propose are always without significant loss of efficiency, and thus the results of this paper can hopefully simplify the design of algorithms for P systems with active membranes.

1 Introduction

P systems with active membranes [10] have been extensively investigated as computing devices, both from the computability and the computational complexity standpoints.

By analysing the algorithms for P systems with active membranes described in the literature, it is possible to identify a number of useful and recurring techniques or "design patterns". A standard one is using elementary membrane division to produce all assignments of a set of variables $x_1, \ldots, x_n$ [10]; the results of evaluating a Boolean formula under those assignments can then be combined in several ways:

- by disjunction, allowing the solution of the SAT problem, and thus all **NP**-complete problems [15];

* This work was partially supported by Fondo d'Ateneo (FA) 2015 of Università degli Studi di Milano-Bicocca: "Complessità computazionale e applicazioni crittografiche di modelli di calcolo bioispirati".

- by counting the number of satisfying assignments against a threshold, allowing the solution of counting problems in the class **PP** [12];
- by alternating disjunctions and conjunctions by means of a tree-shaped membrane structure of depth n, allowing the solution of **PSPACE**-complete problems [14].

Other techniques involve simulating register machines [4] or Turing machines [2], also in their nondeterministic version, by simulating nondeterminism with parallelism as above for solving **NP**-complete problems [6]. Membranes at different nesting levels can also be employed as "subroutines", simulating multiple Turing machines and becoming functionally equivalent to oracles for subproblems [6].

While the main ideas behind those constructions are generally straightforward and show clear affinity with techniques from the theory of traditional computing devices, their implementation unfortunately often involves a number of technical details which obfuscate the big picture. One of the main culprits are the ubiquitous timer objects, which keep the different parts of the P system synchronised and allow the halting of the computation immediately after producing the output, a condition that is usually imposed by the definition of *recogniser P systems* and that often requires extra work to be met.

One of the crucial aspects of the definition of P systems with active membranes is the number of possible membrane charges, which is 3 in the original definition. Although charges are not needed to solve **PSPACE**-complete problems in polynomial time[2] [3] and two charges suffice to achieve universality [1], having access to a number of charges growing with the size of the input allows a simpler implementation of many algorithms. For instance, when this is allowed, the simulation of bounded-tape Turing machines becomes trivial [6]. In that paper, an arbitrary number of charges was reduced to three without loss of efficiency, but only in a very restrictive set of circumstances (essentially, no communication with adjacent membranes is allowed, and the membrane must behave deterministically).

The purpose of this paper is twofold. On the one hand, we want to understand which features of recogniser P systems with active membranes are actually essential to characterise their behaviour. On the other hand, we want to provide an array of useful extensions which can be added to P systems with active membranes but can be simulated by the original model without loss of efficiency. This will hopefully reduce the amount of "boilerplate code" (repetitive rules unrelated to the main algorithm) in proofs and allow focusing on a higher-level description of P systems, such as dividing membranes working in parallel and their communication patterns.

The formally redundant but convenient features we describe in this paper are the ability to use any number of charges, any partial priority ordering of rules (as in the original definition of transition P systems [9]), and the ability

[2] However notice that, in the absence of membrane dissolution rules, the lack of charges seems to reduce the efficiency of P systems [5].

to output the result of the computation in less restrictive ways, such as not requiring the P system to halt after having sent out the result, or rejecting by halting without output. Furthermore, we show that all these enhancements can be simulated efficiently by standard recogniser P systems with active membranes using only *two* charges (even when working in super-polynomial time).

2 Basic notions

We recall the formal definition of P systems with active membranes using weak non-elementary division rules [10, 16].

Definition 1. *A* P system with active membranes with weak non-elementary division rules *of initial degree* $d \geq 1$ *is a tuple*

$$\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$$

where:

- Γ *is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted* unordered *tree, usually represented by nested brackets) consisting of* d *membranes labelled by elements of* Λ *in a one-to-one way;*
- $w_{h_1}, \ldots, w_{h_d}$*, with* $h_1, \ldots, h_d \in \Lambda$*, are strings over* Γ*, describing the initial multisets of objects placed in the* d *regions of* μ*;*
- R *is a finite set of rules.*

Each membrane possesses, besides its label and position in μ, another attribute called *electrical charge*, which can be either neutral (0), positive (+) or negative (−) and is always neutral before the beginning of the computation.

The rules in R are of the following types:

(a) *Object evolution rules*, of the form $[a \to w]_h^\alpha$
They can be applied inside a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by the objects in w).

(b) *Send-in communication rules*, of the form $a\,[\,]_h^\alpha \to [b]_h^\beta$
They can be applied to a membrane labelled by h, having charge α and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the charge of h is changed to β.

(c) *Send-out communication rules*, of the form $[a]_h^\alpha \to [\,]_h^\beta\, b$
They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the charge of h becomes β.

(d) *Dissolution rules*, of the form $[a]_h^\alpha \to b$
They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the membrane is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b.

(e) *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\, [c]_h^\gamma$
They can be applied to a membrane labelled by h, having charge α, containing an occurrence of the object a but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes having label h and charges β and γ; the object a is replaced, respectively, by b and c, while the other objects of the multiset are replicated in both membranes.

(f') *Weak non-elementary division rules, of the form* $[a]_h^\alpha \to [b]_h^\beta\, [c]_h^\gamma$
They can be applied to a membrane labelled by h, having charge α, and containing an occurrence of the object a, even if it contains further membranes; the membrane is divided into two membranes having label h and charges β and γ; the object a is replaced, respectively, by b and c, while the rest of the contents (including whole membrane substructures) is replicated in both membranes.

The instantaneous *configuration* of a membrane consists of its label h, its charge α, and the multiset w of objects it contains at a given time. It is denoted by $[w]_h^\alpha$. The *(full) configuration* $\mathcal{C}$ of a P system Π at a given time is a rooted, unordered tree. The root is a node corresponding to the external environment of Π, and has a single subtree corresponding to the current membrane structure of Π. Furthermore, the root is labelled by the multiset located in the environment, and the remaining nodes by the configurations $[w]_h^\alpha$ of the corresponding membranes.

A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules: inside each membrane, several evolution rules can be applied simultaneously.
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). Analogously, each membrane can only be subject to one communication, dissolution, or division rule (types (b)–(f')) per computation step; these rules will be called *blocking rules* in the rest of the paper. In other words, the only objects and membranes that do not evolve

are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.
- In each computation step, all the chosen rules are applied simultaneously (in an atomic way). However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of microsteps as follows. First, all evolution rules are applied inside the elementary membranes, followed by all communication, dissolution and division rules involving the membranes themselves; this process is then repeated to the membranes containing them, and so on towards the root (outermost membrane). In other words, the membranes evolve only after their internal configuration has been updated. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; this way, the objects that are duplicated during the division are already the final ones.
- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ of configurations, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable from $\mathcal{C}_i$ via a single computation step, and no rules of Π are applicable in $\mathcal{C}_k$. A *non-halting* computation $\vec{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted.

P systems can be used as language *recognisers* by employing two distinguished objects yes and no: we assume that all computations are halting, and that either object yes or object no (but not both) is sent out from the outermost membrane, and only in the last computation step, in order to signal acceptance or rejection, respectively. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. If this is not necessarily the case, then we have a *non-confluent* P system, and the overall result is established as for nondeterministic Turing machines: it is acceptance iff an accepting computation exists.

In order to solve decision problems (or, equivalently, decide languages), we use *families* of recogniser P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x deciding the membership of x in a language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for inputs of any length, as discussed in detail in [7].

Definition 2. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is* (polynomial-time) uniform *if the mapping $x \mapsto \Pi_x$ can be computed by two polynomial-time deterministic Turing machines E and F as follows:*

- *$F(1^n) = \Pi_n$, where n is the length of the input x and Π_n is a common P system for all inputs of length n with a distinguished input membrane.*

- $E(x) = w_x$, *where* w_x *is a multiset encoding the specific input* x.
- *Finally,* Π_x *is simply* Π_n *with* w_x *added to its input membrane.*

The family $\boldsymbol{\Pi}$ *is said to be (polynomial-time) semi-uniform if there exists a single deterministic polynomial-time Turing machine* H *such that* $H(x) = \Pi_x$ *for each* $x \in \Sigma^\star$.

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [7] for further details on the encoding of P systems.

In this paper we also take advantage of *rule priorities*, as in the original paper introducing P systems [9]. A priority is any partial order $\preceq$ of the set of rules such that, whenever a conflict between rules arises, only those with higher priority can be applied; as usual, when two rules are incomparable with respect to $\preceq$, any conflict is resolved via a nondeterministic choice. Furthermore, we also allow *generalised charges*, that is, any set $\Psi \supseteq \{+, 0, -\}$ of charges may be used [6]. Rule priorities and generalised charges will be proved redundant in Section 4.

3 Generalised Acceptance Conditions

In this section we propose a more flexible variant of recogniser P system, where we do not require a single output object at the last step of the computation, or even the halting of the P system itself. This allows the omission of a number of technical details from membrane computing solutions, which are sometimes unrelated to the main algorithm but are still required in order to ensure compliance to the formal definition of recogniser P systems. We prove that there is no loss of generality in using these variants of accepting condition, as it can always be simulated without significant loss of efficiency by the standard one; we show this result first for P systems with priority and generalised charges, and in a later section for standard P systems.

Definition 3. *A* generalised recogniser P system Π *is a P system employing two distinguished objects* yes *and* no *and behaving in any of the three following ways:*

1. *It sends out an instance of object* yes *from its outermost membrane before sending out any instance of object* no*; it can later send out any combination of objects* yes *and* no*, and is not required to halt.*

2. *It sends out an instance of object* no *from its outermost membrane before sending out any instance of object* yes*; it can later send out any combination of objects* yes *and* no*, and is not required to halt.*
3. *It halts without sending out neither an instance of* yes*, nor an instance of* no*.*

The P system Π is said to accept *in case 1, and to* reject *in case 2. The behaviour of 3 can be interpreted as either accepting or rejecting, according to a specified convention.*

It is trivial to observe that a standard recogniser P system [11] is a special case of generalised recogniser P system, always halting and sending out exactly an instance of yes or no and only in the last computation step. Furthermore, other acceptance conditions proposed in the literature [8] are also special cases of generalised recogniser P systems; in particular, we have *acknowledger* P systems (which accept by sending out one or more instances of yes and reject by halting without output) and *recogniser*$_{\geq 1}$ P systems (which accept by sending out one or more instances of yes and reject by sending out one or more instances of no). The only notable case not covered by the notion of generalised recogniser P systems is accepting by outputting yes while rejecting by not halting; since P systems are known to be universal [1], this acceptance condition characterises the whole class of recursively enumerable sets.

3.1 Ensuring Output on Halting

We can now show that standard recogniser P systems with priority and generalised charges solve exactly the same problems as generalised recogniser P systems using the same features with polynomial slowdown. Priority and generalised charges will then be eliminated, also without loss of efficiency, in Section 4. We begin by reducing case 3 of Definition 3 to one of the other two cases: case 2 if halting without output is interpreted as rejecting, or case 1 if it is interpreted as accepting. The idea is to have a timer located inside the outermost membrane, which is sent out as a no object if it does not receive a signal for $2d$ consecutive steps, where d is the depth of the membrane structure. This signal indicates that at least one rule was applied in the P system in the last $2d$ steps, and is propagated as an object $\clubsuit$ towards the outermost membrane.

Let Π be the generalised recogniser being simulated, and let Π' be the standard recogniser simulating it; both P systems have priority and generalised charges. The initial membrane structure of Π' is identical to that of Π. Inside each membrane, besides the original multiset, we place an instance of $\diamondsuit$ and one of $\heartsuit$; finally, the outermost membrane also contains an instance of the timer object T_d.

The rules of Π are modified in Π' so that their application is always detectable; in order to do so, we always either change the charge of a membrane where a rule was applied to a new, specific charge (for rules involving the

membranes themselves) or produce an extra object on the right-hand side (for object evolution rules). Assuming $h \in \Lambda$, $\alpha, \beta, \gamma \in \Psi$, $a, b, c \in \Gamma$, and $w \in \Gamma^\star$, the new rules are

$$\begin{array}{lll} [a \to w]_h^\alpha & \text{becomes} & [a \to w\ \clubsuit]_h^\alpha \\ a\ [\ ]_h^\alpha \to [b]_h^\beta & \text{becomes} & a\ [\ ]_h^\alpha \to [b]_h^{\widetilde{\beta}} \\ [a]_h^\alpha \to [\ ]_h^\beta\ b & \text{becomes} & [a]_h^\alpha \to [\ ]_h^{\widetilde{\beta}}\ b \\ [a]_h^\alpha \to b & \text{remains identical} & \\ [a]_h^\alpha \to [b]_h^\beta\ [c]_h^\gamma & \text{becomes} & [a]_h^\alpha \to [b]_h^{\widetilde{\beta}}\ [c]_h^{\widetilde{\gamma}} \end{array} \tag{1}$$

Here the new charges of the form $\widetilde{\alpha}$, with $\alpha \in \Psi$, encode the new charge of the membrane and the information that a rule involving that membrane was applied in the previous step. If object evolution rules were applied, then a corresponding number of objects $\clubsuit$ appear.

In membranes where no blocking rule was applied, the charge is not of the form $\widetilde{\alpha}$. In that case, the following rule (which has lower priority) is applied instead:

$$[\heartsuit]_h^\alpha \to [\ ]_h^{\alpha'}\ \# \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$

The object $\heartsuit$ is restored at each computation step by the following rules:

$$\begin{array}{ll} [\diamondsuit \to \diamondsuit\ \heartsuit]_h^\alpha & \text{for } h \in \Lambda \text{ and } \alpha \in \Psi \\ [\diamondsuit \to \diamondsuit\ \heartsuit]_h^{\widetilde{\alpha}} & \text{for } h \in \Lambda \text{ and } \alpha \in \Psi \\ [\diamondsuit \to \diamondsuit\ \heartsuit]_h^{\alpha'} & \text{for } h \in \Lambda \text{ and } \alpha \in \Psi \end{array}$$

(Notice that these rules impede the halting of the P system, but this is allowed by case 2 of Definition 3.)

Now each membrane has either a charge of the form $\widetilde{\alpha}$ or one of the form α'. This denotes that we will now perform a signal propagation step, rather than a step simulating rules of Π. All instances of $\clubsuit$, both those just created by applying rule (1) and those created in previous steps, are propagated one level up (except for the outermost membrane k) and simultaneously change all charges $\widetilde{\alpha}$ and α' to plain α:

$$[\clubsuit]_h^{\widetilde{\alpha}} \to [\ ]_h^\alpha\ \clubsuit \qquad \text{for } h \in \Lambda - \{k\} \text{ and } \alpha \in \Psi \tag{2}$$

$$[\clubsuit]_h^{\alpha'} \to [\ ]_h^\alpha\ \clubsuit \qquad \text{for } h \in \Lambda - \{k\} \text{ and } \alpha \in \Psi \tag{3}$$

The following rule, with priority lower than (2) and (3), create and immediately propagate a new signal object if a rule involving the membrane was applied and no object $\clubsuit$ was already present:

$$[\heartsuit]_h^{\widetilde{\alpha}} \to [\ ]_h^\alpha\ \clubsuit \qquad \text{for } h \in \Lambda - \{k\} \text{ and } \alpha \in \Psi$$

If a membrane has charge α' (no rule involving that membrane was applied in the previous step) and there is no ♣ to propagate, then ♡ changes the charge to α with the following rules with lower priority:

$$[\heartsuit]_h^{\alpha'} \to [\,]_h^{\alpha} \,\# \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$

Any extra occurrence of ♡ is always deleted by the following rules, which have minimal priority:

$$[\heartsuit \to \epsilon]_h^{\alpha} \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$
$$[\heartsuit \to \epsilon]_h^{\widetilde{\alpha}} \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$
$$[\heartsuit \to \epsilon]_h^{\alpha'} \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$

Any extra occurrence of ♣ is deleted *only in the propagation steps* by the following rules with minimal priority (which are thus only enabled if the signal is already propagated from the current membrane):

$$[\clubsuit \to \epsilon]_h^{\widetilde{\alpha}} \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$
$$[\clubsuit \to \epsilon]_h^{\alpha'} \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$

The timer object T_t (with $0 \leq t \leq d$) in the outermost membrane k counts down in the simulation steps, when the charge of the that membrane is one of the original ones:

$$[T_t \to T_{t-1}]_k^{\alpha} \qquad \text{for } \alpha \in \Psi \text{ and } 0 < t \leq d$$

In order to reset the timer when a signal ♣ reaches the outermost membrane, that object changes the charge, currently of the form $\widetilde{\alpha}$ or α', to a new charge $\alpha_{\clubsuit}$, whose presence denotes that the charge of the outermost membrane of Π is α and at least one rule was applied in the last d simulated steps:

$$[\clubsuit]_k^{\widetilde{\alpha}} \to [\,]_k^{\alpha\clubsuit} \,\# \qquad \text{for } \alpha \in \Psi$$

All original rules related to the outermost membrane, which have a plain charge $\alpha \in \Psi$ on the left-hand side, must be duplicated in order to maintain the same behaviour when the left-hand charge is $\alpha_{\clubsuit}$ (the right-hand side must remain unchanged, i.e., the subscript ♣ is removed when changing the charge to one of the form $\widetilde{\beta}$ or β').

When the charge of the outermost membrane k has the form $\alpha_{\clubsuit}$, the counter is reset to d:

$$[T_t \to T_d]_k^{\alpha\clubsuit} \qquad \text{for } \alpha \in \Psi \text{ and } 0 < t \leq d$$

If, however, the timer reaches 0 while the charge of the outermost membrane k has no subscript ♣, this means that no rule of Π was simulated by the P system Π' in the last d steps. We can thus assume that Π has halted, and send out the timer as a no object:

$$[T_0]_k^{\alpha} \to [\ ]_k^{\widetilde{\alpha}}\ \mathsf{no} \qquad \text{for } \alpha \in \Psi$$

If, on the other hand, halting without output is interpreted as accepting, we send out a yes object instead:

$$[T_0]_k^{\alpha} \to [\ ]_k^{\widetilde{\alpha}}\ \mathsf{yes} \qquad \text{for } \alpha \in \Psi$$

If no object yes or no has been previously sent out, this no (or yes) object becomes the result of the computation, otherwise it does not change the previous result according to cases 1 and 2 of Definition 3.

The computation time of the P system Π' is as follows: if Π sends out a yes or no object at step t, then the same object is sent out by Π' at step $2t-1$ (the corresponding simulation step); if, on the other hand, Π rejects by halting after t steps without output, then Π' sends out a no object at time $2t-1+2d$, i.e., the time required for simulating the t steps of Π, plus the time required to propagate the signal from the deepest membrane and the time for a last timer cycle, before the final output step.

Discutere dissoluzione

3.2 Ensuring Halting on Output

Having reduced case 3 to case 2 of Definition 3, we still need to ensure that a single output object is sent out of the P system, and only in the last computation step in cases 1 and 2, in order to prove that each generalised recogniser can be replaced by a standard recogniser without significant loss of efficiency. We can further ensure that the all membranes of the simulating system have a new, distinguished charge ♠ with no associated rules in the last configuration, denoting that the P system is halting; this technical detail will prove useful in Section 4.

Let Π be a generalised recogniser P system which always produces output (i.e., accept either by case 1 or 2) but not necessarily a unique output, and that does not necessarily halt. We design a recogniser P system Π' satisfying the requirements above.

The initial configuration of Π' is exactly the same as Π, except that each membrane contains as many instances of the new object ♠ as the number of its children membranes, and the outermost membrane contains an instance of the new object R_d. The rules and the alphabet of Π' include all those of Π, except as described below.

The P system Π' executes all rules of Π, with the same priority, except for those sending out the result of the computation from the outermost membrane, while simultaneously doubling the amount of ♠ contained inside each membrane by using the following rules, which are only enabled by the original charges of Π:

$$[\spadesuit \to \spadesuit\ \spadesuit]_h^{\alpha} \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi \tag{4}$$

Notice that these rules do not compete with the original rules of Π, since they are object evolution rules.

When Π sends out the result object yes or no from the outermost membrane k by means of a rule of the form

$$[a]_k^\alpha \to [\ ]_k^\beta \,\mathsf{yes} \qquad\qquad [a]_k^\alpha \to [\ ]_k^\beta \,\mathsf{no}$$

the P system Π' applies instead a rule of the form

$$[a]_k^\alpha \to [\ ]_k^{\mathsf{yes}} \,\# \qquad\qquad [a]_k^\alpha \to [\ ]_k^{\mathsf{no}} \,\# \tag{5}$$

These update rules maintain the same priority as the original ones.

The final result of the computation is thus temporarily stored in the charge of the outermost membrane, and a "junk" object is sent out instead. Notice that, since the charges yes and no are new, the objects of the original alphabet Γ of Π cannot apply any rule inside the outermost membrane. The other membranes might continue computing; we now propagate the information about having produced output towards the internal membranes in order to stop the computation.

Notice that the number of objects ♠ has always been kept at least equal to the number of children membranes during the computation, even when taking membrane division into account (the membranes can at most double in number during each step). When the charge of the outermost membrane of Π' becomes yes or no, the rules (4) becomes disabled for the outermost label, and the following rules *with priority lower than* (4) *but higher than the simulated rules of* Π become now applicable:

$$♠\,[\ ]_h^\alpha \to [\#]_h^♠ \qquad\qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi \tag{6}$$

The charge of each children membrane thus changes to ♠. Notice that the objects ♠ in excess of the number of children membrane become inert, since all their rules are now disabled (all reachable membranes now having charge ♠). The new charge ♠ also disables the rules of type (4) for membrane h, enabling those of type (6) for its children membranes. This propagates the charge ♠ to the next level, and so on.

The P system reaches a configuration where all membranes, except the outermost one, have charge ♠ exactly d steps after applying one of the rules in (5). The timer R_d inside the outermost membrane k, also enabled when rule (5) is applied, counts these d steps, using the rules

$$[R_i \to R_{i-1}]_k^{\mathsf{yes}} \qquad [R_i \to R_{i-1}]_k^{\mathsf{no}} \qquad \text{for } 0 < i \leq d$$

When reaching zero, the object R_0 is finally sent out as the result of the computation while setting the charge of the remaining membrane to ♠:

$$[R_0]_k^{\mathsf{yes}} \to [\ ]_k^♠ \,\mathsf{yes} \qquad\qquad [R_0]_k^{\mathsf{no}} \to [\ ]_k^♠ \,\mathsf{no}$$

Notice that Π' has exactly the same number of computations as Π and with the same result; indeed, the new rules do not interfere with the simulation of Π while this is still running, and the last phase, where all charges become ♠, is deterministic. Furthermore, if Π sends out its first result object at time t, then Π' sends out the same result *and halts* at time $t + d + 1$.

By combining the results of Sections 3.1 and 3.2 we obtain:

Lemma 1. *Let Π be a confluent (resp., non-confluent) generalised recogniser P system with priority and generalised charges working in time t. Then, there exists a standard confluent (resp., non-confluent) recogniser P system with priority and generalised charges having the same result and working in time $O(t + d)$, where d is the depth of both P systems.* □

By using a variant of the proof techniques of Section 3.1 and 3.2 it is possible to employ even other accepting conditions. For instance, it is possible to keep the *last* output object (before halting) as the result of the computation, rather than the first one, by storing the last of the sequence of output objects in the charge of the outermost membrane, but only outputting it when the original P system halts. Even more generally, we can collect the sequence of output objects and combine them by applying any computable function (exploiting the universality of P systems).

4 Charges and Priority

We will now show how confluent P systems Π with priority and any number of charges can be efficiently simulated by confluent P systems Π' without priority and using only two charges. The idea is to give a total ordering of the set of rules of Π compatible with its original priority, say $r_1 \succ r_2 \succ \cdots \succ r_m$; we decompose each computation step of Π into m micro-steps, each one applying exactly one rule in the whole system as much as possible. The computation is thus sequential across the set of rules, but each rule r_i is applied in a maximally parallel way in all membranes involved in r_i. Halting in Π' is triggered by the halting of Π, assuming that each membrane of the latter system has charge ♠, as proved possible in Section 3.2.

Notice that a linear priority does not make the P system Π deterministic, since send-in rules choose an arbitrary membrane among a set of different but externally indistinguishable ones having the same label (this will be the only form of nondeterminism for Π with priority $\succ$ and thus for Π'). On the other hand, using a total priority ordering of the rules requires, in general, the simulated P system Π to be confluent, since only a subset of its computations are simulated by Π'. Non-confluent P systems Π can be simulated using our construction if they already have a total priority ordering (in that case, the simulating P system Π' is also non-confluent).

The membrane structure of Π' is, once again, identical to that of Π. A configuration $\mathcal{C}$ at time t of Π is encoded as a configuration $\mathcal{C}'$ at time *something*

of Π' as follows: if $\mathcal{C}$ contains a membrane having configuration $[w]^{\alpha}_h$, then the corresponding membrane in $\mathcal{C}'$ has configuration $[w\ \alpha]^0_h$, that is, the original charge is encoded as an object in $\mathcal{C}'$. We can view this as an invariant maintained by the simulation for all time steps t of Π. Notice it is trivial to recover the original configuration $\mathcal{C}$ from $\mathcal{C}'$ (and vice versa).

Simulating each step of Π begins with an initialisation phase of four steps of Π'. First we rewrite the charge-object α as $\alpha'\ \oplus$:

$$[\alpha \to \alpha'\ \oplus]^0_h \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$

Each membrane of Π' has now the configuration $[w\ \alpha'\ \oplus]^0_h$. The object $\oplus$ changes the charge of the membrane to $+$ (it will always have this behaviour in the rest of the paper), while α' is rewritten into $\alpha''\ \odot$:

$$[\oplus]^0_h \to [\ ]^+_h\ \# \qquad \text{for } h \in \Lambda$$
$$[\alpha' \to \alpha''\ \odot]^0_h \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$

This leads to the configuration $[w\ \alpha''\ \odot]^+_h$. The object $\odot$ changes the charge to 0 (here and in the rest of the paper), while the objects in the original alphabet Γ gain a prime; the object α'' is rewritten into $\alpha'''\ \bullet$, where $\bullet$ is $\odot$ if rule r_1 has membrane h and charge α on the left-hand side, and $\bullet$ is $\oplus$ otherwise:

$$[\odot]^+_h \to [\ ]^0_h\ \# \qquad \text{for } h \in \Lambda$$
$$[a \to a']^+_h \qquad \text{for } h \in \Lambda \text{ and } a \in \Gamma$$
$$[\alpha'' \to \alpha'''\ \bullet]^+_h \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$

The current configuration is thus $[w'\ \alpha'''\ \bullet]^0_h$, where w' is w with all objects primed. The object $\bullet$ is then sent out, setting the charge of h to $+$ (if $\bullet$ is $\oplus$) or 0 (if $\bullet$ is $\odot$), while all remaining objects take a subscript 1:

$$[\odot]^0_h \to [\ ]^0_h\ \# \qquad \text{for } h \in \Lambda$$
$$[\alpha''' \to \alpha_1]^0_h \qquad \text{for } h \in \Lambda \text{ and } \alpha \in \Psi$$
$$[a' \to a_1]^0_h \qquad \text{for } h \in \Lambda \text{ and } a \in \Gamma$$

This leads either to configuration $[w_1\ \alpha_1]^0_h$ or $[w_1\ \alpha_1]^+_h$, depending on whether rule r_1 has the right label and charge on the left-hand side.

We now establish a second invariant: for each $1 \le i \le m$, we have four possible forms of configurations at time something:

1. $[w_i\ \alpha_i]^0_h$ denotes that we have already tried to apply rules $r_1, \ldots, r_{i-1}$ in sequence (each of them in a maximally parallel way), but no blocking rule for h has been applied during the simulation of the current step of Π. The membrane contains the multiset of objects w_i, where each object has subscript i, and the charge of the simulated membrane is α. Furthermore, rule r_i has label h and charge α on the left-hand side.

2. $[w_i\ \alpha_i]_h^+$ is as in 1, but rule r_i has either the wrong label or the wrong charge on the left-hand side.
3. $[w_i\ \alpha_{i,j}]_h^0$ is as in 1, but a blocking rule r_j for h, for some $j < i$, has been previously applied during the simulation of the current step of Π, and r_i is necessarily an object evolution rule.
4. $[w_i\ \alpha_{i,j}]_h^+$ is as in 1, but a blocking rule r_j for h, for some $j < i$, has been previously applied during the simulation of the current step of Π, rule r_i is either an object evolution rule with wrong label or charge on the left-hand side, or it is any blocking rule.

Let us consider the four types of possible configuration separately.

4.1 Configuration of the Form $[w_i\ \alpha_i]_h^0$

The behaviour of the P system Π' when the configuration at time whatever is $[w_i\ \alpha_i]_h^0$ depends on the type of rule r_i of Π to be simulated.

Applicable Send-Out Rules

Suppose that r_i is a send-out rule of Π of the form $[a]_h^\alpha \to [\ ]_h^\beta\ b$; also suppose that the simulated membrane h contains at least one instance of a, i.e., that the configuration of the membrane in Π' is $[a_i\ v_i\ \alpha_i]_h^0$ for some multiset v_i. The rule is implemented by first sending out an object a_i as $\widetilde{b}_i$; the tilde here denotes that that instance of object b_i has already been subject to a rule during this simulated step of Π. The charge of the membrane is also changed to $+$ in order to signal that rule r_i was actually applied (i.e., that the membrane contained at least one instance of a_i):

$$[a_i]_h^0 \to [\ ]_h^+\ \widetilde{b}_i \tag{7}$$

At the same time, the object α_i is primed:

$$[\alpha_i \to \alpha_i']_h^0$$

The configuration of the membrane is now $[u_i\ \alpha_i']_h^+$, where u_i is v_i with any extra objects received from children membranes[3], and the object $\widetilde{b}_i$ is now managed by the outer membrane. When the membrane becomes positive, each original object of Γ (possibly in a tilded version) gains a prime, while α_i' becomes $\alpha_{i,i}''$, storing in its second subscript the index i of the blocking rule that has actually been applied:

[3] This is not actually possible with standard P systems, since the children of a membrane always have a different label, and thus cannot apply any rule while r_i is being applied. However, we will prove later that we can replace labels by charges; therefore, we will consider this case anyway.

$$
\begin{aligned}
&[c_i \to c_i']_h^+ && \text{for } c \in \Gamma \\
&[\widetilde{c_i} \to \widetilde{c_i'}]_h^+ && \text{for } c \in \Gamma \\
&[\alpha_i' \to \alpha_{i,i}'']_h^+ && (8)
\end{aligned}
$$

This leads us to the configuration $[u_i' \; \alpha_{i,i}'']_h^+$, where u_i' is u_i with all objects primed. The object $\alpha_{i,i}''$ is now rewritten as follows:

$$[\alpha_{i,i}'' \to \alpha_{i,i}''' \odot]_h^+$$

The configuration is now $[u_i' \; \alpha_{i,i}''' \odot]_h^+$. The object $\odot$ sets the charge to 0:

$$[\odot]_h^+ \to [\,]_h^0 \; \#$$

while $\alpha_{i,i}'''$ produces $\bullet$, where $\bullet$ is $\odot$ if rule r_{i+1} is an evolution rule with label h and charge α (i.e., r_{i+1} is potentially applicable), and $\bullet$ is $\oplus$ otherwise (i.e., r_{i+1} is not applicable due to the label, the charge, or the membrane h having already been used):

$$[\alpha_{i,i}''' \to \alpha_{i,i}'''' \bullet]_h^+$$

The configuration is thus $[u_i' \; \alpha_{i,i}'''' \bullet]_h^0$. In the last step, we need to increase the rule counter i to $i+1$, remove all primes, and update the charge of h according to $\bullet$:

$$
\begin{aligned}
&[\alpha_{i,i}''' \to \alpha_{i+1,i}]_h^0 \\
&[c_i' \to c_{i+1}]_h^0 && \text{for } c \in \Gamma \\
&[\widetilde{c_i'} \to \widetilde{c}_{i+1}]_h^0 && \text{for } c \in \Gamma \\
&[\odot]_h^0 \to [\,]_h^0 \; \# \\
&[\oplus]_h^0 \to [\,]_h^+ \; \#
\end{aligned}
$$

We have thus reached the configuration $[u_{i+1} \; \alpha_{i+1,i}]_h^0$ or $[u_{i+1} \; \alpha_{i+1,i}]_h^+$ after 5 steps of Π', thus restoring the invariant.

Applicable Send-In Rules

If r_i is a send-in rule $a \; [\,]_h^\alpha \to [b]_h^\beta$ and the outer membrane contains an instance of object a that is actually assigned to this rule for the current membrane, the computation proceeds exactly as for applicable send-out rules, in 5 steps, except that rule (7) is replaced by the symmetrical rule

$$a_i \; [\,]_h^0 \to [\widetilde{b_i}]_h^+$$

Applicable Division Rules

If r_i is a weak (elementary or non-elementary) division rule $[a]_h^\alpha \to [b]_h^\beta\ [c]_h^\gamma$ and membrane h contains an instance of a, then the computation evolves again as for applicable send-out rules, in 5 steps, with the following variations. Rule (7) is replaced by the send-out rule

$$[a_i]_h^0 \to [\ ]_h^+ \#$$

This rule sets the charge to +, thus signalling that the rule is actually being applied. This fact is recorded by the object α_i' using rule (8); the actual division does not happen immediately, but is delayed until the end of the iteration across all rules. The reason for this delay is to comply with the usual semantics of P systems, where internal membranes logically evolve before external ones: if division happened immediately, the internal membranes may evolve differently in the two copies of the membrane, due to the nondeterminism possibly introduced by send-in rules.

Applicable Dissolution Rules

A dissolution rule $r_i = [a]_h^\alpha \to b$ is simulated in 5 steps as a send-out rule followed by a delayed dissolution; this is recorded, as for division rules, in the second subscript of the object $\alpha''_{i,i}$. The actual dissolution is delayed because further object evolution rules might be applicable.

Object Evolution Rules

An object evolution rule $r_i = [a \to x]_h^\alpha$, with $x \in \Gamma^\star$, is simulated in a slightly different way than blocking rules: since they are applied in parallel to all objects a contained in h, this rule cannot change the charge of the membrane to signal its application. The object α_i must thus evolve without knowing if and to how many objects the rule r_i is being applied.

In the first step the actual evolution occurs:

$$\begin{aligned} &[a_i \to \widetilde{x}_i]_h^0 \\ &[\alpha_i \to \alpha_i']_h^0 \end{aligned} \tag{9}$$

where $\widetilde{x}_i$ is x with all objects tilded and subscripted by i. This leads to the configuration $[v_i\ \alpha_i']_h^0$, where v_i is the multiset w_i updated according to rule (9). In the second step the following rule is applied:

$$[\alpha_i' \to \alpha_i''\ \oplus]_h^0$$

leading to configuration $[v_i\ \alpha_i''\ \oplus]_h^0$. The charge is then set to +:

$$\begin{aligned} &[\oplus]_h^0 \to [\ ]_h^+ \# \\ &[\alpha_i'' \to \alpha_i'''\ \odot]_h^0 \end{aligned}$$

leading to $[v_i\ \alpha_i'''\ \odot]_h^+$. The objects in Γ and their tilded versions are now primed, while the charge becomes 0:

$$\begin{array}{ll}
[c_i \to c_i']_h^+ & \text{for } c \in \Gamma \\
[\widetilde{c}_i \to \widetilde{c}_i']_h^+ & \text{for } c \in \Gamma \\
[\odot]_h^+ \to [\ ]_h^0\ \# & \\
[\alpha_i''' \to \alpha_i''''\ \bullet]_h^+ &
\end{array}$$

where $\bullet$ works as described above. This leads to the configuration $[v_i'\ \alpha_i''''\ \bullet]_h^0$. The last step is as for applicable send-out rules, and leads to $[v_{i+1}\ \alpha_{i+1}]_h^0$ or $[v_{i+1}\ \alpha_{i+1}]_h^+$ depending on r_{i+1}.

Non-Applicable Blocking Rules

If r_i is a blocking rule with label h and charge α, but the object on the left-hand side of the rule is missing, we reach the configuration $[v_i\ \alpha_i']_h^0$ after one step, where v_i is w_i except for any objects coming from or sent in children membranes. The object α_i' detects that rule r_i was not applied by observing the neutral charge of the membrane. The membrane can then evolve as for object evolution rules, leading to configuration $[v_{i+1}\ \alpha_{i+1}]_h^0$ or $[v_{i+1}\ \alpha_{i+1}]_h^+$ (depending on r_{i+1}) in 5 computation steps.

4.2 Configuration of the Form $[w_i\ \alpha_i]_h^+$

If the P system reaches configuration $[w_i\ \alpha_i]_h^+$, then rule r_i is not applicable either because it involves a label different from h, or a charge different from α on the left-hand side. In that case, the P system must reach configuration $[v_{i+1}\ \alpha_{i+1}]_h^0$ (or $[v_{i+1}\ \alpha_{i+1}]_h^+$ if r_{i+1} has the wrong label or charge), where v is w except for any object coming from or sent in children membranes, after exactly 5 steps, in order to keep all membranes synchronised. In this particular configuration, unlike the previous one, the objects of Γ and their tilded counterparts have their subscript incremented without first being primed.

First the object α_i waits for two steps, and then produces a $\odot$; the object α_i is also tilded to record the fact that rule r_i is not being applied at this time:

$$[\alpha_i \to \widetilde{\alpha}_i']_h^+ \qquad [\widetilde{\alpha}_i' \to \widetilde{\alpha}_i'']_h^+ \qquad [\widetilde{\alpha}_i'' \to \widetilde{\alpha}_i'''\ \odot]_h^+$$

While the charge is set to 0, the object $\bullet$ (which is either $\odot$ or $\oplus$ according to the label and charge of r_{i+1}) is produced:

$$[\odot]_h^+ \to [\ ]_h^0\ \# \qquad [\widetilde{\alpha}_i''' \to \widetilde{\alpha}_i''''\ \bullet]_h^+$$

Finally, all subscripts are incremented:

$$
\begin{aligned}
&[\widetilde{\alpha}_i'''' \to \alpha_{i+1}]_h^0 && \\
&[c_i \to c_{i+1}]_h^0 && \text{for } c \in \Gamma \\
&[\widetilde{c}_i \to \widetilde{c}_{i+1}]_h^0 && \text{for } c \in \Gamma
\end{aligned}
$$

This leads to the configuration for rule r_{i+1}.

Notice that the fact that the objects in Γ (both in plain and tilded versions) are not primed in this simulated micro-step allows them to be sent into a children membrane if rule r_i requires so.

4.3 Configuration of the Form $[w_i\ \alpha_{i,j}]_h^0$

If the configuration has the form $[w_i\ \alpha_{i,j}]_h^0$, then a blocking rule r_j, with $j < i$, has already been applied to that membrane, and r_i is thus necessarily an object evolution rule with label h and charge α. The computation then proceeds as for object evolution rules in Section 4.1, except that the second subscript j of $\alpha_{i,j}$ is also preserved, thus reaching configuration $[v_{i+1}\ \alpha_{i+1,j}]_h^0$ (or $[v_{i+1}\ \alpha_{i+1,j}]_h^+$) after 5 steps, where v is w updated according to r_i and any object coming from or sent in children membranes.

4.4 Configuration of the Form $[w_i\ \alpha_{i,j}]_h^+$

If the configuration has the form $[w_i\ \alpha_{i,j}]_h^+$, then a blocking rule r_j, with $j < i$, has already been applied to that membrane, and r_i either has the wrong label or charge, or it is another blocking rule (and thus it is not applicable). In this case, the computation proceeds as in Section 4.2, except that the second subscript j of $\alpha_{i,j}$ is preserved.

4.5 Concluding the Simulation of One Step

After having simulated all rules $r_1 \succ r_1 \succ \cdots \succ r_m$ in priority order, the subscripts of the objects inside each membrane will reach the value $m + 1$. Suppose that all membranes have been set to neutral at that time (as if the non-existing rule r_{m+1} were always applicable). In order to restore the outer invariant of Section 4, we need to remove the subscripts and the tildes, update the objects representing the charges, and completing the application of dissolution and division rules.

The objects in Γ and their tilded counterparts can be immediately rewritten into their final form:

$$
\begin{aligned}
&[c_{m+1} \to c]_h^0 && \text{for } h \in \Lambda \text{ and } c \in \Gamma \\
&[\widetilde{c}_{m+1} \to c]_h^0 && \text{for } h \in \Lambda \text{ and } c \in \Gamma
\end{aligned}
$$

If a dissolution rule r_j involving the membrane being simulated was applied during this simulated step, the actual dissolution can now take place (recall that the object b on the right-hand side of the rule has already been sent out):

$$[\alpha_{m+1,j}]_h^0 \to \#$$

If a weak (elementary or non-elementary) division rule r_j involving membrane h was applied, then we can first perform the actual division:

$$[\alpha_{m+1,j}]_h^0 \to [\alpha'_{m+1,j}]_h^0\ [\alpha''_{m+1,j}]_h^0$$

and then update the simulated charges and create the right-hand side objects in the two copies of h:

$$[\alpha'_{m+1,j} \to \beta\ b]_h^0 \qquad\qquad [\alpha''_{m+1,j} \to \gamma\ c]_h^0$$

If a blocking rule r_j of the remaining types (send-in or send-out) was applied to the membrane, then we just need to update the charge object to β, the right-hand side charge of r_j; however, this must take two steps to maintain synchronisation with membranes where division was applied:

$$[\alpha_{m+1,j} \to \alpha'_{m+1,j}]_h^0 \qquad\qquad [\alpha'_{m+1,j} \to \beta]_h^0$$

Finally, if no blocking rule was applied in membrane h, then we must keep the same charge as in the previous step of Π; once again, this must take two steps to maintain all membranes synchronised:

$$[\alpha_{m+1} \to \alpha'_{m+1}]_h^0 \qquad [\alpha'_{m+1} \to \alpha]_h^0 \qquad \text{for } h \in \Lambda,\ \alpha \in \Psi$$

The new configuration of Π' then corresponds to a reachable configuration of Π according to the encoding described above.

4.6 Halting and Output

In the simulation of this section, detecting whether a membrane of Π has stopped computing paradoxically requires us to iterate across all rules, thus preventing the simulating P system Π' to halt. However, according to the results of Section 3, we can always assume the simulated P system Π to be a standard recogniser, and that all membranes assume charge ♠ when they stop computing. Thus, we simulate each membrane as described above until it assumes the charge ♠. When this happens, the simulating membrane contains the object ♠; however, by construction the children of this membrane, if any, have not yet assumed the charge ♠, as this will propagate there by send-in in the next computation step. Hence, when a simulated membrane reaches charge ♠, we must perform a last iteration across the rules of Π in order to simulate those send-in rules, and then the simulating membrane can finally halt. This last iteration is needed in order to update the subscripts of the objects c_i (with $c \in \Gamma$). Another small modification to be made involves sending out the yes or no object: in Π, the corresponding rules do not generally have the lowest priority, and thus the actual sending out of the result object

must be delayed in order to be the last action performed by the simulating P system Π'.

Halting the simulation of a membrane can thus be performed by simply deleting the object $\spadesuit_{m+1}$ obtained after the last iteration across the rules:

$$[\spadesuit_{m+1} \to \epsilon]_h^0 \qquad \text{for } h \in \Lambda \tag{10}$$

The outputting of yes or no by Π' at the last step can be achieved by replacing any outermost membrane output rules r_j of the forms

$$[a]_k^\alpha \to [\ ]_k^\beta \text{ yes} \qquad\qquad [a]_k^\alpha \to [\ ]_k^\beta \text{ no}$$

of Π by a rule of the form

$$[a]_k^\alpha \to [\ ]_k^\beta \# \qquad\qquad [a]_k^\alpha \to [\ ]_k^\beta \#$$

During the subscript-deleting phase of Section 4.5 we can perform the actual output by using one of the following rules:

$$[\alpha_{m+1,j}]_k^0 \to [\ ]_k^0 \text{ yes} \qquad\qquad [\alpha_{m+1,j}]_k^0 \to [\ ]_k^0 \text{ no} \tag{11}$$

Since, by hypothesis, the rest of the P system Π has already halted, the simulation of Π' in the worst case completes the last iteration across the rules of Π for the innermost membranes by applying a rule of type (10) exactly when an output rule (11) is applied. The P system Π' halts immediately after.

4.7 Main Result

The only remaining detail to consider is the amount of resources needed in order to perform the simulations described in this section and in Section 3. It suffices to observe that all rules of the final P system are obtained by repeating simple patterns with parameters ranging over sets of polynomial size with respect to the description of the simulated P system (e.g., the set of rules of the original P system, its set of labels, the set of integers up to the membrane nesting dept, ...).

For example, the rules of type (6) can be output by using two nested loops as follows:

for $h \in \Lambda$ **do**
  for $\alpha \in \Psi$ **do**
    output "$\spadesuit\ [\ ]_h^\alpha \to [\#]_h^\spadesuit$"
  end
end

The construction of Π' can thus be performed in polynomial time. This leads immediately to our main result:

Theorem 1. *Let Π be a generalised confluent recogniser P system using priority and generalised charges working in time t. Then, there exists a standard confluent recogniser P system Π' without priority and using only two charges having the same result as Π and working in time $O(r \times (d+t))$, where r is the number of rules of Π and d its depth. Furthermore, the mapping $\Pi \mapsto \Pi'$ can be computed in polynomial time with respect to the length of the description of Π.* □

4.8 A Note on Rule Types

The construction used to prove Theorem 1 necessarily requires evolution, send-in and send-out rules. Any other type of rule (dissolution, elementary and weak non-elementary division) is only necessary if the original P system Π being simulated employs it. This construction might, in principle, be extended in order to simulate other kinds of rules, provided that the simulating P system Π' is also allowed to use them. The technical details are, however, necessarily dependent on the specific type of rule.

5 Conclusions

The results of this paper show that the number of charges (as long as it is at least two) and the exact accepting conditions of recogniser P systems with active membranes are immaterial, and can always be reduced to two charges and to the standard definition of recogniser without loss of efficiency. This allows us to use as many charges as are convenient for the solution of the current problem, to employ more relaxed halting conditions, and even to add a rule priority. Hopefully, these tools will yield algorithms having less irrelevant technical details and a better focus on the novel techniques and ideas employed.

We conjecture that results analogous to those presented in this paper may also be proved for other classes of P systems, therefore further simplifying membrane computing algorithms. For instance, it would be interesting to explore which features (such as charges, rule priorities and accepting conditions) may be added to tissue P systems [13] without changing their computing power or efficiency.

References

1. Alhazov, A., Freund, R., Riscos-Núñez, A.: One and two polarizations, membrane creation and objects complexity in P systems. In: Zaharie, D., Petcu, D., Negru, V., Jebelean, T., Ciobanu, G., Cicortas, A., Abraham, A., Paprzycki, M. (eds.) Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC'05. pp. 385–394. IEEE (2005)

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., 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)
4. Ciobanu, G., Marcus, S., Păun, Gh.: New strategies of using the rules of a P system in a maximal way: Power and complexity. Romanian Journal of Information Science and Technology 12(2), 157–173 (2009)
5. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Nuñez, A., Romero-Campero, F.J.: Computational efficiency of dissolution rules in membrane systems. International Journal of Computer Mathematics 83(7), 593–611 (2006)
6. 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)
7. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
8. Murphy, N., Woods, D.: Uniformity is weaker than semi-uniformity for some membrane systems. Foundamenta Informaticae 134(1–2), 129–152 (2014)
9. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
10. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
11. 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)
12. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with elementary active membranes: Beyond NP and coNP. In: Gheorghe, M., Hinze, T., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 11th International Conference, CMC 2010, Lecture Notes in Computer Science, vol. 6501, pp. 338–347. Springer (2011)
13. Păun, Gh., Pérez-Jiménez, M.J., Riscos Núñez, A.: Tissue P systems with cell division. International Journal of Computers, Communications & Control 3(3), 295–303 (2008)
14. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
15. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, pp. 289–301. Springer (2001)
16. Zandron, C., Leporati, A., Ferretti, C., Mauri, G., Pérez-Jiménez, M.J.: On the computational efficiency of polarizationless recognizer P systems with strong division and dissolution. Fundamenta Informaticae 87, 79–91 (2008)

Building a basic membrane computer

Alejandro Millan, Julian Viejo, Juan Quiros,
Manuel J. Bellido, David Guerrero, and Enrique Ostua

Grupo ID2 – Universidad de Sevilla (Spain)
Email: amillan@us.es, julian@dte.us.es, jquiros@dte.us.es,
bellido@dte.us.es, guerre@dte.us.es, ostua@dte.us.es
www.dte.us.es/id2

Summary. In this work, we present the building of two well-known membrane computers (squares generator and divisor test). Although they are very basic machines they present problems common to every P system (competition, parallel execution of rules, membrane dissolution, etc.) that have to be solved in order to get real emulations for them. The presented designs mimic the systems operation in a realistic way, by achieving both maximum parallelism and non-determinism, and demonstrating for the first time that a membrane computer can actually be built *in silico*. Our architectures fully emulate the membranes behaviour yielding to a performance of one transition per clock cycle, supposing a real physical realization of the mentioned machines.

Key words: membrane computing, P system, digital circuit design, parallel computing, reconfigurable hardware, FPGA.

1 Introduction

Membrane computing was introduced in 2000 by Gheorghe Păun [12]. This topic is based on living cells and the first models are defined as an hierarchical structure of compartments (membranes), which contains objects (chemicals), which evolve according to applicability rules (chemical reactions). However, membrane computing has a very important problem: implementation. This kind of computing is extremely powerful but is a machine-oriented solution so current efforts in this way have been focused on improve the simulation of such system on both software and hardware platforms.

Several works exist on membrane computing from the hardware point of view. All of them have been focused on FPGA designing, trying to mimic the internal structure of P systems, in one way or another, into an electronic device.

Firstly, in [13], authors present a development in which the membranes of the system evolve in a parallel way although, internally, rules in each membrane are executed sequentially. Also, the general functioning of the system is deterministic: rules are applied by following a pre-established order.

Secondly, in [1], after some works on this topic [5, 6], authors achieve an architecture based on grouping rules into macro-rules and generating a new power set of macro-rules (each macro-rule is conformed by one or more of the original rules). In this way, they calculate this power set by including all the possible combinations of the original rules and, following a non-deterministic process, the architecture chooses one macro-rule and apply it in a maximal way. This process continues sequentially until none macro-rule can be applied. So, they achieve a certain degree of parallelism between the execution of the original rules.

Thirdly, in [10, 11], after a considerable amount of works [7, 8, 9], authors finally present two architectures that simulate P systems: the first one is focused on rules and the second one is focused on membranes. This second architecture mimics very well the structure of P systems but it suffers from important limitations: it allows systems in which competing rules (rules that consume the same objects) are prioritized what yields to deterministic systems only. Also, object selection is done sequentially so parallelism is set aside in this way.

Fourthly, in [15, 14], a very important advance on the topic is presented. These works cover the emulation of P systems on reconfigurable hardware, observing both parallelism (in terms of competing rules and object selection) and non-determinism. Authors present a development capable of automatically generate the hardware equivalent to a given P system that can compute each of its transitions in constant time (5 clock cycles). The problem with this approach is that the architecture supports min_1 transition mode only (rules can be applied one time maximum [3]).

In conclusion, although several authors have performed relevant efforts on the topic of electronic realization of P systems, until now, nobody has achieved an structure that mimics the fully parallel and non-deterministic behaviour of such machines.

Exactly in this aspect, our current work tries to contribute something relevant. We have adopted a very different point of view: membrane computing is a machine-oriented computational model so we think it is impossible to design a machine (or architecture) that solves all the problems. In the same way that we need to develop new algorithms to solve new problems in an algorithm-oriented computational model, we think it is necessary to design a new specific machine for each P system when needed. At least, while being important to observe the inherent features of this kind of systems: i.e. non-determinism and maximal parallelism. So, following this idea, we have started our work by trying to build the very first membrane computers we all know (Fig. 1 and Fig. 2: the ones presented in the foundational paper [12]).

The rest of the paper is organized as follows: in Sect. 2, we explain in detail the developed architectures and the designs employed to exactly emulate maximum parallelism and non-determinism of the chosen P systems, in Sect. 3, we present the operation results obtained by the built emulators, and finally in Sec. 4, we finish with the main conclusion of this work.

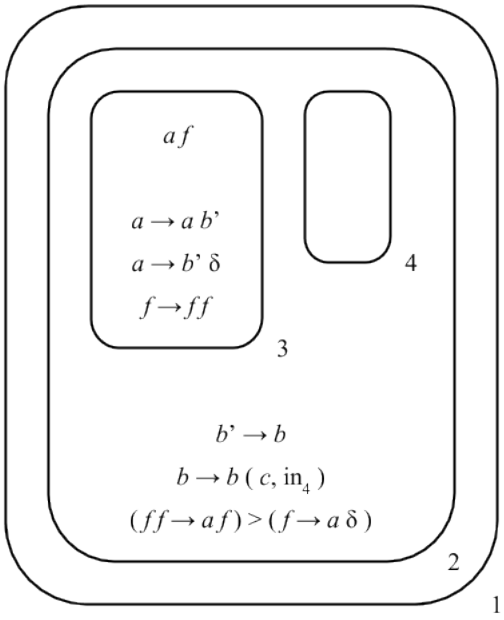


Fig. 1. Computer 1: n^2 generator ($n >= 1$).

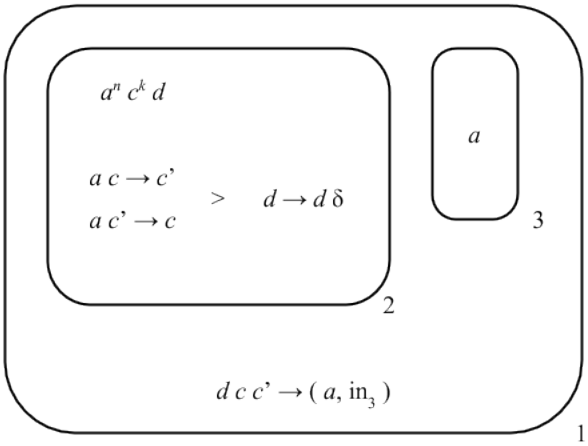


Fig. 2. Computer 2: Divisor test (Does k divide n?).

2 Design

With the exposed idea, we have developed architectures for the both mentioned P systems. However, there are common aspects that are interesting to be exposed firstly.

The main problem in the implementation of P systems is the application of rules, moreover when they are competing for common objects. So, the basic structure we have employed is shown in Fig. 3. It does not represent a rule corresponding to any of the chosen systems but illustrates how we have oriented the design of rules. In this figure, a very simple rule is shown: the REG block stores the amount of available f objects while the Logic block (associated to rule r_1) calculates how many times the rule has to be applied in order to consume all f objects. Finally, there is a feedback operation in which the object amount is adjusted according to rule applications. This construct allows the execution of the rule multiple times in a single clock cycle. From this basic case, we are going to show how we have addressed with competition in the chosen P systems.

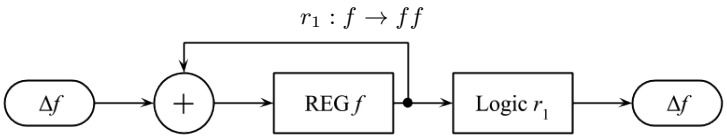


Fig. 3. Basic structure emulating object and rule.

Competition in Computer 1 (Fig. 1) comes from the two first rules in membrane 3. They consume a same object a. In this case, object a maintains its amount until the second rule is triggered (then membrane 3 is dissolved). So, in this system, we have included a random number generator (based on an maximum-length Galois linear-feedback shift register, LFSR [2, 4]) that ensures all possible executions are performed in a non-deterministic way. The LFSR determines the cycle in which the second rule is applied. Note that, being a maximum-length LFSR, it allows the system to go through all the possible executions without repeating anyone. The hardware structure corresponding to these rules is shown in Fig. 4.

Competition in Computer 2 (Fig. 2) comes from the two first rules in membrane 2. They consume a same object a and an specific object (c or c' in each case). In this membrane, it is necessary to distribute object a randomly between the two rules, also taking into account the amount of objects c and c' available in the system. The hardware employed to mimic this behaviour is presented in Fig. 5 and it is based on the following algorithm:

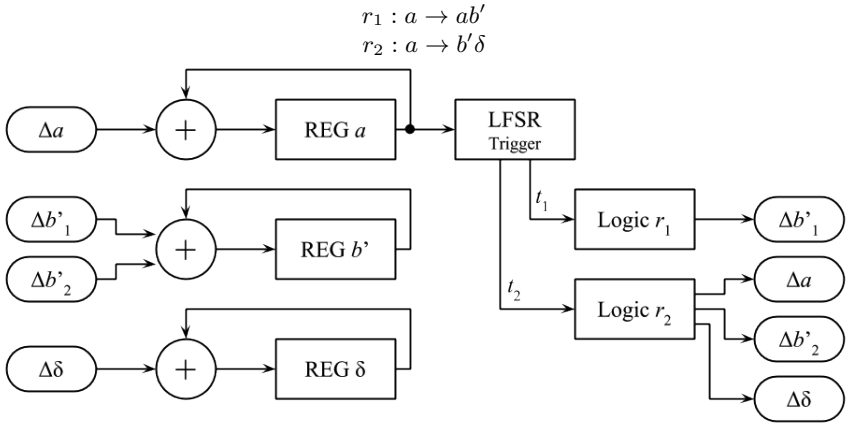


Fig. 4. Structure resolving competition case in Computer 1.

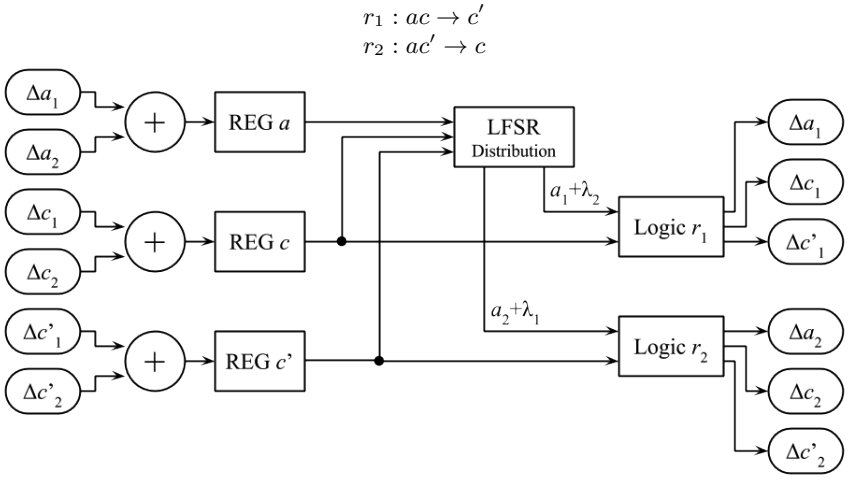


Fig. 5. Structure resolving competition case in Computer 2.

1. Randomly let:
   $a = a_1 + a_2$
2. Let:
   $\lambda_1 = max\{0; a_1 - c\}$, $\lambda_2 = max\{0; a_2 - c'\}$
   $\tau_1 = min\{c; a_1 + \lambda_2\}$, $\tau_2 = min\{c'; a_2 + \lambda_1\}$
3. Apply $r_1 \times \tau_1$ times and $r_2 \times \tau_2$ times.

It is important to denote that the algorithm describes only the idea under the design but the hardware structure calculates τ_1 and τ_2 in a combinational way (i.e. in a single clock cycle).

The rest of rules on both machines have been designed following the structure described in Fig. 3. They do not imply competition so they can be constructed in a direct way.

Finally, the machines interfaces are shown in Fig. 6 and 7. In the case of Computer 1, the Reset signal launch a new system computation (performing one transition per Clock cycle). The Master Reset and Seed inputs are employed at first only in order to initialize the LFSR by the user (the seed is obtained from the time past since the circuit was powered on; by using a 50 MHz counter). Then, the machine works continuously generating, in a non-deterministic way, all the squares existing in its computing range. Each time a square is produced, its value is presented through the Answer output bus and the Data Valid signal is activated (yielding to a new Reset and a new execution).

In the case of Computer 2, the Clock, Reset, Master Reset, and Seed signals work in a similar way to Computer 1. Also, Number n and Number k pass to the system the test inputs. Once the numbers have been tested, Data Valid is activated and the Answer signal indicates True or False as a response.

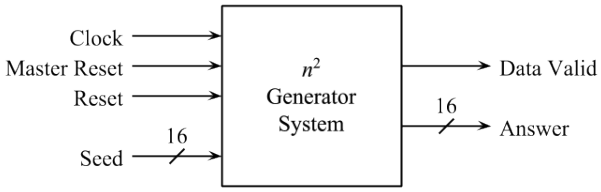


Fig. 6. Computer 1 interface.

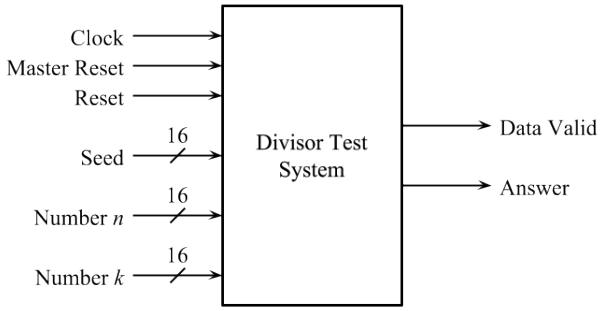


Fig. 7. Computer 2 interface.

3 Results

Both machines have been tested in both software and hardware ways. On the one hand, the software testing has been performed with Xilinx ISE v11.4. In Fig. 8 we present an operation detail of Computer 2 having $k = 21$ and $n = 63$. The machine starts having 63 a objects and 21 c objects inside membrane 2. In the first transition, rule r_1 is applied 21 times, giving us a new configuration without c objects and 42 a objects and 21 c' objects. In the second transition, rule r_2 is applied 21 times, giving us a new configuration without c' objects and 21 a objects and 21 c objects. In the third transition, rule r_1 is applied 21 times (again), giving us a new configuration without a and c objects and 21 c' objects. In the fourth transition neither rules r_1 nor r_2 can be applied, so membrane 2 is dissolved (rule r_3), giving us a new configuration with 21 c' objects and 1 d object inside membrane 3. Finally, rule r_4 can not be applied because the lack of c objects inside the membrane, so system is halted. At this moment, DV (data valid) signal activates and the Answer output indicates True (k divides n). As we have mentioned previously, each transition is processed in a single clock cycle.

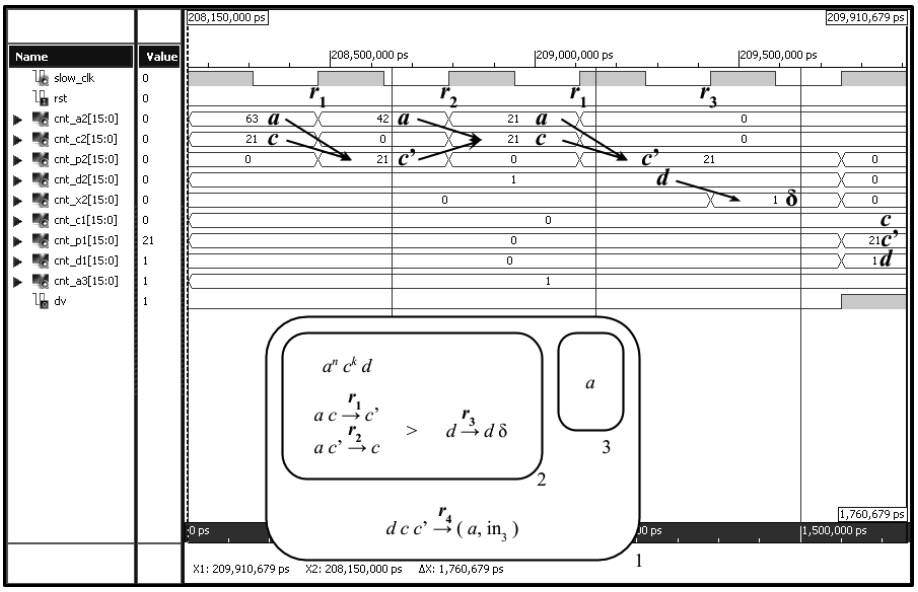


Fig. 8. Simulation detail of Computer 2 testing if $k = 21$ divides $n = 63$.

On the other hand, the machines have been tested on hardware by programming them into both Digilent Basys 2 training boards equipped with Spartan 3E-1200 FPGAs.

Operation of Computer 1 board (Fig. 9) is started by pressing the Master Reset button what feeds a random seed to the LFSR (also lightning on the Master

Reset indicator). Then, the machine starts computing executions of the P system, yielding the calculated answers to the 4-digit 7-segment display. These executions are carried out in a non-deterministic order but they cover all the possible combinations and generate all squares existing between 1^2 and 63^2. Once all squares are shown, the machine halts and the Finish indicator is lighted on. For demonstration purposes only, the board clock frequency has been reduced to a human-visible one (ca. 25 Hz) and can be observed through the Clock indicator.

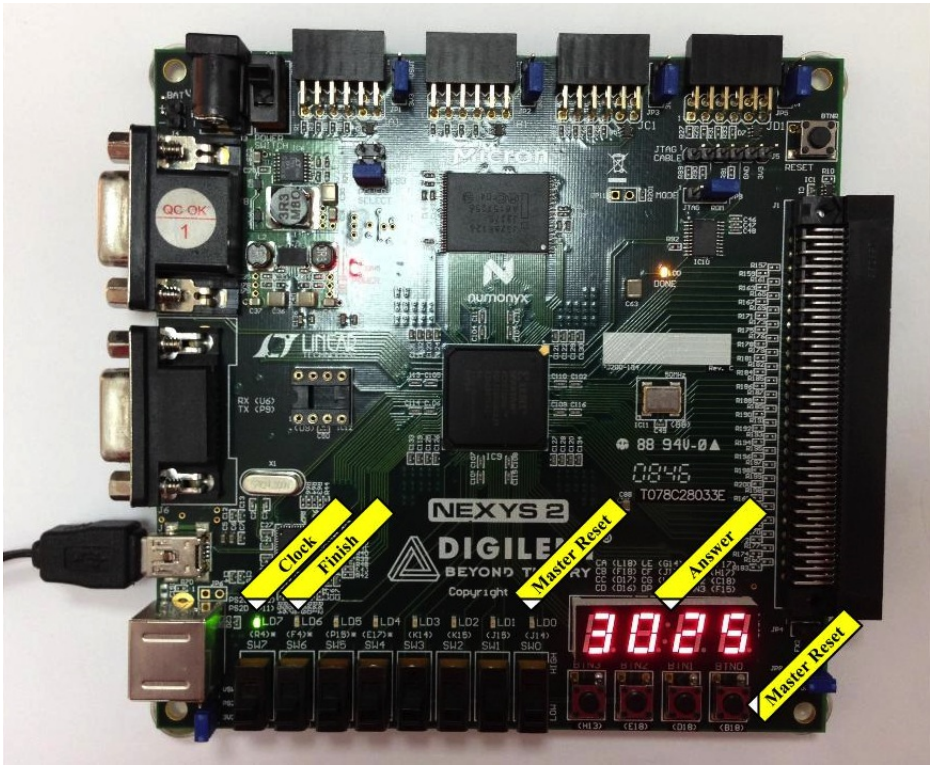


Fig. 9. Computer 1 training board emulator: Clock indicator (LD7), Finish indicator (LD6), Master Reset indicator (LD0), Answer (7-segment display), and Master Reset button (BTN0).

In a similar way, operation of Computer 2 board (Fig. 10) is also started by pressing the Master Reset button what feeds a random seed to the LFSRs present in the circuit. Then, the machine starts computing executions of the P system (each time a execution finishes the Data Valid indicator is lighted on). The executions are fed with random inputs and the machine shows them continuously on the 7-segment display (Numbers n and k). In order to facilitate humans to understand results, the board pauses execution when the division test is successful during

enough time to read the display. Also for demonstration purposes only, the board clock frequency has been reduced to a human-visible one (ca. 25 Hz) and can be observed through the Clock indicator.

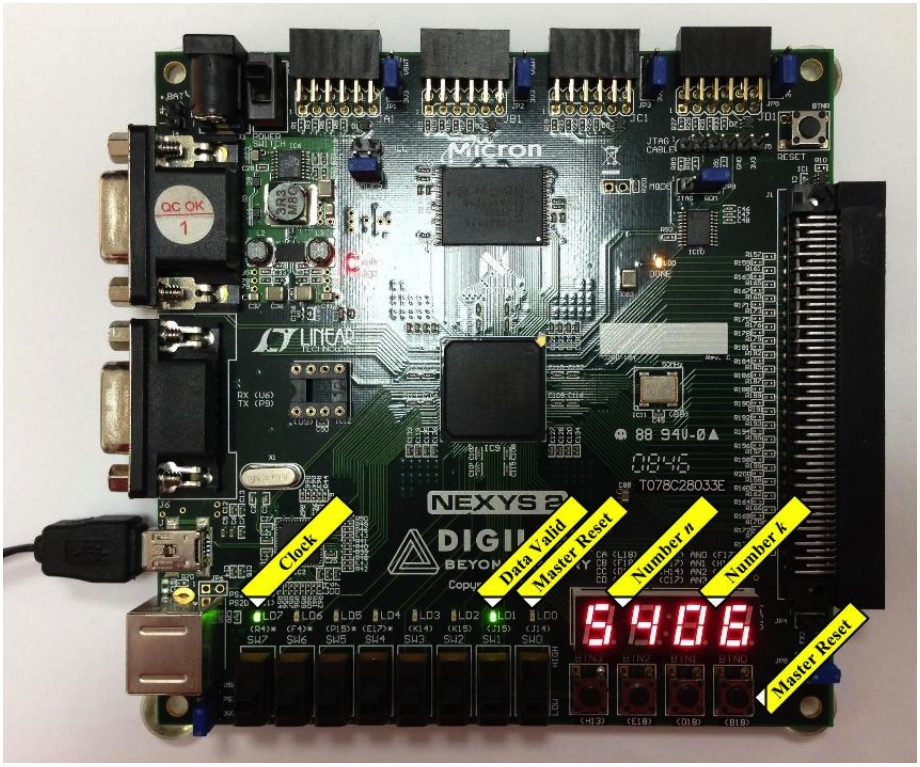


Fig. 10. Computer 2 training board emulator: Clock indicator (LD07), Data Valid indicator (LD1), Master Reset indicator (LD0), Number n (first two digits of 7-segment display), Number k (last two digits of 7-segment display), and Master Reset button (BTN0).

The hardware details are presented in Table 1 and the device utilization is shown for both cases in Table 2 and Table 3 respectively. As we can observe, both computers consume a small portion of the available resources showing themselves as a very efficient design also in terms of hardware needs. Also, it is noticeable that register width has an important impact on resource utilization: 64-bit in the case of Computer 1 what yields to a maximum amount of ca. 10^{19} objects of each type (register values are of signed type). These register widths has been chosen according to the 4-digit displays available on the training boards.

	Computer 1	Computer 2
Target Device	xc3s1200e-4fg320	
Clock Frequency	50 MHz	
Performance	50 Mtransition/s	
Register Width	64-bit	16-bit

Table 1. Hardware details.

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	625	17344	3%
Number of 4 input LUTs	2434	17344	14%
Number of occupied Slices	1345	8672	15%
- Number of Slices containing only related logic	1345	1345	100%
- Number of Slices containing unrelated logic	0	1345	0%
Total Number of 4 input LUTs	2479	17344	14%
- Number used as logic	2434		
- Number used as route-thru	45		
Number of bonded IOBs	18	250	7%
- IOB Flip Flops	2		
Number of BUFGMUXs	2	24	8%
Number of MULT18X18SIOs	10	28	35%
Average Fanout of Non-Clock Nets	2.27		

Table 2. Device utilization for Computer 1.

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	269	17344	1%
Number of 4 input LUTs	842	17344	4%
Number of occupied Slices	503	8672	5%
- Number of Slices containing only related logic	503	503	100%
- Number of Slices containing unrelated logic	0	503	0%
Total Number of 4 input LUTs	958	17344	5%
- Number used as logic	842		
- Number used as route-thru	116		
Number of bonded IOBs	19	250	7%
- IOB Flip Flops	1		
Number of BUFGMUXs	2	24	8%
Number of MULT18X18SIOs	4	28	14%
Average Fanout of Non-Clock Nets	3.25		

Table 3. Device utilization for Computer 2.

4 Conclusion

In this work, we have shown that it is possible to fully emulate a P system without loosing its intrinsic features of maximal parallelism and non-determinism. With that aim, machines presented in the foundational paper of the discipline (square generator and divisor test) have been built. Our designs mimic the internal structure of the P systems allowing the resulting hardware to perform as the theoretical system should: processing one transition per clock cycle. The systems evolve in a non-deterministic way and rules are applied in a maximal parallel derivation mode; what, to the best of our knowledge, supposes the first real emulation of a P system *in-silico*.

References

1. Alonso, S., Fernandez, L., Arroyo, F., Gil, J.: A Circuit Implementing Massive Parallelism in Transition P Systems. International Journal "Information Technologies and Knowledge", vol. 2, pp. 35–42 (2008).
2. Bonde, V., Kale, A.: Design and Implementation of a Random Number Generator on FPGA. International Journal of Science and Research, vol. 4, no. 5, pp. 203–208 (2015).
3. Freund, R., Ibarra, O., Paun, A., Sosik, P., Yen H.: Catalytic P Systems. In "The Oxford Handbook of Membrane Computing", pp. 83–117, Oxford University Press (2009). ISBN:9780199556670
4. George, M., Alfke, P.: Linear Feedback Shift Registers in Virtex Devices. Xilinx Application Note, XAPP210 v1.3 (2007).
5. Martinez, V., Fernandez, L., Arroyo, F., Garcia, I.: A HW circuit for the application of Active Rules in a Transition P System Region. Proc. 4th International Conference Information Research and Applications, Varna (Bulgary), pp. 80–87 (2006). ISBN-10: 954-16-0036-0.
6. Martinez, V., Fernandez, L., Arroyo, F., Guiterrez, A.: HW Implementation of a Bounded Algorithm for Application of Rules in a Transition P-System. Proc. 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara (Romania), pp. 32-38 (2006).
7. Nguyen, V., Kearney, D., Gioiosa, G.: Balancing performance, flexibility and scalability in a parallel computing platform for membrane computing applications. Lecture Notes in Computer Science, vol. 4860, pp. 385–413 (2007). DOI:10.1007/978-3-540-77312-2_24
8. Nguyen, V., Kearney, D., Gioiosa, G.: An implementation of membrane computing using reconfigurable hardware. Computing and Informatics, vol. 27, no. 3+, pp. 551569 (2008).
9. Nguyen, V., Kearney, D., Gioiosa, G.: An Algorithm for Non-deterministic Object Distribution in P Systems and Its Implementation in Hardware. Lecture Notes in Computer Science, vol. 5391, pp. 325–354 (2009). DOI:10.1007/978-3-540-95885-7_24
10. Nguyen, V., Kearney, D., Gioiosa, G.: An extensible, maintainable and elegant approach to hardware source code generation in Reconfig-P. The Journal of Logic and Algebraic Programming, vol. 79, no. 6, pp. 383-396 (2010). DOI:10.1016/j.jlap.2010.03.013

11. Nguyen, V., Kearney, D., Gioiosa, G.: A Region-Oriented Hardware Implementation for Membrane Computing Applications. Lecture Notes in Computer Science, vol. 5957, pp. 385–409 (2010). DOI:10.1007/978-3-642-11467-0_27
12. Paun, G.: Computing with Membranes. Journal of Computer and System Sciences, vol. 61, pp. 108–143 (2000). DOI:10.1006jcss.1999.1693
13. Petreska, B., Teuscher, C.: A Reconfigurable Hardware Membrane System. Lecture Notes in Computer Science, vol. 2933, pp. 269–285 (2004). DOI:10.1007/978-3-540-24619-0_20
14. Quiros, J., Millan, A., Viejo, J.: *Implementacion sobre hardware reconfigurable de una arquitectura no determinista, paralela y distribuida de alto rendimiento, basada en modelos de computacion con membranas* (Implementation on reconfigurable hardware of a non-deterministic, parallel, and distributed high performance architecture based on membrane computing models). PhD Thesis (2016).
15. Verlan, S., Quiros, J.: Fast Hardware Implementations of P Systems. Lecture Notes in Computer Science, vol. 7762, pp. 404–423 (2013). DOI:10.1007/978-3-642-36751-9_27

14^{th} Brainstorming Week on *Membrane Computing*

Laura Moreno Valero

Universitat de Barcelona
Email: 95morenolaura@gmail.com

Three weeks ago we, a group of students from the University of Barcelona, were given the possibility of going to Seville, in order to attend a Workshop on *Membrane Computing*. The seven of us that went to the Workshop are all physics students and until the day we received the mail making us aware of that event, I had never heard about *Membrane Computing*, so I was kind of surprised the first time I read about it. In the university we have learned how to program with Python and Fortran. It was mostly on how to deal with physics problems, and how to find a solution using numerical methods. But what we saw when we arrived there was completely different, it was like entering another world of computation. While dealing with physics we are always reminded of how important is to keep in mind the physical part of the problem (i.e. how a given system is supposed to react when a field is applied, or if the result makes sense with what we can see in the reality), when programming with that new computational way you try to think how to model things but once you get it, you abstract (somehow) from reality and work with maths, or that is what I understood the first days. It was really difficult, specially at the beginning, as I was completely lost and maybe it could have been a better idea to have been previously informed about what the talks were about, or even to have an idea of what *Membrane Computing* meant.

During the days I spent there I got to know that this picturesque way of computing is based on a system analogous to how alive cells behave or how elements in these ones evolve, using P-Systems (a computational model based on the structure of living cells that permits to perform calculations). Indeed, the inspiration came from how processes take place in cells. This new computing paradigm makes use of the structure of the cells, working with their different membranes and even with the environment. But it is not only something involving one cell, they also work with multiple ones, and so there are different models on which they can work. The most usual one is the cell-like system, which is the one I have already talked about. There are also other kinds of models, such as tissue-like or neural-like systems; the first ones are typically used to represent interaction between cells, or better said to

implement exchanges of elements between cells. The second one is used to analyse the behaviour of neurons, and tries to represent synapsis' process.

To work with all that stuff, a new language is needed, and so P-lingua takes action. P-lingua is a programming language to define P-Systems and it is highly used, as it is a language close to scientific notation, as well as from my point of view really visual of what it is going on. While we were attending the talks, we were invited to participate by doing a small project where we could use what we were learning, so our first idea was how could we use it in our own field, physics. We decided to reproduce two physical situations in order to see the process: the final result as well as their intermediate stages. These two projects were about: i) the Stern-Gerlach experiment, by modifying it a bit and ii) the Uranium-238 Decay Chain. At the beginning of our project, we tried to program a small code to get a bit used to the language, and I was truly surprised when I saw that it was not that weird to write it, and that by following one example one could understand almost perfectly what we were expressing. Nevertheless, I am aware that I only got a small view of how it works and that, for sure, it turns more difficult when you keep on programing more advanced things.

However, even though it was not easy to understand most of the parts of the talks, people was extremely nice with us. One of my biggest fears was to break the dynamic of the event, as we could probably slow down the pace of it with our low level, but it surprised me a lot when everyone was inviting us to make as many questions as we wanted. Of course I did not want to disturb the pace of the event, but sometimes it was almost impossible to follow explanations, although I consider I learned a lot during that week, so to say, we got a slight insight on the general topic, but nothing really deep.

Most of the conferences were interesting as it was another way of thinking, one which we are not so used to. Nevertheless, there was one that made me think of other fields and possible further applications. It was about the concept of Eco-P colonies. It is based on systems of only one membrane and in each of them we can find the objects that we want to study or see the final product of evolution. In Eco-P colonies, these objects do interact with a shared environment and this was the main point of the talk, titled *P Colonies with Dynamic Environment*. In order to evolve, the objects inside the membrane interact with the environment by some rules, specially rewriting and communication ones. The first ones basically transform the object into another one, or even into multiple objects. They are also called evolution rules and are basically applied inside the same membrane/entity (or even called agent), whereas the second kind of rules are based in an exchange between objects inside the membrane and objects outside it (environmental objects, usually symbolized by e). To sum up, Eco-P colonies would be a way to show mechanisms of generation and consumption from the environment. The system halts when no more rules can be applied to the objects, or no more exchange

between the agents and the environment can be done.

In fact, before the talk began, Petr Sosík, who gave the conference, showed us a video where we could see some bacterias multiplying, and we could see how the growth was really fast. This reminded me of a conference we were given some months ago, about cancer cells and which are their mechanisms to propagate cancer. I was wondering, when I heard about Eco-P colonies and their rules, if they could have some applications to Medicine, by adapting some of their rules to how cancer cells behave, or even in other cases. In fact, *Membrane Computing* modelizes non-deterministic processes, so somehow we could be able to implement the probabilities of a possible mutation (that produces the beginning of the disease).

Another thing that surprised me was the characteristic of maximal parallelism when working with P-systems. At what we are used to, we have an order, so the first written commands are the first to be displayed, whereas in maximal parallelism, more than one rule can be executed at the same time. Indeed, it is based in the fact that the maximum number of rules that can be applied in each step, are applied, maximizing the number of processes that can be done at one time.

To sum up, this experience has been really advantageous. First of all because I still do not know which is the field that interests me the most, but I have seen that computation keeps attracting me, and now even more than ever. Secondly, going there has made me realize of how vast this area can be and that it does not only reduce to solve some problems by numerical approximations but they are also a useful tool to visualize some difficult experiments or processes. And to end up with, it was a kind of personal growth, where we got to work together as a team and got the chance to talk with great people that taught us about science in general and more about research.

Open Problems, Research Topics, Recent Results on Numerical and Spiking Neural P Systems (The "Curtea de Argeş 2015 Series")

Gheorghe Păun[1], Tingfang Wu[2], Zhiqiang Zhang[2]

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
gpaun@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
whutwutf@163.com, zhiqiangzhang@hust.edu.cn

Summary. A series of open problems and research topics are formulated, about numerical and spiking neural P systems, initially prepared as a working material for a three months research stage of the second and the third co-author in Curtea de Argeş, Romania, in the fall of 2015. Further problems were added during this period, while certain problems were addressed in this time; some details and references are provided for such cases.

1 Introduction

In membrane computing there are numerous open problems and research topics in circulation, many of them also collected in systematic lists, compiled, for instance, for the yearly Brainstorming Week on Membrane Computing, see http://ppage.psystems.eu and www.gcn.us.es. Because the present list had initially a working material character, we do not provide here complete references. Furthermore, as the reader is supposed to be a researcher in membrane computing, we do not give basic definitions either (but we give details for the new classes of P systems considered). As a general reference we refer to the Handbook [8].

The list deals only with numerical P systems (in short, NP systems, [7]) and spiking neural P systems (in short, SNP systems, [3]). Some problems are more specific, many others are just general ideas, so that the first step in approaching them implies a formal definition (possibly, of new classes of NP or SNP systems). We recall some details from several papers written in Curtea de Argeş during the mentioned period of time.

2 Bridging NP and SNP

The two "exotic" classes of P systems (their "biochemistry" is not very closely related to the biological one) have many common and, also, many different characteristics. Of the first type is the fact that both of them process numbers and a specific "production" of a cell is distributed among neighboring cells. Thus, it is just natural to check whether features of one class can be extended to the other class, and conversely. This proves to be a very fruitful idea.

Here are a few more precise suggestions of this kind.

A. NP with SNP features. Three main ingredients of SNP systems can be exported also to NP systems: the tissue-like arrangement of membranes, the regular expressions which control the application of spiking rules, and the replication of the production to all the adjacent cells. However, further features can be considered – only a few suggested below, so this is already a general research issue.

B. Passing from the cell-like membrane structure of usual NP systems to tissue-like NP systems is just an extension which will also extend all computing power results and computational complexity properties. As the distribution of the *production* of a compartment is done with a precise identification of the target variables, a simple (real-time?) mutual simulation between cell-like and tissue-like NP systems is expected.

C. What about using, in the tissue-like NP systems, weights on "synapses" instead of repartition coefficients? The simulation of NP systems with weights by systems with repartition coefficients and conversely is a natural research topic. What about using both repartition coefficients and weights on "synapses"? (This might also have economic interpretations, thus bringing the model closer to the initial motivation, the economics.)

D. Associating a regular expression with each evolution program seems to be a non-trivial extension of the enzymatic control from [6]. (Note that we can compare the values of two variables, even in terms of regular expressions, in the following way: $(x_1x_2)^*x_2^*$ can be interpreted as $x_2(t) \geq x_1(t)$, meaning that the value of x_2 at time t is greater than or equal to the value of x_1 at time t.) In what cases can this intuition be confirmed? A good candidate is the descriptional complexity of various NP systems, for instance, constructed for controlling robots.

E. While considering a regular expression looks like adding computing power, distributing the production of a cell to all neighboring variables, replicated, on the one hand, increases the total values of variables in the system, on the other hand, removes the control possibilities provided by the repartition coefficients. Which is the power (and the efficiency) of NP systems with such a repartition protocol remains to be checked.

F. Directly related to the previous idea is the following problem: what about NP systems with an "egalitarian" repartition, i.e., with all distribution coefficients equal in each evolution program/in each compartment/in the whole system? A

particular case is that of considering all distribution coefficients equal to 1. Are such restricted NP systems still universal?

G. Continuing with the restrictions, what about considering NP systems with only $k \geq 1$ variables in the distribution protocols? Is any difference in power between NP systems with k variables and those with $k+1$ variables, for various values of k? (It is expected that $k = 1$ is, indeed, a special case.) What about NP systems with both egalitarian distribution and at most k variables in each distribution protocol?

H. SNP with NP features. This is the "reverse" of problem **A**, again with (at least) three basic directions: considering SNP systems with a cell-like membrane structure, using a production function instead of spiking rules, and considering a distribution protocol for communicating the produced spikes. What else, it remains to imagine.

I. SNP systems with a cell-like membrane structure look "non-natural" from a biological point of view, but it is mathematically interesting, especially in view of the children-parent membrane interaction; remember that circularity is not allowed in standard SNP systems.

This idea was explored in [14]; we recall some details.

A *cell-like SN P system* (in short, a cSN P system), of degree $m \geq 1$, is a construct

$$\Pi = (O, \mu, n_1, \ldots, n_m, R_1, \ldots, R_m, i_o),$$

where $O = \{a\}$, μ is a hierarchical membrane structure with m membranes, $n_i, 1 \leq i \leq m$, is the number of spikes present in compartment i of μ at the beginning of the computation, $R_i, 1 \leq i \leq m$, is the finite set of rules from compartment i, and i_o indicates the output region (this is the environment if $i_o = env$).

Besides forgetting rules of the form $a^s \to \lambda,\ s \geq 1$, the sets R_i contain *spiking rules* of the (extended) form $E/a^c \to u$, where E is a regular expression over O, $c \geq 1$, and u is a sequence of couples of the form (a^p, tar), where $p \geq 1$ and tar is a target indication specifying the destination of the p associated spikes. This target can be *here, out, in*, in_j, where j is the label of a membrane, with the usual meaning in cell-like P systems, or in_{all}, with the meaning that the p spikes will be sent, replicated, to all immediately inner membranes (each of them will receive p spikes). Of course, in the case of non-extended rules, when only one spike is produced by a rule, only one couple of the form (a, tar) will be used.

The computations in a cSN P system are defined as in usual SN P systems: (at most) one rule in each compartment is applied, but the compartments work in parallel, synchronously. The result can be obtained as the number of the spikes in region i_o in the moment when the computation halts, and this can be inside the system or outside, when $i_o = env$. We denote by $N_{in}(\Pi)$ the set of numbers computed (generated) by the system Π in the internal mode. We will not consider here also the external output in the form of the number of spikes sent out, as this is a direct dual of the inner mode, but, like in SN P systems, we also consider as the result of a computation the distance in time between the first two steps when

the system sends spikes out; this can be done by rules introducing couples (a^p, out) used in the skin region of Π, hence in this case the indication of i_o is omitted. We denote by $N_2(\Pi)$ the set of numbers computed by Π in this sense, by means of halting or non-halting computations. (By convention, number 0 is computed by a computation which sends out spikes only once.)

It is important to note that in the previous definition we have imposed no restriction on the number of produced spikes, that is, it can be greater than the number of consumed spikes. Actually, we need rules for producing more spikes than consumed, otherwise we cannot increase the number of spikes in the system – unless if we use the replication target command in_{all}.

We denote by $N_\alpha cSNP_m(forg, here, in_t, in_{all})$, $\alpha \in \{2, in\}$, the family of sets of numbers $N_\alpha(\Pi)$ computed by cSN P systems Π with at most m membranes, using forgetting rules and target indications of the types $here, in_t, in_{all}$, together with indications *in, out*. We explicitly write only *forg* and the indications $here, in_t, in_{all}$ because these features are powerful and they can be avoided in certain cases (this also happens in standard SN P systems with *forg*). When all spiking rules $E/a^c \to u$ have c greater than or equal to the number of spikes in u we write $N_\alpha cSN'P_m(...)$ instead of $N_\alpha cSNP_m(...)$. When the number of membranes is not bounded, we replace the subscript m with $*$.

Here are the results reported in [14]:

1. $N_{in}cSNP_4(here, in_t) = NRE$.
2. $N_{in}cSNP_7(in_t) = NRE$.
3. $N_{in}cSNP_7 = NRE$.
4. $N_{in}cSN'P_*(in_t, in_{all}) = NRE$.
5. $N_2cSNP_4(here, in_t) = NRE$.
6. $N_2cSN'P_*(here, in_t, in_{all}) = NRE$.

Several open problems and research topics were formulated in [14].

First, a large research area is open just by checking whether the results obtained for usual SN P systems can be extended to cSN P systems. Many questions are of interest: looking for small (as the number of membranes) universal cSN P systems, adding anti-spikes, working in a parallel way also in the membranes, working asynchronously, and so on and so forth. In [14] one starts directly with extended systems (without delay). Which is the power of non-extended cSN P systems? In this case we need a way to replicate spikes. In [14] in_{all} it is used to that aim; what else can be imagined? Look for restrictions which lead to characterizations of sub-universal families of numbers (such as $NREG$) or of languages (in the case of the external output; note that the spike train can be also a sequence of symbols over an arbitrary alphabet).

The languages generated by cell-like SN P systems were investigated in [13] – many results were obtained, but also many questions remain to be further examined. We do not enter into details.

J. Replacing the spiking rules with a production function (of one variable if only spikes are considered, of two variables if also anti-spikes are used; the interplay

with the annihilation rule is also of interest – useful seems to be to apply the annihilation rule after computing the production, of both spikes and anti-spikes). The production function can be a polynomial, as in usual NP systems, but we can try to capture also other neural ingredients, such as the sigmoid function on which the functioning of the biological neuron is based.

K. Using a distribution protocol, for SNP systems with "standard" spiking rules looks easy: just associate distribution coefficients to synapses. This add "programming" possibilities, hence simpler proofs than for usual SNP systems are expected.

3 Further Problems for NP

The investigations on NP systems reported so far only deal with the basic systems and with the enzymatic ones, but there are many possibilities for considering further classes. Some ideas were mentioned also before, a few others will be suggested below, but the reader can imagine many more.

L. For instance, we can consider NP systems with restricted communication, in the sense that the production of a compartment is distributed only to variables from one or two levels out of the three used so far: *here, down, up*. For "one-way" systems it is expected to obtain rather restricted families of numbers generated in this way. Which cases still lead to universality?

M. A very natural idea is, instead of having the variables associated with compartments, to move variables across variables by associating with them the usual target indications *here, in, out*.

Numerical P systems *with migrating variables* (in short, MNP systems) were considered in [17] in the following form:

$$\Pi = (m, H, \mu, Var, (Pr_1, Var_1(0)), \ldots, (Pr_m, Var_m(0)), (x_{i_0}, j_0)),$$

where:

- $m \geq 1$ is the number of membranes;
- H is an alphabet (of labels for membranes in μ);
- μ is a hierarchical (cell-like) membrane structure with m membranes labeled with the elements of H;
- $Var = \{x_1, x_2, \ldots, x_n\}$ is a set of variables for the system;
- $Var_i \subset Var, 1 \leq i \leq m$, is a set of variables from Var, initially present in region i;
- $Var_i(0), 1 \leq i \leq m$, is a vector which indicates the values of the initial variables in region i;
- Pr_i, $1 \leq i \leq m$, is the finite set of programs in region i; each program has the following form:

$$F_{j,i}(x_{p_1},\ldots,x_{p_k}) \to c_{j,1}|(x_{r_1},tar_1)+\ldots+c_{j,q}|(x_{r_q},tar_q),$$

where $F_{j,i}(x_{p_1},\ldots,x_{p_k})$ is the production function, $c_{j,1}|(x_{r_1},tar_1)+\ldots+c_{j,q}|(x_{r_q},tar_q)$ is the repartition protocol of the program and $tar_1,\ldots,tar_q \in \{here, out, in\}$; the symbols *here, out, in* are called *target commands* or *target indications*; all the variables $x_{p_1},\ldots,x_{p_k}$ and $x_{r_1},\ldots,x_{r_q}$ are from Var.

- $x_{i_0} \in Var$, $j_0 \in H$.

The variables initially placed in membrane i have non-zero values specified by $Var_i(0), 1 \leq i \leq m$. A variable equal to zero is simply supposed not to be present in a membrane. This is called the NZP assumption. A program $F_{j,i}(x_{p_1},\ldots,x_{p_k}) \to c_{j,1}|(x_{r_1},tar_1)+\ldots+c_{j,q}|(x_{r_q},tar_q)$ can be applied only when all variables $x_{p_1},\ldots,x_{p_k}$ ("production variables") are present in membrane i at that time with non-zero values. By using the production function, the system computes a *production value* which is distributed to variables specified by the repartition protocol. An important observation is that variables involved in the production function are reset to zero after computing the production value.

The application of programs is as usual in numerical P systems, with the following specific points. After the application of the program, the variables involved in the repartition protocol are moved to the region indicated by the target command associated with them. Specifically, *here* means the variable will be placed in the same region i where the program is applied; *out* means the variable will be moved to the region immediately outside membrane i – this region can be the environment in the case when i is the skin membrane; *in* means the variable should be moved to a membrane immediately inside membrane i, non-deterministically chosen.

When a program is applied, for a variable involved in the program there are five cases to consider: i) if the variable appears in the production (it must be present in the membrane for the program to be applied) and not also in the repartition protocol, then this variable is zeroed and removed from the membrane; ii) if the variable appears both in the production function (with a non-zero value) and in the repartition, then it is first zeroed, then the variable with the contribution received from the repartition protocol is moved to the membrane indicated by the associated target; iii) if the variable appears in the repartition protocol and is not present in the membrane (hence it must not appear in the production function), then the variable with its contribution received from the repartition protocol is moved to the membrane as the associated target indicates; iv) if the variable appears in the repartition protocol and it was initially present in the membrane but not in the production, then the initial value plus the contribution it receives is moved to the membrane indicated by its associated target; v) if the variable appears in several repartition protocols, then, in order to avoid any conflicts/complications, we restrict to applying programs where the same variable has associated the same target indication in all programs; then, each program separately changes the variable as stated above and the variable, with the summed value, is moved to the associated target.

After moving variables to the target membranes, all the values of the same variable received from different membranes are added up, and the sum is the value of this variable in the destination membrane. If the sum is zero, then the variable is simply removed from the membrane. (Another possibility is to immediately move variables with the value received from each program to the associated targets and to sum the values at the destination.)

MNP systems can evolve in the *all-parallel* mode (at each step, in each membrane, all programs which can be applied are applied, allowing that several programs share the same variable), in the *sequential* mode (at each step, only one program is applied in each membrane; if more than one program in a membrane can be used, then one of them is non-deterministically chosen), or in the *one-parallel* mode (apply programs in the all-parallel mode with the restriction that one variable can appear in only one of the applied programs; in the case of multiple choices, the programs to apply are chosen in the non-deterministic way). In the one-parallel mode, where more than one program can be applied in a membrane, one can also impose the restriction that there is no conflict between the targets associated with variables in the repartition protocols of the applied programs.

Besides programs as above (called non-enzymatic), numerical P systems also have enzymatic programs of the form $F_{j,i}(x_{p_1},\ldots,x_{p_k})|_{e_{j,i}} \to c_{j,1}|(x_{r_1},tar_1)+\ldots+c_{j,q}|(x_{r_q},tar_q)$, where $e_{j,i}$ is a variable present in membrane i and different from $x_{p_1},\ldots,x_{p_k}$ and $x_{r_1},\ldots,x_{r_q}$. Such a program is applied at time t only if $e_{j,i}(t) > min(x_{p_1}(t),\ldots,x_{p_k}(t))$. Note that $e_{j,i}(t)$ remains unchanged in the program where it appears as an enzymatic variable; in other programs, $e_{j,i}$ can appear as a usual variable in production functions or repartition protocols, and it can be "consumed" or receive "contributions".

If every program is enzymatic, we call the system *purely enzymatic*.

Using the programs in the way mentioned above, we obtain *transitions* among configurations. A sequence of such transitions forms a *computation*. If no program can be applied in the current configuration, we say that the system *halts*. When the system halts, the value taken by the special variable x_{i_0} in membrane j_0 is the number generated by the computation.

The set of natural numbers generated by a system Π working in the *one-parallel* or *sequential* mode is denoted by $N_\alpha(\Pi), \alpha \in \{one, seq\}$, where *one* stands for one-parallel, *seq* stands for sequential. We use $N_\alpha M^0 \beta NP_m^D(poly^n(r), Var_{k_1}, Pro_{k_2})$, to denote the family of all sets $N_\alpha(\Pi)$, $\alpha \in \{one, seq\}$, $\beta \in \{E, pE, -\}$ of numbers generated by β numerical P systems Π with migrating variables (E = enzymatic, pE = purely enzymatic; if the system is non-enzymatic, then β is omitted), with at most m membranes, at most k_1 variables, and at most k_2 programs, with production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial; D indicates the use of deterministic systems (we remove it when the systems may also be non-deterministic); the superscript 0 means the system works under the NZP assumption. If this assumption is removed, hence the variables can be present also with value zero, and the programs can be applied if the variables are present in the membrane, does not

matter whether or not their values are zero (we say that we work without the NZP assumption), then the superscript 0 is removed. If one of the parameters m, n, r, k_1, k_2 is not bounded, then we replace it with $*$.

Here are part of the results proved in [17]:

1. $N_\alpha M^0 NP_1(poly^1(1), Var_2, Pro_2) - SLIN_1^+ \neq \emptyset, \alpha \in \{one, seq\}$.
2. $SLIN_1^+ \subset N_\alpha M^0 NP_1(poly^1(1), Var_*, Pro_*), \alpha \in \{one, seq\}$.
3. $N_{one} M^0 NP_1(poly^1(3), Var_*, Pro_*) = NRE$.
4. $N_{seq} M^0 NP_2(poly^1(3), Var_*, Pro_*) = NRE$.
5. $N_{one} MENP_1(poly^1(3), Var_*, Pro_*) = NRE$.
6. $N_{seq} MENP_2(poly^1(3), Var_*, Pro_*) = NRE$.

Also the possibility to generate strings with these systems was explored in [17].

N. Associate a language to a computation in an NP system. For instance, the values of a variable can form a string – in general, over an infinite alphabet (like in [2]), or on a finite alphabet. For instance, we can consider the binary string obtained by marking with 0 and 1 the odd and the even values of the distinguished variable. We can also "read" the natural numbers modulo a given constant $k \geq 2$, so that we can obtain strings over an alphabet with k letters.

O. In particular, we can associate a language to an NP system by considering an *external output*: we add a variable also to the environment, which gets parts of the production of the skin compartment. This can be used both for computing numbers and strings (in the latter case, following the suggestions from the previous question).

NP systems used as string generators were considered in [16]. A string is associated with a computation in a way somewhat similar to that adopted for spiking neural P systems: one just considers a special variable *out* in the environment which can appear in the repartition protocol of programs in the skin region of a numerical P system. At each step its value is first reset to zero, then it receives a new value. If at one step it receives several values from several programs, all these values are added up and the sum is the value it receives at this step. If the value is a number i between 1 to q, for some constant q, then the symbol b_i is added to the generated string. If at any step variable *out* receives a value which is greater than q or smaller than 0, then this computation aborts, no result is associated with it.

In order to define the generated string, we need to define its end. This is clear in the case when the computation halts (no further program can be applied), and this can be taken as a definition of successful computations in purely enzymatic P systems. In non-enzymatic and in (non-purely) enzymatic systems the computations never halt, and then we define the end of the string by means of a *signal*, e.g., the step when the system sends out value 0. Because for purely enzymatic systems we have halting at our disposal, in this case we avoid sending out value 0, that is, this case is simply ignored. (For a general definition, however, a decision should be made also for value 0 sent out. A possibility is to proceed as in spiking neural P systems, where in such a case a special symbol, b_0, is added to the string.)

It still remains a case not covered: the steps when the system sends no value to variable *out*. We have two choices: to forbid such steps, by the definition of correct computations, or to proceed as in the case of spiking neural P systems and to associate the string λ to the generated string (the string is not increased, the system can continue working).

In this way, we define two languages generated by a numerical P system Π. If at each step a positive value is sent to variable *out* (with the exception of the last step, for non-enzymatic and for enzymatic P systems, when value 0 is sent out, marking the end), then we denote the generated language by $L^{res}(\Pi)$ (with *res* coming from *restricted*). If in the steps when no value is sent out (neither 0) we interpret that the system adds λ to the generated string, then the generated language is denoted by $L^{\lambda}(\Pi)$.

For an easier remembering, we synthesize the previous conventions/definitions in a table:

Sending out	Non-enzymatic & Enzymatic	Purely enzymatic
$1, 2, \ldots, q$	b_i	b_i
< 0 or $> q$	abort	abort
0	end signal	ignored here
nothing	λ or forbidden (*res*)	λ or forbidden (*res*)

We denote by $L^{\alpha}\beta NP_m^{\gamma}(poly^n(r), Var_{k_1}, Pro_{k_2})$, $\alpha \in \{res, \lambda\}$, $\beta \in \{E, pE, -\}$, $\gamma \in \{hal, fin\}$, the family of languages $L^{\alpha}(\Pi)$, generated by β numerical P systems Π (E = enzymatic, pE = purely enzymatic; if the system is non-enzymatic, then β is omitted) with at most m membranes, at most k_1 variables, and at most k_2 programs, with production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial; the superscript $\gamma = hal$ is used for purely enzymatic systems, to indicate that the result is obtained when the system reaches a halting configuration; in the case when the end of the computation is defined by means of a signal (sending value 0 out), then we replace *hal* by *fin*. If one of the parameters m, n, r, k_1, k_2 is not bounded, then we replace it with $*$.

Here are some of the results proved in [16]:

1. $L^{res}\beta NP_*^{\gamma}(poly * n(*), Var_*, Pro_*) \subseteq REC$, $\beta \in \{E, pE, -\}$, $\gamma \in \{hal, fin\}$.
2. $L^{res}NP_1^{fin}(poly^1(1), Var_1, Pro_2) - FIN \neq \emptyset$.
3. $REG \subseteq L^{res}NP_1^{fin}(poly^1(1), Var_*, Pro_*)$.
4. $L^{res}NP_1^{fin}(poly^1(4), Var_4, Pro_4) - REG \neq \emptyset$..
5. $L^{res}NP_1^{fin}(poly^1(4), Var_7, Pro_6) - CF \neq \emptyset$.
6. The family $L^{res}NP_1^{fin}(poly^1(4), Var_4, Pro_7)$ contain non-semilinear languages.
7. $L^{res}pENP_1^{hal}(poly^1(1), Var_2, Pro_2) - FIN \neq \emptyset$.
8. $FIN \subset L^{res}pENP_1^{hal}(poly^1(1), Var_*, Pro_*)$.
9. $REG \subseteq L^{res}pENP_1^{hal}(poly^1(2), Var_*, Pro_*)$.
10. $L^{res}pENP_1^{hal}(poly^1(1), Var_6, Pro_4) - REG \neq \emptyset$.

11. $L^{res}pENP_1^{hal}(poly^1(1), Var_9, Pro_6) - CF \neq \emptyset$.
12. The family $L^{res}pENP_1^{hal}(poly^1(1), Var_7, Pro_6)$ contains non-semilinear languages.
13. $RE = L^{\lambda}pENP_1^{hal}(poly^1(2), Var_*, Pro_*)$.
14. The family $L^{res}ENP_1^{hal}(poly^1(2), Var_4, Pro_6)$ contains non-semilinear languages.

P. The external variable can be useful also for considering an NP system as a decidability device: an instance of a decision problem is encoded in the values of certain variables, and the values of a specified variable – maybe the external one (which is not used in any production function) – at a well defined moment (in a halting configuration, if halting can be defined and ensured) is the yes/no answer to the problem instance. Using NP systems in this way, as decidability devices, is a general research topic of definite interest. Which is the efficiency of this approach? Can **NP**-complete problems be solved in polynomial time in this framework? If not, which ingredients can help?

Q. In general, what about NP systems with "active membranes", i.e., with possibilities of dissolving, creating, dividing membranes? Are these operations useful for speeding-up the computations?

R. Related also to the previous questions is the natural one of looking for interesting sequences of numbers and for interesting functions which can be computed by NP systems. Are there *hard* sequences/functions (hard with respect to Turing machines) which can be computed in a more efficient way with NP systems (maybe endowed with membrane manipulating rules)?

S. The answer to the previous question can have a practical interest, e.g., for robot controllers. In this context, an exercise is natural: passing from robots acting in a 2D space, as those considered so far in membrane computing area, to 3D robots (drones, satellites). This is, expectedly, only a programming issue/exercise, but of interest in view of the popularity of 3D machineries which need an automatic (maybe intelligent) controller.

T. In robot control there were useful numerical P systems with *enzymes* controlling the applicability of programs. A natural idea is to count the variables used as enzymes, then to try to keep this number as small as possible without diminishing the computing power (without losing the universality). The numbers of enzymes used so far in proofs is surprisingly large: For instance, the result in [11] can be written as

$$\begin{aligned} NRE &= N_{gen}E_*NP_*(poly^1(2), oneP) \\ &= N_{gen}E_{776}NP_{254}(poly^2(253), allP) \end{aligned}$$

(the subscript of E indicates the number of enzymes used) whereas the improvement of the last equality given in [10] can be written as

$$NRE = N_{gen}E_{427}NP_4(poly^1(6), allP).$$

The improvements of the above results in [4] are also not concerned with keeping under control the number of variables used as enzymes.

In [18], the following – again surprising – results were obtained:

$$\begin{aligned} NRE &= N_{acc}E_1NP_1(poly^1(2), allP) \\ &= N_{gen}E_2NP_1(poly^1(2), oneP) \\ &= N_{acc}E_1NP_1(poly^1(2), oneP). \end{aligned}$$

What other results about enzymatic numerical P systems remain to be improved from this point of view? (What about small universal numerical P systems?)

U. The previous problem is related to another way to control the use of programs, namely by means of *thresholds*, constants associated with programs, compared with the current values of variables in the production function or with the value of the production itself. See precise definitions in [19] and [15]. Universality results with a small number of thresholds are obtained in these papers.

4 Further Problems for SNP

V. In the same way as NP systems can compute (also for robot controllers) functions $f : \mathbf{N}^n \longrightarrow \mathbf{N}^m$, such a function can be computed also by SNP systems. Can such systems be used for designing robot controllers? Which is the (practical and theoretical) efficiency of such an approach?

W. On the one hand, the brain is supposed to be a non-Turing "computer", on the other hand, it is supposed to have a deterministic conscious part and a non-deterministic unconscious part, the first one problems problems to the latter, this one proposing solutions, which are evaluated by the conscious part, and the process is iterated until either finding the right solution, or the problem is abandoned. Can such a strategy be implemented in terms of SNP systems? Is it possible to devise such a hybrid SNP system able of computing beyond Turing?

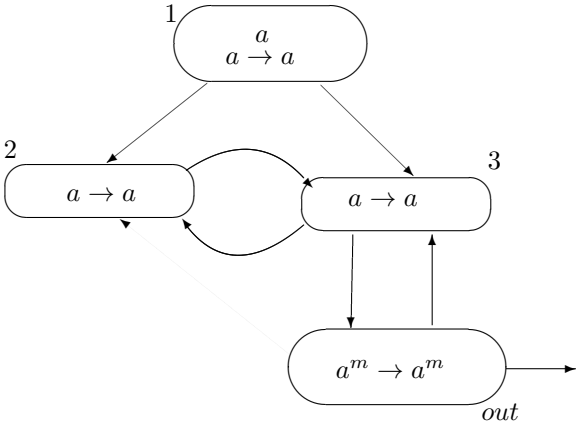
X. What about extending to SNP systems other ideas currently explored in hypercomputing, see, e.g., [9]? Can they be formulated for SNP systems in such a way to make them compute beyond Turing (as – again – the human brain is supposed to do)?

For instance, what about *accelerated SNP systems*, where the neurons "learn" during the functioning of the system. First time when a neuron uses a rule, the application of the rule lasts one time unit (the time is measured by an external clock, the *user clock*). Next time (does not matter how many steps the neuron is not working in between), the rule is applied in half of a user time unit – and so on, always half of the duration of the previous rule application.

Thus, a neuron which works each step, in two external time units will perform an infinity of steps.

The example in figure below shows an SN P system with only one spike inside, with neurons 1 and 4 working only once, but with neurons 2 and 3 working each step from step 2 on. Thus, in at most 2 external time units, neurons 2 and 3 send to neuron 4 any number of spikes. When m spikes are present in neuron 4, neuron 4 fires and the computation halts.

Thus, irrespective how large is m, starting with only one spike inside, this system will produce m spikes (sent outside) in at most 4 external time units (but working internally a number of steps which depends on m).



Conjectures:

1. Using the acceleration, we can solve **NP**-complete problems in polynomial time. (The first step is to find a suitable problem to be addressed in this framework.)
2. Accelerated P systems can go beyond Turing (can solve the halting problem – see the example of [1]); can this result be extended to SN P systems?

Both these conjectures are, metaphorically, supported by the fact that the brain is efficient and "non-Turing".

Y. Add to SNP systems further biologically-inspired features, to get closer to the brain. Ideally, bring enough further features to the SNP systems so that processes taking place in the "real" brain can be modeled/simulated (at the level of biologists interest).

Z. We have left to the end a very promising new class of SNP systems, which are no longer using regular expressions for controlling the application of spiking rules, but instead *polarizations* are associated with the neurons and the rules. The idea was explored in [12]. For the reader convenience, we recall the definition with full details.

A *spiking neural P systems with polarizations* (in short, a PSN P system) of degree $m \geq 1$ is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out),$$

where

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

$$\sigma_i = (\alpha_i, n_i, R_i), 1 \leq i \leq m,$$

   where:
   (a) $\alpha_i \in \{+, 0, -\}$ is the *initial polarization* of neuron σ_i;
   (b) n_i is the *initial number of spikes* contained in σ_i;
   (c) R_i is a finite set of *rules* of the following two forms:
      (i) $\alpha/a^c \to a; \beta$, for $\alpha, \beta \in \{+, 0, -\}, c \geq 1$ (*spiking rules*);
      (ii) $\alpha/a^s \to \lambda; \beta$, for $\alpha, \beta \in \{+, 0, -\}, s \geq 1$ (*forgetting rules*);
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for each $(i, j) \in syn, 1 \leq i, j \leq m$ (synapses between neurons);
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and *output* neurons, respectively.

Note that the definition of PSN P systems is the same as the usual definition of SN P systems given in the literature, with two differences: the applicability of a rule is not determined by checking the total number of spikes contained in the neuron against a regular expression associated with the rule, but the neurons have charges and a rule can be applied only if the neuron has the charge indicated in the left hand side of the rule. Of course, in order to use a rule, the total number of spikes inside the neuron should not be less than the number of spikes consumed by the rule. Moreover, the neurons not only send out spikes, but also charges, even when using forgetting rules.

A spiking rule $\alpha/a^c \to a; \beta$ is used as follows. If the neuron σ_i has the charge α and it contains at least c spikes, then the rule can be applied, and its application means that c spikes are consumed, the neuron fires and produces a spike, which carries the charge β. The spike is replicated and each neuron σ_j such that $(i, j) \in syn$ receives the spike and the charge β.

The output neuron also sends spikes out of the system, but no electrical charge is sent out (it is "lost" in the environment).

A forgetting rule $\alpha/a^s \to \lambda; \beta$ is applied when the neuron has the charge α and contains at least s spikes; s spikes are removed from the neuron and the charge β is sent to all neurons σ_j such that $(i, j) \in syn$. (Note that we do not necessarily forget all spikes, as in the case of usual SN P systems, where exactly s spikes should be present in order to use a forgetting rule $a^s \to \lambda$.)

After a neuron receives charges from other neurons, we perform a computation of charges inside the neuron as described below:

1. several positive charges (+), several neutral charges (0), several negative charges (−) lead to one positive charge (+), one neutral charge (0), one negative charge (−), respectively;
2. a positive charge (+) and a negative charge (−) cancel each other and give the neutral charge (0);
3. a positive charge (+) or a negative charge (−) is not changed by a neutral charge (0).

We stress that (i) the computation of charges takes no time; (ii) step 1 of the above computation of charges is done first. For example, if a given neuron which is initially neutral receives two positive charges and one negative charge, then first the two positive charges lead to one positive charge, after that the positive charge and the negative charge cancel each other, thus the neuron remains neutral.

As usual in SN P systems, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized. In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. If several rules can be used at the same time in a neuron, then the one to be applied is chosen non-deterministically. Thus, the rules are used in a sequential manner in each neuron, but the neurons function in parallel with each other.

The *configuration* of the system is described by both the number of spikes and the charge of each neuron; thus, the *initial configuration* of the system is $C_0 = \langle n_1, n_2, \ldots, n_m; \alpha_1, \alpha_2, \ldots, \alpha_m \rangle$. Using the rules as described above, one can define *transitions* among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Rightarrow C_2$. Any sequence of transitions starting from the initial configuration is called a *computation*. A computation is *successful* if it reaches a configuration where no rule can be used in any neuron of the system. We say that the computation is *halting*.

A PSN P system can be used as a generative, an accepting, or a computing device.

With any computation, halting or not, we associate a *spike train*, the sequence of symbols 0 and 1 describing the behavior of the output neuron: 1 indicates a spiking step, 0 indicates a step when no spike exits the system. With a spike train, a *result of a computation* can be defined in several ways. For instance, the result of a computation can be defined as usual in general SN P systems: we only consider the first two time instances t_1 and t_2 that neuron *out* spikes and we say that the number $t_2 - t_1$ is computed/generated by Π. The set of all numbers generated in this way by a PSN P system Π is denoted by $N_2(\Pi)$ (the subscript 2 indicates that the computation result is encoded by the time distance between the first two spikes of any computation).

In the generative case, the neuron with label *in* is ignored. In the accepting mode, the neuron with label *out* is ignored. A number n is introduced in the system, by introducing a sequence $10^{n-1}1$ in neuron *in* (two spikes are introduced, at a time distance of n steps) and this number is accepted if the computation halts.

When both an input and an output neuron are considered, the PSN P systems can be used to compute numerical functions. In order to compute a function f :

$\mathbf{N}^k \to \mathbf{N}$, k natural numbers $n_1, \ldots, n_k$ are introduced into the system by "reading" from the environment a spike train of the form $z = 10^{n_1-1}10^{n_2-1}1 \ldots 10^{n_k-1}1$. Note that exactly $k+1$ spikes are "read", that is, after the last spike, it is assumed that no further spike is sent to the input neuron. The result of the computation is also encoded as the distance between the first two spikes emitted by the output neuron with the restriction that the system outputs exactly two spikes and halts (maybe some further steps after the second spike), hence it produces a spike train of the form $0^b10^{r-1}10^d$ for some $b, d \geq 0$ with $r = f(n_1, n_2, \ldots, n_k)$. The system outputs no spike in the $b \geq 0$ steps from the beginning of the computation until the first spike.

We denote $N_2PSNP(ch_p)$ the family of all sets of numbers $N_2(\Pi)$ generated by PSN P systems with at most p charges.

The two results proved in [12] are the following:

1. $NRE = N_2PSNP(ch_3)$.
2. There exists a universal PSN P system (with three charges) for computing functions, having 164 neurons.

The proofs are rather complex, at least in comparison with the proofs of the corresponding results for usual SN P systems, and this is due to the fact that the polarizations provide a much weaker control on the applicability of the rules in neurons than the regular expressions.

Again, many research topics remain to be explored. Practically the whole program of investigation carried on usual SN P systems has to be explored also for the new type of spiking neural P systems: normal forms (can we get rid of forgetting rules?), using extended rules (producing more than one spike can help, e.g., in simplifying the proofs?), generating strings or infinite sequences, considering asynchronous computations or a parallel/exhaustive use of spiking rules in each neuron, adding astrocytes or other biology inspired ingredients, and so on and so forth. Another idea is to consider cell-like PSN P systems; polarized cell-like SN P systems seem to be challenging to investigate (maybe not universal).

There also appear specific open problems. Of a definite interest is the question whether or not the number of electrical charges considered in the universality proof from [12], three, can be decreased. Which is the power of PSN P systems with 2 charges, or even without any charge? Is any of the corresponding classes of computing devices sub-universal? If so, which are the properties (size, closure, decidability) of the corresponding family of sets of numbers or of languages generated? Finally: can the number of neurons in universal PSN P systems be (significantly) decreased? (We are pessimistic about this, as clever codifications in terms of the number of spikes, as usual for standard SN P systems, do not seem to help in the absence of regular expressions.)

Definitely, we believe that SN P systems with polarizations deserve further research efforts.

Acknowledgments. This work of T. Wu and Z. Zhang was supported by the National Natural Science Foundation of China (61033003, 91130034, and

61320106005), Ph.D. Programs Foundation of Ministry of Education of China (2012014213008), and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

References

1. C. Calude, Gh. Păun: Bio-steps beyond Turing, *CDMTCS Research Report* 226, Univ. of Auckland (November 2003), and *BioSystems*, 77 (2004), 175–194
2. J. Dassow, G. Vaszil: P finite automata and regular languages over countable infinite alphabets. *Proc. WMC 2006, Leiden, The Netherlands* (H.J. Hoogeboom et al., eds.), LNCS 4361, Springer, 2006, 367–381.
3. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
4. A. Leporati, A.E. Porreca, C. Zandron, G. Mauri: Improving universality results on parallel enzymatic numerical P systems. *Proceedings of 11th Brainstorming Week on Membrane Computing*, Sevilla, February 2013, Fenix Editora, Sevilla, 2013, 177–200.
5. A. Leporati, A.E. Porreca, C. Zandron, G. Mauri: Enzymatic numerical P systems using elementary arithmetic operations. *Membrane Computing. Proc. 14th Intern. Conf., CMC2013, Chişinău, August 2013*, LNCS 8340 (A. Alhazov et al., eds.), Springer, Berlin, 2014, 249–264.
6. A.B. Pavel, C.I. Vasile, I. Dumitrache: Robot localization implemented with enzymatic numerical P systems. *Proc. Conf. Living Machines 2012*, LNCS 7375, Springer, 2012, 204–215.
7. Gh. Păun, R. Păun: Membrane computing and economics: Numerical P systems. *Fundamenta Informaticae*, 73 (2006), 213–227.
8. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
9. A. Syropoulos: *Hypercomputation: Computing Beyond the Church-Turing Barrier*. Springer, Berlin, 2008.
10. C.I. Vasile, A.B. Pavel, I. Dumitrache: Universality of enzymatic numerical P systems. *International Journal of Computer Mathematics*, 90(4) (2013), 869–879.
11. C.I. Vasile, A.B. Pavel, I. Dumitrache, Gh. Păun: On the power of enzymatic numerical P systems. *Acta Informatica*, 49(6) (2012), 395–412.
12. T. Wu, A. Păun, Z. Zhang, L. Pan: Spiking neural P systems with polarizations. Submitted, 2015.
13. T. Wu, Z. Zhang, L. Pan: On string languages generated by cell-like spiking neural P systems. Submitted, 2015.
14. T. Wu, Z. Zhang, Gh. Păun, L. Pan: Cell-like spiking neural P systems. Submitted, 2015.
15. Z. Zhang, L. Pan: Numerical P systems with production thresholds. Submitted, 2015.
16. Z. Zhang, T. Wu, L. Pan, Gh. Păun: On string languages generated by numerical P systems. Submitted, 2015.
17. Z. Zhang, T. Wu, A. Păun, L. Pan: Numerical P systems with migrating variables. Submitted, 2015.
18. Z. Zhang, T. Wu, A. Păun, L. Pan: Universal enzymatic numerical P systems with a small number of enzymatic variables. Submitted, 2015.
19. Z. Zhang, J. Xu, L. Pan: Numerical P systems with thresholds. Submitted, 2015.

Individual memory about the 14^{th} Brainstorming Week on Membrane Computing

Ariadna Ribes Metidieri

Universitat de Barcelona
Email: aribesmetidieri@gmail.com

Summary. The main objective of this memory is to stand out one of the research methods for developing new P system models observed during the 14^{th} Brainstorming Week on Membrane Computing. Firstly, a general overview of P systems is provided. To continue, the use of register machines in order to justify completeness and universality is justified. And to end up, an example of the method is provided.

1 Motivation and experience

The motivation for investigating further into this subject arose when observing that new computational model could be proposed. However, computability (the model's capability of acting as a computer) was always required, meaning that each new proposed model was tested versus Turing completeness. i.e., every proposed model should be demostrated to be capable of performing a computation.

The proof of the universality of the P systems can be attained using different methods, but the utilization of the Register Machines was widely promoted. Moreover, through '*Rudi's fancy homework*' we learned that register machines were in fact, simple but really useful interesting devices.

2 General Overview

Membrane computing is a biologically-inspired research branch in the field of computer science which starts from the assumption that processes taking place in the structure of a living cell can be interpreted as a computation and it gathers the study of different kinds of P systems. P systems are the devices used in this new computing paradigm which performs calculations based on the idea of a hierarchical arrangement of membranes acting as channels of communication. These systems are inspired in cellular structures, being the cell-like, tissue-like and Spiking Neural P systems current developed models [1]. All three models are based on cells, but seems important to recall that they are formal models which should not be considered as representations of the truth.

Membrane computer models share the same structure composed by membranes, objects, catalysts and a multiset of rules (i.e, evolution, communication, dissolution or division rules)[2] which are applied on the objects in each region delimited by a membrane. The computation works from an initial starting state or configuration to an end state through a number of discrete steps or transitions between configurations. The evolution rules are used in a non-deterministic and maximally parallelism way, i.e., in any computational step of the P system Π, a multiset of rules from the sets R_1,...,R_m is chosen in a non-deterministic way such that no further rule can be added to it. The obtained multiset will still be applicable to the existing objects in the membrane regions 1, . . . ,m. When no more rules can be applied, the computation ends (it is said to halt), leaving the result of the process in a given membrane or in the environment.[3]

Membrane computing was first developed in order to solve NP-complete problems. The research in this field moves in two different directions. On the one hand, theoretical models are being developed. This branch of research tries to find a theoretical foundation for new P system models and works in computational complexity, which tries to find an efficient solution to hard problems and works on the P conjecture. On the other hand, a practical approach is postured, including simulations in silico (for example, using MeCoSim)[4] as well as research in order to implement P systems in vitro.

3 Register machines as reference model for computational completeness and universality

Most P system variants (such as purely catalytic P systems, extended Spiking Neural P systems, P systems with anti-matter...) can be demonstrated to be computationally universal or Turing complete, i.e, the system of data-manipulation rules can be used to simulate a single-taped Turing machine.

A Turing machine is a hypothetical device with an infinite memory capacity, which manipulates symbols on a supposedly infinite strip of tape according to a set of rules. The Church-Turing thesis conjectures that any function whose values can be computed by an algorithm can be computed by a Turing machine, and therefore that any real computer is equivalent to a Turing machine.

The register machines are known to be computationally complete and equal in power to (non-deterministic) Turing machines. Consequently, register machines provide a simple universal computational model, which can be used to provide the proofs of the computational completeness of P systems based on the simulation of this kind of machines.

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

- l_1: $(ADD(j), l_2, l_3)$ with $l_1 \in B \backslash \{l_h\}, l_2, l_3 \in B, 1 \leq j \leq m$.
  Increases the value of the register j by one, followed by a non-deterministic jump to instructions l_2 or l_3. This instruction is usually called *increment*.
- l_1: $(SUB(j), l_2, l_3)$ with $l_1 \in B \backslash \{l_h\}, l_2, l_3 \in B, 1 \leq j \leq m$.
  If the value of the register j is 0 then jumps to l_3 (instruction called *zero-test*), otherwise the value of the register j is decreased by one, followed by a jump to instruction l_2 (*decrement*).

- l_2: HALT: stops the execution of the register machine.

A specific model of a P system should be called computationally complete or universal if for any (generating, accepting, computing) register machine M we can effectively construct an equivalent P system Π of that type simulating each step of M in a bounded number of steps and yielding the same result.[5]

Once a new P system model has been proposed, the main goal to achieve is to determine that effectively it can perform all the calculations computable by a real computer and not just the operation it was first thought to perform.

The rule complexity of universal P systems depends on the objects as well as on the specific types of rules.

3.1 Example: SN P systems with States

Let's consider a particular SN P system with states and a single neuron stP_1 Π. It's formal definition is given by

$$\Pi = (1, O = O_T = \{a\}^*, Q = B, \delta, f_I, f_O, q_i = l_0, F = l_h, C_i = 0) \tag{1}$$

The stP_1 starts with the initial configuration computed by the input function f_I, the initial state $q_i = l_0$ and the input object $a \in O$, which are equal to the set of terminal objects O_T. The transitions between configurations and states are computed by δ to the new ones until the computation reaches a final state $f = l_h \in F$.

The computations of the register machine $M = (m, B, l_0, l_h, P)$ can be simulated by the stP_1 Π working with multisets as follows (the states of a single neuron represent the instruction labels of the register machine) [6]

$$\delta(p, (w)) = \{(\{q, s\}, \{(a \to a^{p_r}, maxpar)\})\} \tag{2}$$

for p: (ADD(r),q,s) $\in$P, w$\in \{a\}^*$

$$\delta(p, (w)) = \{(q, \{(a^{p_r} \to a, maxpar)\})\} \tag{3}$$

for p: (SUB(r),q,s)$\in$P, $p_r/|w|$

$$\delta(p, (w)) = \{(s, 0)\} \tag{4}$$

for p: (SUB(r),q,s) $\in$ P, not $p_r/|w|$

To sum up, as can be seen from the example above, a SN P system acting in the maximally parallel derivation mode (*maxpar*) is in fact computationally complete, as the rules which define the system can be simulated using a Turing machine.

References

1. Gh. Păun *Membrane Computing: An Introduction*
2. Claudio Zandron, Alberto Leporati, Luca Manzoni, Giancarlo Mauri and Antonio E. Porreca *P systems with Active Membranes working in Sublinear Space*
3. Mario J. Pérez-Jiménez *A Bioinspired Computing Approach to Model Complex Systems*

4. J. M. Cecilia, J. M. García, G. D. Guerrero, M. A. Martínez-del-Amor, I. Pérez-Hurtado, M. J. Pérez-Jiménez *Simulation of P systems with active membranes on CUDA*
5. Artiom Alhazov, Rudolf Freund, and Sergey Verlan *Computational Completeness of P Systems Using Maximal Variants of the Set Derivation Mode*
6. Artion Alhazov, Rudolf Freud, Sergiu Ivanov and Marion Oswald *P systems with States*

Memory about the 14 th BWMC: SN P systems vs. ESN P systems with Transmittable States

Patricia Ribes Metidieri

Universitat de Barcelona
Email: ribesmetidieri@gmail.com

Summary. The objectives of this memory are, on one hand, to provide a general overview about the topic of Membrane Computing, answering basic questions as what is it, which are its basic elements, which problems it allows to solve, its current limitations... and, on the other hand, to provide a more specific information about the model of Spiking Neural P systems in both its original formulation and on the variant of the model suggested on the 14th edition of the Brainstorming Week on Membrane Computing, the Extended Spiking Neural P systems with Transmittable States.

The motivation to further explore the topic of Spiking Neural P systems (SN P systems for short) comes from the idea that they could be a really suitable framework in order to model chain-reaction processes as the fission of ^{235}U taking place inside a nuclear reactor.

1 General overview

Membrane computing is a branch of Natural computing, which, based on the idea that the processes within a cell can be interpretated as computations, abstracts computing models from the architecture of living cells and their organisation in tissues, organs, ... Devices of this model are called P systems and are defined as cell-like structures consisting on a set of hierarchically arranged membranes where multisets of objects (that represent the chemicals in the cell) evolve following sets of rules which only apply in the membrane where they are implemented. [1]

Just as in a biological cell, the rules in a P system are applied at random, which often results in multiple solutions being encountered if the computation is repeated, thus P-systems are non-deterministic. Furthermore, all the rules which can be applied in a computation step are executed whenever they are applicable, so the system evolves in each step until no more rules can be implemented, i.e. the rules are applied in a maximally parallel manner.

A computation in this kind of system is a sequence of instantaneous transitions between configurations guided by the rules executed, until it reaches a state where

no further reactions are possible, which marks the end of the computation. Its result is all those objects contained in a particular 'results' membrane.

So far, the implementation of this biological computers "in vitro" has not been achieved and technical problems such as the fact that the theoretical model is based on the assumption that all the reactions within each step of computation lasts the same time while within a real cell each reaction is completed in a characteristic time, has not yet been solved.

However, on the lasts years, simulators as MeCoSim have been developed [2] and, within the computational complexity theory, Membrane Computing has proven itself a really powerful tool to solve NP-complete problems in polynomial even linear time. Finally, P-systems have also been used lately to model biological phenomena within the framework of Computational Systems Biology.

2 Spiking Neural P Systems: basic concepts

Spiking Neural (SN) P systems are a class of neural-like P systems in which one-membrane cells (called *neurons*) placed in the nodes of a directed graph send electric signals or spikes (represented by a single object denoted s) under certain conditions (given by the rules defined on each cell) along their axons to all the neurons connected by *synapses* (arcs of the graph). [3]

A computation in this model starts by fixing the number of spikes in the input neurons, which propagate the spikes through the net of connected cells. Each neuron fires after having accumulated a certain number of spikes and the result of the computation is the number of steps elapsed between two spikes of the labeled output neuron have been sent to the environment.

It is interesting to recall that the rules defined in each neuron are applied in a non-deterministic but sequential mode with at most one rule used in each step, though the maximal parallelism condition is implemented at a system level, since in each step of the computation all neurons which can evolve (use a rule) have to do it. In this model only two different kind of rules are defined: *firing or spiking rules* and *forgetting rules*.

On the one hand, spiking rules are of the form $E/s^r \rightarrow s;t$, where E represents the required content of the neuron for the rule to apply and $r \geq 1$, the spikes consumed when the rule is executed. Once the neuron is fired, it produces a spike which will be sent to other neurons after $t \geq 0$ time steps and the cell is assumed closed in the period between firing and spiking (which represents the *refractory period*).

On the other hand, forgetting rules are of the form $s^k \rightarrow \lambda$, which means that, when the neuron contains exactly $k \geq 1$ spikes, k spikes are removed ("forgotten").

Finally, on [4] SN P systems are shown to be *computationally complete* when the number of spikes inside each neuron is not bounded, i.e., when the cell can accumulate an arbitrary number of spikes inside. However, it is also shown that, if a more realistic model with a bounded number of spikes present in any neuron

along a computation is considered, these devices lose their capacity to generate all Turing computable set of numbers.

3 Extended Spiking Neural P systems with Transmittable States

Since the emergence of the SN P systems in [4] in 2006, many variants of this model have been proposed. Particularly, in the 14th edition of the Brainstorming Week on Membrane Computing a new variant was suggested, so in this section a brief analysis of the new proposals is discussed, explaining its advantages from the point of view of the computational completeness.

Formally, Extended Spiking Neural P systems with Transmittable States are defined [5] as

$$\Pi = (O, Q, \sigma_1, \sigma_2, \ldots, \sigma_n, in, out) \tag{1}$$

being

- $O = \{s\}$ the objects conforming the system, with s a spike.
- Q is a finite alphabet of states, where $polarizations \subseteq states$
- σ_i are the neurons of the systems, defined as $\sigma_i = (\alpha_i, n_i, f_i, R_i)$ where
  - $\alpha \in Q$ is the initial state.
  - n_i is the initial number of spikes
  - f_i is the state combining function
  - R_i is a finite set of rules of the form

$$\alpha/a^c \rightarrow (t_1, a^{k_1}, \beta_1), \ldots, (t_m, a^{k_m}, \beta_n) \tag{2}$$

    1. send k_j spikes to neuron t_j after having accumulated a^c spikes, for $1 \leq j \leq m$,
    2. send state b_j to neuron σ_j, combine them using f_i and set the new state of neuron σ_i, combine them using f_i, and set the new state of neuron σ_i.

  and

$$\alpha/a^c \rightarrow \lambda \tag{3}$$

Comparing this definition with the given for SN P systems on the previous section, the new contributions can be appreciated. Firstly, the firing rules incorporate the elements α and β, which denote the initial and final states of a neuron after each step of the computation. This incorporation modifies the process of spiking, since now not only a spike, but the state of the neuron, is sent along the axon in the firing process.

Secondly, the state combining function, f_i is added in the definition of the neurons conforming the system in order to compute the state of the neuron receiving states of all those connected to it through synapses.

For instance, if the states being passed are polarisations $Q = \{+, -, 0\}$ (which represent the electrical charge of the membrane), a possible state combining function would assign positive polarisation to the neuron receiving the spikes if more positive than negative polarisations were received, negative polarisation in the opposite case and null polarisation if the number of received + and − polarisations were the same.

Finally, the time between firing and spiking is 0 for all the neurons.

In contrast with SN P systems, where computational completeness is achieved by allowing the non-realistic notion that the number of spikes within each cell is not bounded, the ESN P systems with Transmittable States are powerful enough for universal computation considering the number of spikes in each active neuron reduced to 1 and replacing the unbounded number of spikes to an unbounded number of neurons.

This model can reproduce the rules of Conway's Game of Life using only 2 states $Q = \{0, 1\}$ (which represent "dead" and "alive") and, since this kind of cellular automata has the power of a universal Turing machine, by reproducing the rules of the Game, the computational completeness of the ESN P systems with Transmittable States is proved.

References

1. Gh. Păun *Membrane Computing: An Introduction*
2. J. M. Cecilia, J. M. García, G. D. Guerrero, M. A. Martínez-del-Amor, I. Pérez-Hurtado, M. J. Pérez-Jiménez *Simulation of P systems with active membranes on CUDA*
3. J. Wang, L. Zou, H. Peng, G. Zhang *An extended Spiking Neural P System for fuzzy knowledge representation*
4. M. Ionescu, Gh. Păun, T. Yokomori *Spiking Neural P Systems*
5. R. Freud, S. Ivanov *Extended SNP systems with States*

On the complexity of active P systems

Gábor Román

Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary
Email: romangabor@caesar.elte.hu

Summary. We are going to present a polynomially uniform solution to the Quantified 3SAT decision problem with restricted instances where the quantifiers alternate, based on recognizer P systems with active membranes and no input membrane, having three polarizations using only dissolution and division rules.

1 Introduction

In the twelfth chapter of "The Oxford Handbook on Membrane Computing" [3] the following question can be found: What is the efficiency of P systems with active membranes and electrical charges where evolution and communication rules are forbidden? The answer to this question is that one can give a uniform solution to the **PSPACE**-complete Quantified 3SAT decision problem (having a restricted quantification, which does not alter its complexity class) using such systems. Similar result is obtained by Alberto Leporati et al. in their [1] article. They gave a semi-uniform solution for the Q3SAT decision problem using polarisationless P systems.

In the second section we will recall the definition of the recognizer P systems with active membranes, together with the definition of uniform solution. In the third section, the Q3SAT decision problem will be defined. In the fourth section we will describe the main result of this paper, namely the uniform solution to the restricted Q3SAT decision problem. In the fifth section we are going to draw the conclusions.

2 Recognizer P systems with active membranes

We are going to use P systems with the above mentioned properties through the rest of the paper, so now we give the definition of such systems. For more detailed description see [2].

Definition 1. *A P system with active membranes, having three polarizations using only dissolution and division rules, of degree $q \geq 1$ is a tuple*

$$\Pi = (\Gamma, H, \mu, w_1, \ldots, w_q, h_0, R)$$

where

- *Γ is the finite alphabet of objects,*
- *H is the alphabet of labels for the membranes,*
- *μ is the initial membrane structure of degree q, with all membranes labeled with the elements of H and with electrical charges (positive, negative or neutral) associated with them,*
- *$w_1, \ldots, w_q$ are strings over Γ specifying the multisets of objects present in the compartments of μ,*
- *$h_0 \in \{0, 1, \ldots, q\}$ indicates the region where the result of a computation is obtained (0 represents the environment),*
- *and R is a finite set of rules.*

The rules are of the following types.

- *Dissolution rules of the form*

  $$[a]_h^\alpha \to b$$

  where $h \in H$, $\alpha \in \{+, -, 0\}$ and $a, b \in \Gamma$. Here, every membrane in the membrane with label h together with every other object in it goes into the upper neighbor.
- *Division rules for elementary membranes:*

  $$[a]_h^{\alpha_1} \to [b]_h^{\alpha_2} [c]_h^{\alpha_3}$$

  where $h \in H$, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$ and $a, b, c \in \Gamma$. Here, every membrane and other objects in the initial h labeled membrane are copied into both newly created h labeled membranes.
- *Division rules for non-elementary membranes:*

  $$[[\ \ ]_{h_1}^{\alpha_1} \cdots [\ \ ]_{h_k}^{\alpha_1} [\ \ ]_{h_{k+1}}^{\alpha_2} \cdots [\ \ ]_{h_n}^{\alpha_2}]_h^\alpha$$
  $$\to$$
  $$[[\ \ ]_{h_1}^{\alpha_3} \cdots [\ \ ]_{h_k}^{\alpha_3}]_h^\beta \quad [[\ \ ]_{h_{k+1}}^{\alpha_4} \cdots [\ \ ]_{h_n}^{\alpha_4}]_h^\gamma$$

  for $k \geq 1$, $n > k$, $h, h_1, \ldots, h_n \in H$, $\alpha, \beta, \gamma, \alpha_1, \ldots, \alpha_4 \in \{+, -, 0\}$ and $\{\alpha_1, \alpha_2\} = \{+, -\}$. Here, every object and the membranes with neutral polarity in the h labeled membrane are copied into both newly created h labeled membranes.

A configuration in a P system can be described by its actual membrane structure together with the multisets of objects present in the regions. A computational step changes the current configuration according to the following principles.

- Each membrane can be subject to at most one rule per computation step. Newly created membranes cannot be the subjects of rules in the actual computational step. The skin membrane should not dissolve or divide.
- The rules are applied in a maximally parallel manner. This means, that every membrane which could be the subject of a rule must be the subject of exactly one rule. When there is more than one rule which we can apply, then the choice should be nondeterministic.
- The rules are applied "from bottom up", so first the rules are applied on the innermost membranes, then on their upper neighbors, and so on until the skin membrane.

We are going to use a recognizer P system. This means, that the Γ alphabet has two distinguished objects representing "yes" and "no", and if one of these objects reach the membrane with label h_0, then the computation halts. The result of the computation is acceptance in the former-, and rejection in the later case.

The computation of such P system is the sequence of its configurations starting from its initial configuration. Every configuration of such computation should be reached from the previous configuration using the principles described above. Such computation can be finite, arriving to a configuration where one of the "yes" or "no" objects enter the h_0 labeled membrane, or it can be infinite if this does not happens. We will only consider confluent recognizer P systems, in which all computations starting from the initial configuration halt and agree on the result.

We are going to build the initial membrane structure according to the given instance of the examined problem. We will show, that this can be done in a polynomial amount of steps which means that our solution is polynomially uniform. In the following definition, I_X denotes the possible instances for the problem X.

Definition 2. *A family $\Pi = \{\Pi(w)|w \in I_X\}$ of recognizer membrane systems without input membrane is polynomially uniform by Turing machines if there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(w)$ for the instance $w \in I_X$.*

3 The Quantified 3SAT decision problem

The Boolean satisfiability problem (abbreviated as SAT) can be stated as the following. Lets consider the $x_1, \ldots, x_n$ Boolean variables. An instance of SAT consists of conjunctions of clauses, which are disjunctions of literals, occurrences of x_i or $\neg x_i$. An interpretation of the variables is a mapping, which associates a truth value to the variables. The Boolean satisfiability problem asks the following question: is there an interpretation of the given Boolean variables for which interpretation the conjunction of the clauses evaluates to true? For the rest of the paper, we assume that the literals in the clauses are ordered by the indexes of their variables.

The 3SAT decision problem is a variant of the SAT problem, where the clauses contain only three literals. An instance of the Quantified 3SAT decision problem is

a well-formed Boolean formula $(Q_1 x_1) \dots (Q_n x_n)\, \phi\, (x_1, \dots, x_n)$, where $Q_i \in \{\exists, \forall\}$ and ϕ is an instance of 3SAT over the variables $x_1, \dots, x_n$. This decision problem asks that the given quantified formula is true or false. It can be shown that the Q3SAT decision problem is **PSPACE**-complete, even when restricted to instances where the quantifiers alternate the

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots \exists x_{2i-1} \forall x_{2i} \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$$

way, where n is even.

4 Solving Q3SAT with restricted instances

We are going to describe a recognizer P system without an input membrane, which decides the satisfiability of a given instance of the restricted Q3SAT decision problem. With the following initial membrane structure construction, the number of objects and the number of rules, this will give us a polynomially uniform solution to the restricted Q3SAT decision problem. The required objects, rules and the initial membrane structure together with the initial membrane contents will be given as we describe the system part by part.

4.1 Construction of the initial membrane structure

The initial membrane structure can be seen in figure 1. For the ith universally quantified variable, we introduce the membranes with ε_{t_i} and ε_{f_i} labels having neutral polarity, together with two membranes with δ label having positive polarity, except for the last variable, where we only introduce one such δ labeled membrane. The membranes labeled ε and δ give us $2n - 1$ membranes in the initial membrane structure.

The clauses are encoded in the membranes with C_{i_p,j_p,k_p} labels. We are going to call the nested membrane structure of the membranes representing the clauses a clause-chain. The encoding is similar to the one which is used by Porreca et al. in [4] and [5]. We can represent the $C_p = (l_{p,1} \vee l_{p,2} \vee l_{p,3})$ clause with a membrane labeled C_{i_p,j_p,k_p} having neutral polarity, where i_p (resp. j_p and k_p) is the index of the variable in $l_{p,1}$ (resp. $l_{p,2}$ and $l_{p,3}$) with a negative sign if the variable is negated. So for example if our clause is $(x_1 \vee \neg x_2 \vee x_3)$, then the corresponding membrane will be the label $C_{1,-2,3}$. Using this encoding, the upper bound on the number of membranes with C_{i_p,j_p,k_p} labels in our initial membrane structure is $8\binom{n}{3}$. Also, the rules for these membranes can be given in advance (we will do this in section 4.4), so for a restricted Q3SAT instance we only have to construct the initial membrane structure.

The steps required for the generation of the interpretations are $n + 1$ and one step is required for the evaluation of the quantified formula. So the c labeled membranes should form a chain of polynomial length greater than $n + 2$. We will

discuss the reason for this in more details in section 4.4, but the short explanation is that this way, the n object will arrive to the skin membrane right on time.

Summing up the parts, one can see that the size of the initial membrane structure is polynomially bounded.

4.2 Creation of the interpretations

We included the d_1 object in the initial membrane structure. This object initiates the creation of the interpretations. We are going to use the

$$[d_i]_h^0 \to [d_{i+1}]_h^0[e_{n+1-i}]_h^0 \quad (i = 1, \ldots, n) \tag{1}$$

$$[d_{n+1}]_h^0 \to [d]_h^0[d]_h^0 \tag{2}$$

$$[d]_h^0 \to d \tag{3}$$

rules to generate the e objects. These objects will create the variable interpretations with the

$$[e_i]_h^0 \to [t_i]_h^+[f_i]_h^- \quad (i = 1, \ldots, n) \tag{4}$$

$$[t_i]_h^0 \to t_i \quad (i = 1, \ldots, n) \tag{5}$$

$$[f_i]_h^0 \to f_i \quad (i = 1, \ldots, n) \tag{6}$$

rules. The membranes with labels h and b and the membranes representing the clauses are split with the

$$[[\quad]_h^+[\quad]_h^-]_b^0 \to [[\quad]_h^0]_b^+[[\quad]_h^0]_b^- \tag{7}$$

$$[[\quad]_b^+[\quad]_b^-]_{C_{i_1,j_1,k_1}}^0 \to [[\quad]_b^0]_{C_{i_1,j_1,k_1}}^+[[\quad]_b^0]_{C_{i_1,j_1,k_1}}^- \tag{8}$$

and

$$\begin{gathered}[[\quad]_{C_{i_{p-1},j_{p-1},k_{p-1}}}^+[\quad]_{C_{i_{p-1},j_{p-1},k_{p-1}}}^-]_{C_{i_p,j_p,k_p}}^0 \\ \to \\ [[\quad]_{C_{i_{p-1},j_{p-1},k_{p-1}}}^0]_{C_{i_p,j_p,k_p}}^+[[\quad]_{C_{i_{p-1},j_{p-1},k_{p-1}}}^0]_{C_{i_p,j_p,k_p}}^-\end{gathered} \tag{9}$$

$p = 2, \ldots, m$ rules.

Lemma 1. *Starting from the initial membrane structure, applying rules (1)-(9) from step to step, after the (5) and (6) rules are applied on the* $[t_1]_h^0$ *and* $[f_1]_h^0$ *membranes, the membrane structure contains all the possible interpretations of the variables in the leaves.*

Proof. We are going to give a proof by induction. Figure 2 shows the beginning of the creational process. In the general case, we are going to look at a clause-chain with the

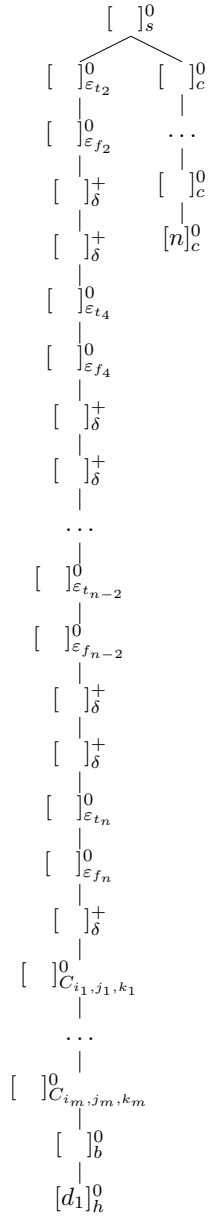


Fig. 1. The initial state. The membrane with the h label is the hatchery, we create the interpretations of the variables here. The membrane with the b label is a boundary membrane. The membranes with the C_{i_p,j_p,k_p} labels are the ones that evaluate the clauses. The δ membranes manage the creation of the quantification tree. The ε membranes check the universal quantifiers. The c contradictory membranes delay the entering of the n symbol into the skin membrane.

$$[d_i]_h^0[e_{n+2-i}]_h^0[t_{n+3-i}]_h^+[f_{n+3-i}]_h^-x_nx_{n-1}\dots x_{n+4-i} \tag{10}$$

membranes in the innermost membrane labeled b, where $i = 4, \dots, n$ and x_p could be either t_p or f_p. The polarity difference induces a non-elementary membrane division according the (7) rule. In the membrane with label b and positive polarity, we will have the

$$[d_i]_h^0[e_{n+2-i}]_h^0[t_{n+3-i}]_h^0x_nx_{n-1}\dots x_{n+4-i} \tag{11}$$

content and in the membrane with label b and negative polarity we will have the

$$[d_i]_h^0[e_{n+2-i}]_h^0[f_{n+3-i}]_h^0x_nx_{n-1}\dots x_{n+4-i} \tag{12}$$

content. The non-elementary membrane divisions are propagated upwards in the structure according the rules (8) and (9), forming two clause-chains. After the divisions stopped, a new step begins and (11) becomes

$$[d_{i+1}]_h^0[e_{n+1-i}]_h^0[t_{n+2-i}]_h^+[f_{n+2-i}]_h^-x_nx_{n-1}\dots x_{n+4-i}t_{n+3-i}$$

and (12) becomes

$$[d_{i+1}]_h^0[e_{n+1-i}]_h^0[t_{n+2-i}]_h^+[f_{n+2-i}]_h^-x_nx_{n-1}\dots x_{n+4-i}f_{n+3-i},$$

so in the innermost membrane with label b of both clause-chains, the same state appeared as in (10) just with the additional truth objects t_{n+3-i} and f_{n+3-i}. Note that this way, every possible $x_nx_{n-1}\dots x_{n+4-i}$ variable interpretation is extended with the mentioned truth objects. This holds in every iteration of the recursion.

At the end, the t_1 and f_1 objects are generated, the (5)-(6) rules are applied, and the t_1, f_1 objects enter the b membranes. At this point, every possible interpretation is generated at the bottom of the membrane structure. Because of the (2) rule, there will be no more e objects introduced into the membrane structure. □

Notice that the d objects get into the membranes with label b the same time when the t_1 and f_1 objects get into the mentioned membranes with b label.

4.3 Creation of the quantifier tree

We introduced the membranes with δ label having positive polarity in the initial membrane structure. These membranes delay the non-elementary membrane divisions, so instead of forming chains, we are going to generate a tree structure as the new variable interpretations are created. We add the

$$[[\quad]^+_{C_{i_1,j_1,k_1}}[\quad]^-_{C_{i_1,j_1,k_1}}]^+_\delta \to [[\quad]^0_{C_{i_1,j_1,k_1}}]^0_\delta[[\quad]^0_{C_{i_1,j_1,k_1}}]^0_\delta \tag{13}$$

$$[[\quad]^+_{\varepsilon_{t_i}}[\quad]^-_{\varepsilon_{t_i}}]^+_\delta \to [[\quad]^0_{\varepsilon_{t_i}}]^0_\delta[[\quad]^0_{\varepsilon_{t_i}}]^0_\delta \quad (i = 2, 4, \dots, n) \tag{14}$$

$$[[\quad]^+_\delta[\quad]^-_\delta]^+_\delta \to [[\quad]^0_\delta]^0_\delta[[\quad]^0_\delta]^0_\delta \tag{15}$$

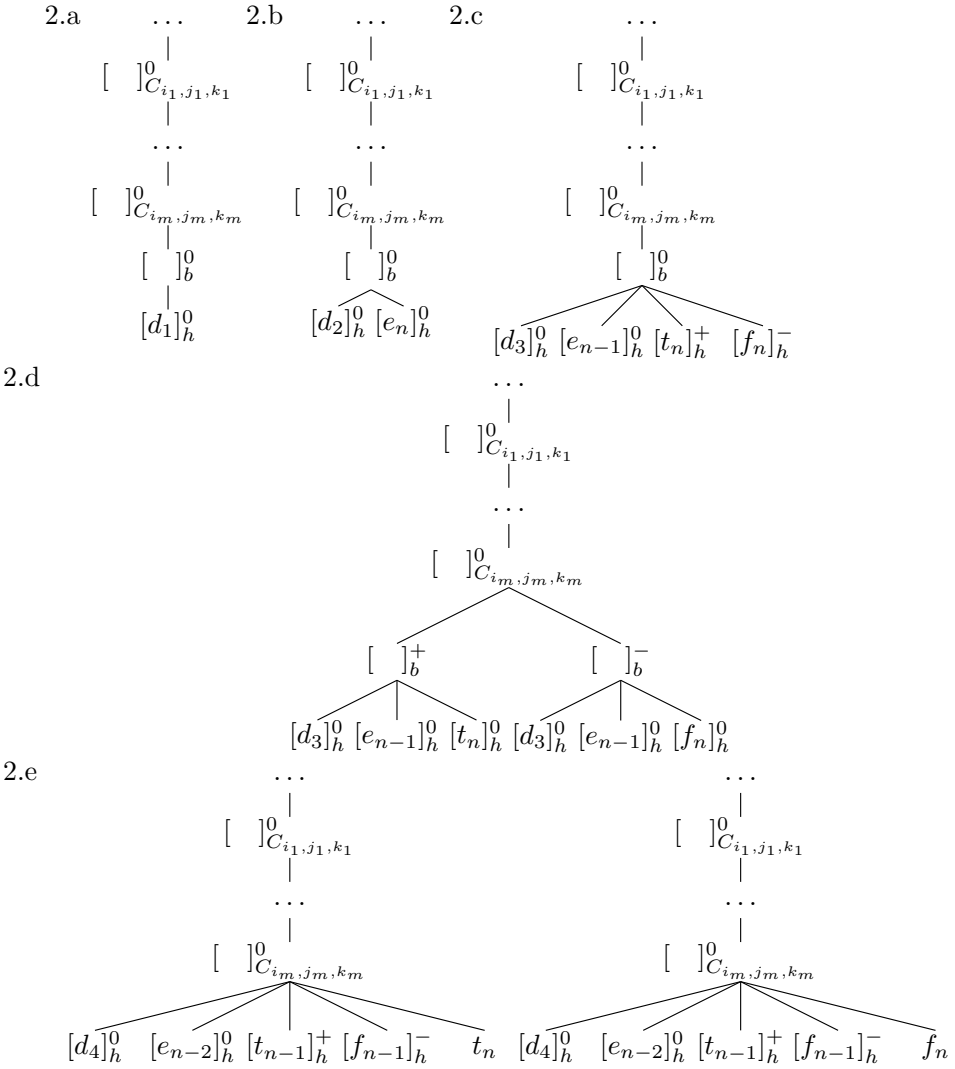


Fig. 2. The beginning of the creation of the variable interpretations. The bottom of the initial membrane structure can be seen on figure (a). The (b) figure shows the structure after one step. We applied the $[d_1]_h^0 \to [d_2]_h^0[e_n]_h^0$ rule. The (c) figure shows the beginning of the second step. Here, the $[d_2]_h^0 \to [d_3]_h^0[e_{n-1}]_h^0$ rule and the $[e_n]_h^0 \to [t_n]_h^+[f_n]_h^-$ rule were applied. The polarity difference induces a non-elementary membrane division, which induces another non-elementary membrane division on the next level, etc. The state after the first division can be seen on figure (d). On figure (e), the non-elementary divisions are finished. The upper membranes are affected by the divisions too, but we are going to describe that in section 4.3. This figure showes the state after the application of the $[d_3]_h^0 \to [d_4]_h^0[e_{n-2}]_h^0$, $[e_{n-1}]_h^0 \to [t_{n-1}]_h^+[f_{n-1}]_h^-$ and $[t_n]_h^0 \to t_n$, $[f_n]_h^0 \to f_n$ rules. One can see how the recursion goes after comparing this figure with figure (c).

rules to the system. These rules change the initial positive polarity of a membrane with label δ to neutral polarity. In this case, we will say that the δ membrane activates. The activation of a δ labeled membrane pair can be seen on figure 3. The activated membrane with label δ will propagate the polarity difference just as the other membranes do with the

$$[[\quad]^{+}_{C_{i_1,j_1,k_1}}[\quad]^{-}_{C_{i_1,j_1,k_1}}]^{0}_{\delta} \to [[\quad]^{0}_{C_{i_1,j_1,k_1}}]^{+}_{\delta}[[\quad]^{0}_{C_{i_1,j_1,k_1}}]^{-}_{\delta} \tag{16}$$

$$[[\quad]^{+}_{\varepsilon_{t_i}}[\quad]^{-}_{\varepsilon_{t_i}}]^{0}_{\delta} \to [[\quad]^{0}_{\varepsilon_{t_i}}]^{+}_{\delta}[[\quad]^{0}_{\varepsilon_{t_i}}]^{-}_{\delta} \quad (i = 2, 4, \ldots, n) \tag{17}$$

$$[[\quad]^{+}_{\delta}[\quad]^{-}_{\delta}]^{0}_{\delta} \to [[\quad]^{0}_{\delta}]^{+}_{\delta}[[\quad]^{0}_{\delta}]^{-}_{\delta} \tag{18}$$

rules, as it can be seen in figure 4. The ε membranes split by using the

$$[[\quad]^{+}_{\delta}[\quad]^{-}_{\delta}]^{0}_{\varepsilon_{f_i}} \to [[\quad]^{0}_{\delta}]^{+}_{\varepsilon_{f_i}}[[\quad]^{0}_{\delta}]^{-}_{\varepsilon_{f_i}} \tag{19}$$

$$[[\quad]^{+}_{\varepsilon_{f_i}}[\quad]^{-}_{\varepsilon_{f_i}}]^{0}_{\varepsilon_{t_i}} \to [[\quad]^{0}_{\varepsilon_{f_i}}]^{+}_{\varepsilon_{t_i}}[[\quad]^{0}_{\varepsilon_{f_i}}]^{-}_{\varepsilon_{t_i}} \tag{20}$$

rules $(i = 2, 4, \ldots, n)$, so they propagate the polarity difference upwards. Notice that both the activated membranes with δ label and the membranes with label ε propagate the polarity difference upwards, while the not activated δ membranes halt this propagation.

Now we are going to show that when a new variable interpretation is created at the bottom of the tree structure (so the $[t_i]^{+}_{h}$ and $[f_i]^{-}_{h}$ membranes are introduced), then at the beginning of the next step, after the (5)-(6) rules are applied, the interpretations in the leaves are ordered. By ordered, we mean that when examining the f_i and t_i objects in the leaves of the tree structure from right to left, there is only f_i objects in the rightmost leaf, the object representing the nth variable negate (f_n changes to t_n and t_n changes to f_n) from leaf to leaf, and the object representing the ith variable negate when the object representing the $(i+1)$th variable changes from t_{i+1} to f_{i+1}. For example, the

$$t_1t_2t_3 \quad t_1t_2f_3 \quad t_1f_2t_3 \quad t_1f_2f_3 \quad f_1t_2t_3 \quad f_1t_2f_3 \quad f_1f_2t_3 \quad f_1f_2f_3 \tag{21}$$

sequence of objects in the leaves form an ordered sequence.

Lemma 2. *When we apply the (4) rule for the kth time, in the same step, the kth membrane with label δ from the bottom of the membrane structure activates.*

Proof. We are going to give a proof by induction. Initially, every membrane with label δ have positive polarity, so they are not activated.

- When the $[e_n]^{0}_{h} \to [t_n]^{+}_{h}[f_n]^{-}_{h}$ rule is applied, the lowest membrane with label δ activates. According the (13) rule, this membrane will not propagate the polarity difference.
- In the general case, when the (4) rule is applied for the $(k+1)$th time, the kth membrane with label δ is already activated according the induction, so it will propagate this polarity difference. We have two case here.

- When the $(k+1)$th membrane with label δ is the upper neighbor of the kth membrane with label δ, then the former membrane activates according the (15) rule.
- When there are ε labeled membranes between the $(k+1)$th and the kth membranes labeled δ, then according the (19) and (20) rules, the polarity difference is propagated to the $(k+1)$th membrane with label δ from the kth, so it activates.

□

Beside from the activation, one can see the process how given membranes permute their lower neighbors on figures 3, 4 and 5. By permutation we mean what we give in the descriptions of the mentioned figures: some membranes go to the opposite subtree from their original subtree. We are going to show exactly which membranes permute their lower membranes.

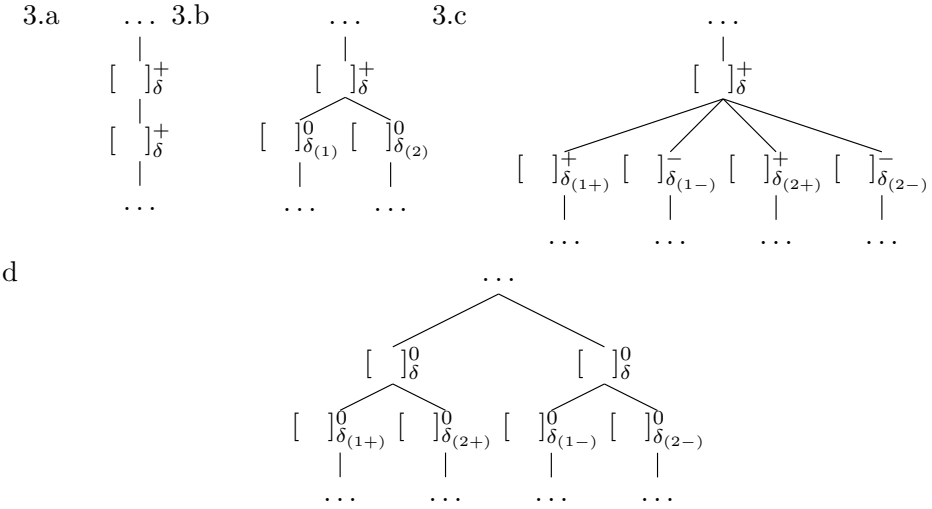


Fig. 3. On the (a) figure, one can see the initial state of two δ labeled membranes. A polarity difference on the lower level splits the bottom δ labeled membrane as one can see in figure (b). Another polarity difference in the bottom membranes splits the activated δ labeled membranes in figure (c). Temporarily we denoted the number and the polarity of the given membranes, so one can trace them. On figure (d) one can see that the membranes with positive polarity go into the left subtree, while the membranes with the negative polarity go into the right subtree. This only influences the $\delta_{(1-)}$ and $\delta_{(2+)}$ membranes: they go to the opposite subtree from their original place.

Lemma 3. *Only the* $(2i+1)$*th membranes with label* δ $(i > 1)$ *and the membranes with label* ε_{f_i} $(i = 2, 4, \ldots, n)$ *perform permutation on non-elementary membrane division.*

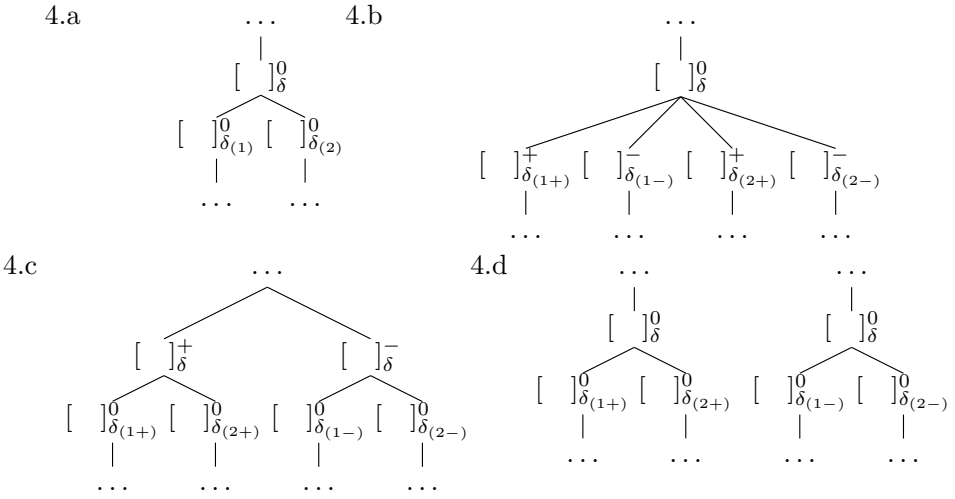


Fig. 4. On the (a) figure, we can see an activated δ labeled formation. We denote the number and the polarity again for better traceability. Only one polarity difference in each bottom δ labeled membrane can cause the whole structure to split in two, see figures (b), (c) and (d). Notice, that the $\delta_{(1-)}$ and $\delta_{(2+)}$ membranes go to the opposite subtree from their original subtree. Furthermore, notice that we arrived at a state where we duplicated the state on figure (a).

Proof. Notice that permutation can occur only in the membranes where more than two membranes are present with polarity difference. This can only happen in the mentioned membranes. These membranes perform permutations as it can be seen in figures 3, 4 and 5. □

Lemma 4. *When we apply the (4) rule for the kth time, in the same step, only the membranes not higher in the membrane structure than the kth membrane with label δ perform permutation.*

Proof. The yet not activated membranes with δ label halt the propagation of the polarity difference, so we should examine the membrane structure from the bottom only until the last activated δ labeled membrane. Applying the (4) rule for the kth time results in the activation of the kth δ labeled membrane according lemma 2, so the membranes lower than this membrane perform their permutation. According to this, we only have to deal with the actually activated membrane with δ label and according lemma 3, we can concentrate on the $(2i + 1)$th membranes with label δ ($i > 1$). But examining figure 3, one can see that these membranes perform their permutation on activation. □

Lemma 5. *Applying the (4) rule in the tree structure when there is an ordered sequence of interpretations in the leaves, at the beginning of the next step, after the (5)-(6) rules are applied, the interpretations in the leaves will be ordered again.*

Proof. When applying the (4) rule for the kth time, according lemma 2 the kth membrane with δ label activates and according lemma 4 only the membranes not higher in the membrane structure than the kth membrane with label δ perform permutation. According this, we are going to give a proof by induction on the number of activated membranes with δ label.

- No permutation happens when the first membrane with δ label activates. At the beginning of the next step, after the (5)-(6) rules are applied, there is two leaves, having t_n in the left leaf and f_n in the right leaf, so the ordering property holds.
- In general either the $2k$th or the $(2k+1)$th $(k > 0)$ membrane with δ label activates. For the former case see figure (6.a) and (6.b), for the later case see figure (6.c) and (6.d).

□

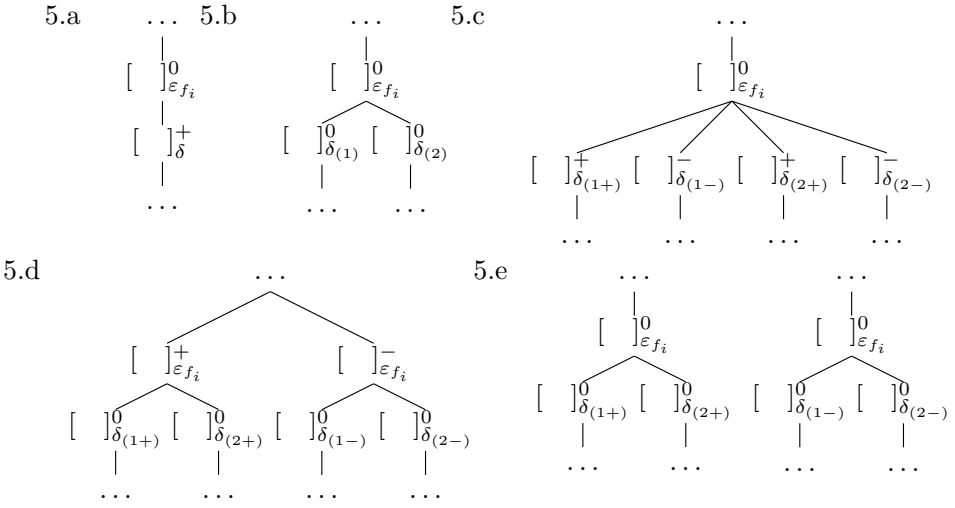


Fig. 5. This figure shows how a membrane with ε_{f_i} permutes the lower neighbors. The membrane with label δ activates in figure (a) and (b). Another polarity difference in the lower neighbors split the membrane with ε_{f_i} label. Notice that the $\delta_{(1-)}$ and $\delta_{(2+)}$ membranes go to the opposite subtree from their original subtree.

Using the result of lemma 1 and lemma 5, we know that after the application of the $[e_1]_h^0 \to [t_1]_h^+[f_1]_h^-$ rule, at the beginning of the next step, after the (5)-(6) rules are applied, we will have all the possible interpretations in an ordered sequence in the leaves of the membrane structure.

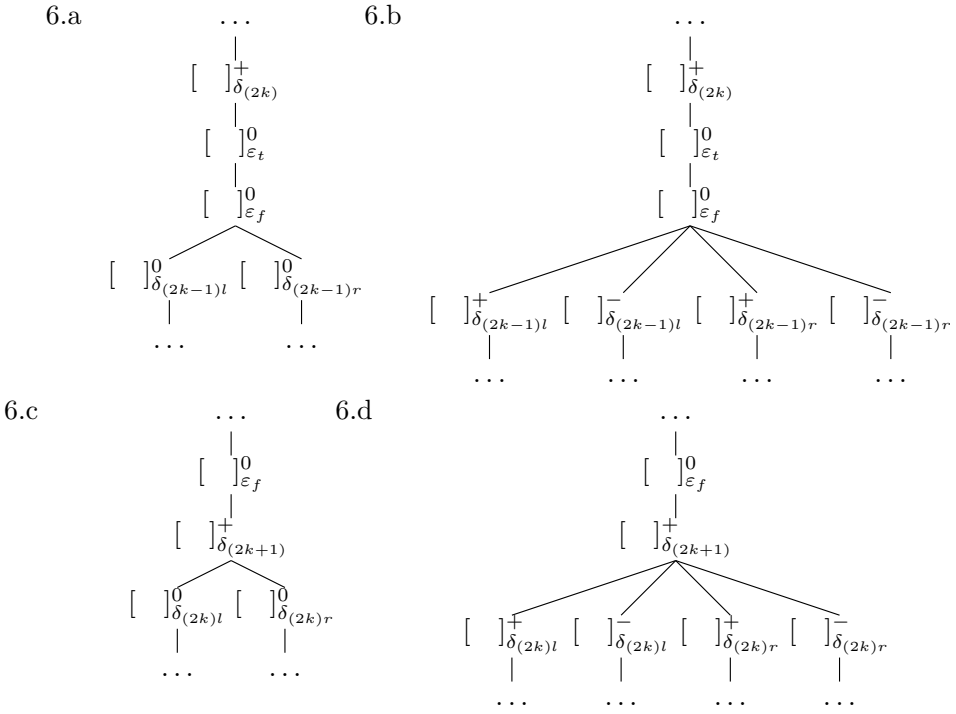


Fig. 6. These figures serve as a part of the proof of lemma 5. We omitted the indexes from the ε_t and ε_f labels for simplicity. In figure (a) we can see the state when the $2k$th membrane with δ label is the next one to be activated, and we have not applied the (4) rule yet. In this case, the interpretations in the leaves of the membrane structure form an ordered sequence. The first half of the ordered sequence is in the subtree under the $\delta_{(2k-1)l}$ labeled membrane, the second half of the ordered sequence is in the subtree under the $\delta_{(2k-1)r}$ labeled membrane. In figure (b), after the application of the (4) rule when the membranes with $\delta_{(2k-1)l}$ and $\delta_{(2k-1)r}$ labels split, according the induction the positively charged membrane with $\delta_{(2k-1)l}$ label contains the first half of the interpretations from the sequence, each one concatenated with $[t_{n+1-2k}]_h^0$ and the negatively charged membrane with $\delta_{(2k-1)l}$ label contains the first half of the interpretations from the sequence, each one concatenated with $[f_{n+1-2k}]_h^0$. The same holds for the membranes with $\delta_{(2k-1)r}$ labels, just with the second half of the sequence. The permutation performed by the membrane with ε_f label exchanges the negatively charged $\delta_{(2k-1)l}$ labeled membrane with the positively charged $\delta_{(2k-1)r}$ labeled membrane. No more permutations are performed after this one in this step. So in the beginning of the next step, after the (5)-(6) rules are applied, the interpretations in the leaves will form an ordered sequence. In figure (c) we can see the state when the $(2k+1)$th membrane with δ label is the next one to be activated, and we have not applied the (4) rule yet. Here we can follow the same reasoning as in the previous case.

4.4 Evaluation

The evaluation stage starts when the d objects get into the b labeled membranes. As we have mentioned, this happens the same time when the t_1 and f_1 enter the b labeled membranes. The evaluation initiates with the use of the

$$[d]_b^0 \to d \tag{22}$$

rule, which sends every object form a b labeled membrane to the upper neighbor. For evaluating the clauses we introduce the

$$[t_p]_{C_{i,j,k}}^0 \to t_p \ \textit{if}\ p \in \{i,j,k\} \tag{23}$$

$$[f_p]_{C_{i,j,k}}^0 \to f_p \ \textit{if}\ p \in \{-i,-j,-k\} \tag{24}$$

rules where $p = 1, 2, \ldots, n$ and $i, j, k \in \{1, 2, \ldots, n\} \cup \{-1, -2, \ldots, -n\}$ satisfying $|i| < |j| < |k|$. So for example if our clause membrane is labeled with $C_{1,-2,3}$, then

$$[t_1]_{C_{1,-2,3}}^0 \to t_1$$
$$[f_2]_{C_{1,-2,3}}^0 \to f_2$$
$$[t_3]_{C_{1,-2,3}}^0 \to t_3$$

will be the rules for this membrane. The upper bound on the number of the possible clauses is $8\binom{n}{3}$, and for every clause we introduce 3 rules, so an upper bound on the number or rules introduced with this reasoning is $24\binom{n}{3}$ which is still polynomial.

An interpretation dissolves a clause membrane if one of the truth values (represented by objects) in it evaluates the given clause to a true truth value. The interpretations propagate upward, and they only get into the quantifier tree if they satisfy the formula. For the ε labeled membranes, we introduce the

$$[t_i]_{\varepsilon_{t_i}}^0 \to t_i \tag{25}$$

$$[f_i]_{\varepsilon_{f_i}}^0 \to f_i \tag{26}$$

rules for $i = 2, 4, \ldots, n$ and for the membranes with δ labels, the

$$[t_i]_\delta^0 \to t_i \tag{27}$$

$$[f_i]_\delta^0 \to f_i \tag{28}$$

rules where $i = 1, \ldots, n$.

Lemma 6. *Membranes with ε_{t_i} and ε_{f_i} label dissolve during the evaluation stage if and only if $\exists x_1 \dots \exists x_{i-1} \forall x_i \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true. (Here, the variables $x_1, \dots, x_{i-2}$ are existentially quantified, then existentially and universally quantified variables come alternately.)*

Proof. We are going to give a proof by induction.

$\Rightarrow$ Here we assume that the membranes with ε_{t_i} and ε_{f_i} label dissolve during the evaluation stage.

- Let $i = n$. In the membrane labeled ε_{f_n}, there is $x_1 \dots x_{n-1} t_n$ in the leaf on the left side of the branch and $x_1 \dots x_{n-1} f_n$ in the leaf on the right side of the branch. (Here x_p is either t_p or f_p.) Because the membranes with ε_{t_n} and ε_{f_n} dissolve, we know that the interpretations passed through the clause-chain which means that the interpretations satisfy ϕ. From this, we get that $\exists x_1 \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true.
- Let $i = n - 2$. Because the membranes with $\varepsilon_{t_{n-2}}$ and $\varepsilon_{f_{n-2}}$ dissolve, we know that $\exists x_1 \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true (because one have to dissolve membranes with ε_{t_n} and ε_{f_n} labels to achieve this) with t_{n-2} and some x_{n-1}, and it is also true with f_{n-2} and some (probably different) x_{n-1}, which means that $\exists x_1 \dots \exists x_{n-3} \forall x_{n-2} \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true.
- In the general case, lets assume that the statement is true for the membranes with ε_{t_i} and ε_{f_i} labels. Because of this and the induction, we know that $\exists x_1 \dots \exists x_{i+1} \forall x_{i+2} \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true with t_i and some x_{i+1}, and it is also true with f_i and some (probably different) x_{i+1}, which means that $\exists x_1 \dots \exists x_{i-1} \forall x_i \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true.

$\Leftarrow$ Here we assume that $\exists x_1 \dots \exists x_{i-1} \forall x_i \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true.

- Lets assume that $\exists x_1 \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true. This means that two clause-chains under the same membrane with ε_{f_n} label dissolve, because the interpretations in the leaves satisfy ϕ. Both interpretations get into ε_{f_n} which dissolves in the presence of f_n. After this, ε_{t_n} dissolves because of t_n.
- Now lets assume that $\exists x_1 \dots \exists x_{n-3} \forall x_{n-2} \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true. This means, that $\exists x_1 \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true with t_{n-2} and some x_{n-1}, and it is also true with f_{n-2} and some (probably different) x_{n-1}. Because of the structure of the quantifier tree, this means that there exists a membrane pair with $\varepsilon_{f_{n-2}}$ and $\varepsilon_{t_{n-2}}$ labels in the tree which dissolves.
- Lets assume that $\exists x_1 \dots \exists x_{i-1} \forall x_i \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true. This means that $\exists x_1 \dots \exists x_{i+1} \forall x_{i+2} \dots \exists x_{n-1} \forall x_n \phi(x_1, \dots, x_n)$ is true with t_i and some x_{i+1} and it is true with f_i and some (probably different) x_{i+1}. Because of this, plus the induction and the structure of the quantifier tree, we get that there exists a membrane pair with ε_{f_i} and ε_{t_i} labels in the tree which dissolves.

□

Lemma 7. *The $\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n \phi(x_1, \ldots, x_n)$ formula (where n is even) is true (resp. false) if and only if at least one d object (resp. no object) arrives to the skin membrane from the quantifier tree part of the membrane structure.*

Proof. We are going to use the results of lemma 6.

$\Rightarrow$ If the formula is true, then a membrane pair with ε_{f_2} and ε_{t_2} labels dissolves, so d objects get into the skin membrane. If the formula is false, then no objects get into the skin membrane, because no membrane pair with ε_{f_2} and ε_{t_2} labels dissolve.

$\Leftarrow$ If d objects get into the skin membrane, then at least one membrane pair with ε_{f_2} and ε_{t_2} labels dissolved, which means that the formula is true. If no objects enter the skin membrane, then no membrane pair with ε_{f_2} and ε_{t_2} labels dissolved, so the formula is false.

□

If the d objects of at least one interpretation get to the skin membrane, then the formula is satisfiable and we stop. Otherwise, all of the interpretations halt somewhere and the n object gets into the skin using the

$$[n]_c^0 \to n \tag{29}$$

rule. We chose the length of the chain formed by the c membranes to be polynomial and to be longer than $n+2$. This way, if the formula is satisfiable, then the n object would get into the skin later than any other d object. Otherwise, it indicates the unsatisfiability.

Examining the given rules in this section, one can see that the number of objects and the number of rules in the system is polynomially bounded. Together with the polynomial bound on the size of the initial membrane structure, we get that the given solution is polynomially uniform.

5 Conclusions

We have shown that recognizer P systems with active membranes and no input membrane, having three polarizations using only dissolution and division rules are able to solve the Q3SAT decision problem in the restricted case when the quantifiers alternate, which problem - even with the restriction - is **PSPACE**-complete. The presented solution is polynomially uniform.

6 Acknowledgments

I would like to thank Alberto Leporati, his colleagues, furthermore Erzsébet Csuhaj-Varjú and Zsolt Gazdag for their helpful suggestions.

References

1. Leporati, A., Feretti, C., Mauri, G., Pérez-Jiménez, M. J., Zandron, C.: Complexity Aspects of Polarizationless Membrane Systems. Natural Computing 8(4), 703–717 (2009)
2. Păun, G.: Membrane computing: An introduction. Springer-Verlag (2002)
3. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
4. Porreca, A. E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes: Trading time for space. Natural Computing 10(1), 167–182 (2011)
5. Porreca, A. E., Leporati, A., Mauri, G., Zandron, C.: Recent complexity-theoretic results on P systems with active membranes. Journal of Logic and Computation (2013)

Minimal cooperation in polarizationless P systems with active membranes

Luis Valencia-Cabrera, David Orellana-Martín,
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, ariscosn, marper}@us.es

Summary. P systems with active membranes is a well developed framework in the field of Membrane Computing. Using evolution, communication, dissolution and division rules, we know that some kinds of problems can be solved by those systems, but taking into account which ingredients are used. All these rules are inspired by the behavior of living cells, who "compute" with their proteins in order to obtain energy, create components, send information to other cells, kill themselves (in a process called *apoptosis*), and so on.

But there are other behaviors not captured in this framework. As *mitosis* is simulated by *division* rules (for elementary and non-elementary membranes), *meiosis*, that is, membrane fission inspiration is captured in *separation* rules. It differs from the first in the sense of duplication of the objects (that is, in *division* rules, we duplicate the objects not involved in the rule, meanwhile in *separation* rules we divide the content of the original membrane into the new membranes created).

Evolution rules simulate the transformation of components in membranes, but it is well known that elements interact with another ones in order to obtain new components. Cooperation in evolution rules is considered. More specifically, minimal cooperation (in the sense that only two objects can interact in order to create one or two objects).

Key words: Membrane Computing, Active membranes, Minimal cooperation, Mitosis, Computational Complexity, The **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, both the chemical substance and the membrane can be modified, at least locally.

Mitosis is a process by which two or more cells are produced/generated from one cell that could be considered as the "mother". Several cell division inspired mechanisms were introduced in Membrane Computing. Specifically, *P systems with active membranes* [9] incorporates the mitosis based mechanisms by means of *membrane division* rules. By applying this kind of rules, under the influence of the object triggering it, the membrane is divided into two membranes and that object is replaced in the two new ones by possibly new objects, while the remaining objects are *duplicated* in both newly created membranes. These models are universal (they are equivalent in power to deterministic Turing machines) and they have the ability to provide efficient solutions to computationally hard problems, by making use of an exponential workspace created in a polynomial time (often, in linear time). Moreover, **PSPACE**-complete problems can be efficiently solved by families of P systems with active membranes which use division for elementary and non-elementary membranes. This paper deals with P systems with active membranes where electrical charges are removed.

The paper is organized as follows. Next section briefly describes some preliminaries in order to make the work self-contained. In Section 3, syntax and semantics of polarizationless P systems with active membranes by using membrane division rules or membrane separation rules are introduced, and minimal cooperation in object evolution rules is considered. Definition of *Recognizer membrane systems* is recalled in Section 4, as a framework to provide efficient solutions to decision problems. The computational efficiency of polarizationless P systems with active membranes, division rules, minimal cooperation and without dissolution rules is established in Section 5 by providing a uniform polynomial-time solution to `SAT` problem. A formal verification of this result is presented in Section 6. Next section is dedicated to show the limits of the computational efficiency of the polarizationless P systems with active membranes, separation rules and minimal cooperation in object evolution rules. 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 $n \in \mathbb{N}$ onto Γ. The number n is called the *length* of the string u and it is denoted by $|u|$, that is, the length of a string is the number of occurrences of symbols that it contains. 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 and we denote by $M_f(\Gamma)$ the set of all finite multisets over Γ.

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.

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. A node with no children is called a *leaf* (see [3] for details).

3 Polarizationless P Systems with Active Membranes

Let us briefly recall some definitions of P systems models that will be used in the paper (see [12] for details).

A *basic transition* P system is a membrane system whose rules are of the following forms: evolution, communication, and dissolution. In these systems the size of the membrane structure does not increase, but an exponential workspace (in terms of number of objects) can be constructed in linear time, e.g. via evolution rules of the type $[\, a \to a^2 \,]_h$. Nevertheless, such capability is not enough to efficiently solve **NP**–complete problems, unless **P** = **NP** (see [6] for details).

Replication is one of the most important functions of a cell and, in ideal circumstances, a cell produces two identical copies by division. Bearing in mind that the reactions which take place in a cell are related to membranes, division rules for elementary and non-elementary membranes are considered in the so-called *P systems with active membranes*. Such variant was first introduced by Gh. Păun [10] and it has associated electrical charges with membranes but the rules are non-cooperative and there are not priorities. Nevertheless, the class of all problems solvable in polynomial time and in a uniform way by means of families of P systems with active membranes which use division for elementary and non-elementary membranes contains class **PSPACE** and it is contained in class

EXP [16]. Thus, in order to provide efficient solutions to computationally hard problems, this framework seems to be too powerful from the computational complexity point of view.

In this paper, electrical charges are removed from P systems with active membranes. Two different ways of producing an exponential number of membranes in linear time will be considered: division and separation rules (abstractions of mitosis and membrane fission processes, respectively).

3.1 Polarizationless P system with active membranes: Syntax

Definition 1. *A polarizationless P system with active membranes and membrane division of degree $q \geq 2$ is a tuple $\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$, where*

- *Γ is a finite alphabet whose elements are called objects;*
- *H is a finite alphabet such that $H \cap \Gamma = \emptyset$ whose elements are called labels;*
- *μ is a labelled rooted tree (called membrane structure) consisting of q nodes injectively labeled by elements of H;*
- *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are finite multisets over Γ;*
- *$\mathcal{R}$ is a finite set of rules, of the following forms:*

  (a_0) $[\, a \to u \,]_h$ *for $h \in H$, $a \in \Gamma$, $u \in M_f(\Gamma)$* (object evolution rules).

  (b_0) $a\,[\ \ ]_h \to [\, b \,]_h$ *for $h \in H$, $a, b \in \Gamma$ and h is not the label of the root of μ* (send–in communication rules).

  (c_0) $[\, a \,]_h \to b\,[\ \ ]_h$ *for $h \in H$, $a, b \in \Gamma$* (send–out communication rules).

  (d_0) $[\, a \,]_h \to b$ *for $h \in H \setminus \{i_{out}\}$, $a, b \in \Gamma$ and h is not the label of the root of μ* (dissolution rules).

  (e_0) $[\, a \,]_h \to [\, b \,]_h\, [\, c \,]_h$ *for $h \in H \setminus \{i_{out}\}$, $a, b, c \in \Gamma$ and h is not the label of the root of μ* (division rules for elementary membranes).

  (f_0) $[\,[\ ]_{h_0}[\ ]_{h_1}\,]_h \to [\,[\ ]_{h_0}\,]_h\, [\,[\ ]_{h_1}\,]_h$, *where $h \in H \setminus \{i_{out}\}$ is not the label of the root of μ and $h_0, h_1 \in H$* (division rules for non–elementary membranes).
- *$i_{out} \in H \cup \{env\}$, where $env \notin H$ and in the case $i_{out} \in H$, i_{out} is the label of a leaf of μ.*

Definition 2. *A polarizationless P system with active membranes and membrane separation of degree $q \geq 2$ is a tuple $\Pi = (\Gamma, \Gamma_0, \Gamma_1, H, H_0, H_1, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$, where*

- *Γ is a finite alphabet whose elements are called objects;*
- *H is a finite alphabet such that $H \cap \Gamma = \emptyset$ whose elements are called labels;*
- *$\{\Gamma_0, \Gamma_1\}$ is a partition of Γ and $\{H_0, H_1\}$ is a partition of H;*
- *μ is a labelled rooted tree (called membrane structure) consisting of q nodes injectively labeled by elements of H;*
- *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are finite multisets over Γ;*
- *$\mathcal{R}$ is a finite set of rules, of the following forms:*

  (a_0) $[\, a \to u \,]_h$ *for $h \in H$, $a \in \Gamma$, $u \in M_f(\Gamma)$* (object evolution rules).

(b_0) $a[\ ]_h \to [b]_h$ *for* $h \in H$, $a, b \in \Gamma$ *and* h *is not the label of the root of* μ (send–in communication rules).

(c_0) $[a]_h \to b[\ ]_h$ *for* $h \in H$, $a, b \in \Gamma$ (send–out communication rules).

(d_0) $[a]_h \to b$ *for* $h \in H \setminus \{i_{out}\}$, $a, b \in \Gamma$ *and* h *is not the label of the root of* μ (dissolution rules).

(e_0) $[a]_h \to [\Gamma_0]_h [\Gamma_1]_h$ *for* $h \in H \setminus \{i_{out}\}$, $a \in \Gamma$ *and* h *is not the label of the root of* μ (separation rules for elementary membranes).

(f_0) $[[\ ]_{h_0}[\ ]_{h_1}]_h \to [\Gamma_0[\ ]_{h_0}]_h\ [\Gamma_1[\ ]_{h_1}]_h$, *where* $h \in H \setminus \{i_{out}\}$ *is not the label of the root of* μ, $h_0 \in H_0$ *and* $h_1 \in H_1$ (separation rules for non–elementary membranes).

- $i_{out} \in H \cup \{env\}$, *where* $env \notin H$ *and in the case* $i_{out} \in H$, i_{out} *is the label of a leaf of* μ.

A polarizationless P system with active membranes of degree $q \geq 2$, can be viewed as a set of q membranes, labelled by elements of H, arranged in a hierarchical structure μ given by a rooted tree 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) $\mathcal{R}$ is a finite set of rules over Γ associated with the labels; and (c) $i_{out} \in H \cup \{env\}$ indicates the output region. We use the term *region* i to refer to membrane i in the case $i \in H$ and to refer to the "environment" of the system in the case $i = env$. The leaves of μ are called elementary membranes, otherwise, the membrane is said to be non-elementary.

3.2 Polarizationless P system with active membranes: Semantics

An *instantaneous description* or a *configuration* $\mathcal{C}_t$ at an instant t of a polarizationless P system with active membranes is described by the following elements: (a) the membrane structure at instant t, and (b) all multisets of objects over Γ associated with all the membranes present in the system at that moment.

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

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

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

A dissolution rule $[a]_h \to b$ for $h \in H \setminus \{i_{out}\}$, $a, b \in \Gamma$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in

$\mathcal{C}_t$, different from the skin membrane and the output region, such that it contains object a. When applying such a rule, object a is consumed, membrane h is dissolved and its objects are sent to the parent (or the first ancestor that has not been dissolved).

A division rule $[\,a\,]_h \rightarrow [\,b\,]_h[\,c\,]_h$ for $h \in H \setminus \{i_{out}\}, a, b, c \in \Gamma$, is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists an elementary membrane labelled by h in $\mathcal{C}_t$, different from the skin membrane and the output region, such that it contains object a. When applying a division rule $[a]_h \rightarrow [\,b\,]_h\,[\,c\,]_h$ to a membrane labelled by h in a configuration $\mathcal{C}_t$, under the influence of object a, the membrane with label h 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 are replicated and copies of them are placed in the two new membranes.

A division rule $[\,[\,]_{h_0}[\,]_{h_1}\,]_h \rightarrow [\,[\,]_{h_0}\,]_h\ [\,[\,]_{h_1}\,]_h$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$, different from the skin membrane and the output region, which contains a membrane labelled by h_0 and another membrane labelled by h_1. When applying such a division rule to a membrane labelled by h in a configuration $\mathcal{C}_t$, the membrane with label h is divided into two membranes with the same label; the first copy inherits membrane h_0 with its contents, and the second copy inherits membrane h_1 with its contents. Besides, if the membrane labelled by h contains more membranes other than those with the labels h_0, h_1, then such membranes are duplicated so that they become part of the contents of both new copies of the membrane h.

A separation rule $[\,a\,]_h \rightarrow [\ \Gamma_0\ ]_h\ [\ \Gamma_1\ ]_h$ for $h \in H$, $a \in \Gamma$, is applicable to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$, different from the skin membrane and the output region, such that it contains object a. When applying such a rule, the membrane is separated into two membranes with the same label; at the same time, object a is consumed and the multiset of objects contained in membrane h gets distributed: the objects from Γ_0 are placed in the first membrane, those from Γ_1 are placed in the second membrane.

A separation rule $[\,[\,]_{h_0}[\,]_{h_1}\,]_h \rightarrow [\ \Gamma_0\ [\,]_{h_0}\,]_h\ [\ \Gamma_1\ [\,]_{h_1}\,]_h$, where h, h_0, h_1 are labels such that $h_0 \in H_0$ and $h_1 \in H_1$, is applicable to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$, different from the skin membrane and the output region, such that it contains a membrane labelled by h_0 and another membrane labelled by h_1. When applying such a separation rule to a membrane labelled by h in a configuration $\mathcal{C}_t$, that membrane is separated into two membranes with the same label, in such a way that the contents (multiset of objects and inner membranes) are distributed as follows: The first membrane receives the multiset of objects from Γ_0, and all inner membranes whose label belongs to H_0; and the second membrane receives the multiset of objects from Γ_1, and all inner membranes whose label belongs to H_1.

In polarizationless P systems with active membranes, 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 membrane* can be the subject of *only one* rule of types (b_0)–(f_0), and then it is applied at most once.
- Object evolution rules can be simultaneously applied to a membrane with one rule of types (b_0)–(f_0). Object evolution rules are applied in a maximally parallel manner.
- If at the same time a membrane labelled with h is divided by a rule of type (e_0) or (f_0) and there are objects in this membrane which evolve by means of rules of type (a_0), then we suppose that first the evolution rules of type (a_0) are used, changing the objects, and then the division (or the separation) is produced. Of course, this process takes only one transition step.
- The skin membrane and the output membrane can never get divided, separated, nor dissolved.

3.3 Polarizationless P systems with active membranes and minimal cooperation in object evolution rules

Next, we incorporate cooperation in object evolution rules of polarizationless P systems with active membranes. In this paper, we use minimal cooperation in the following sense: the left-hand side of each object evolution rules has at most two objects, and the length of the right-hand side cannot be greater than the length of the left-hand side. Consequently, in contrast with the usual object evolution rules in P systems with active membranes, by applying these rules with minimal cooperation the number of objects of the system does not increase.

Definition 3. *A polarizationless P system with active membranes, division or separation rules and minimal cooperation in object evolution rules is a polarizationless P system with active membranes and division or separation rules such that the object evolution rules are of the following form:*

$$[\, a \to c \,]_h,\ [\, a\, b \to c \,]_h,\ [\, a\, b \to c\, d \,]_h$$

for $h \in H$ and $a, b, c, d \in \Gamma$.

The semantics of these variants are analogous to the semantics of polarizationless P systems with active membranes.

4 Recognizer membrane systems

In what follows, a *membrane system* denotes a P system of any of the different variants considered in the previous section.

Definition 4. *We say that 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. *Σ is an (input) alphabet strictly contained in Γ.*
3. *The initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_q$ of Π are finite multisets over $\Gamma \setminus \Sigma$.*
4. *There exists a distinguished membrane labelled by i_{in} 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.*

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. 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.

We use the following notations:

- $\mathcal{DAM}^0(\gamma, \delta)$ where $\gamma \in \{-d, +d\}$ and $\delta \in \{-n, +n\}$, is the class of all recognizer polarizationless P systems with active membranes and division rules.
- $\mathcal{DAM}^0_{mc}(\gamma, \delta)$ where $\gamma \in \{-d, +d\}$ and $\delta \in \{-n, +n\}$, is the class of all recognizer polarizationless P systems with active membranes, minimal cooperation in object evolution rules and division rules.
- $\mathcal{SAM}^0(\gamma, \delta)$ where $\gamma \in \{-d, +d\}$ and $\delta \in \{-n, +n\}$, is the class of all recognizer polarizationless P systems with active membranes and separation rules.
- $\mathcal{SAM}^0_{mc}(\gamma, \delta)$ where $\gamma \in \{-d, +d\}$ and $\delta \in \{-n, +n\}$, is the class of all recognizer polarizationless P systems with active membranes, minimal cooperation in object evolution rules and separation rules.

The meaning of parameters γ and δ is the following:

- if $\gamma = +d$ then dissolution rules are permitted.
- if $\gamma = -d$ then dissolution rules are forbidden.
- if $\delta = +n$ then division rules for elementary and non–elementary membranes are permitted.
- if $\delta = -n$ then division rules only for elementary membranes are permitted.

Let us notice that standard notation in the literature referring to polarizationless P systems with active membranes ($\mathcal{AM}^0(\gamma, \delta)$) corresponds, within this new notation, to the class $\mathcal{DAM}^0(\gamma, \delta)$.

4.1 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 [13] for more details).

Definition 5. *Let $\mathcal{R}$ be a class of recognizer membrane systems. We say that a decision problem X is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of systems from $\mathcal{R}$, in a uniform way, denoted by $X \in \mathbf{PMC}_{\mathcal{R}}$, if the following 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.*

The polynomial complexity class $\mathbf{PMC}_{\mathcal{R}}$ is closed under polynomial-time reduction and under complement [14].

4.2 Known results on polarizationless P systems with active membranes

In previous works, membrane systems have been studied in terms of their computational efficiency and different borderlines between efficiency and non-efficiency have been obtained. Each of them provides attractive characterizations of the $\mathbf{P} \neq \mathbf{NP}$ conjecture.

In [5], by using the dependency graph technique and the tractability of the reachability problem, the following result has been proved.

Theorem 1. $\mathbf{P} = \mathbf{PMC}_{\mathcal{DAM}^0(-d,+n)}$

Thus, only problems in class $\mathbf{P}$ can be solved in polynomial time and in a uniform way by means of families of polarizationless P systems with active membranes making use of division rules for elementary and non-elementary membranes and not using dissolution rules.

In [2], a family of polarizationless P systems that make use of dissolution and division rules for elementary and non-elementary membranes solving the QSAT (*quantified satisfiability*) problem in polynomial time and in a uniform way was proposed.

Theorem 2. $\texttt{QSAT} \in \mathbf{PMC}_{\mathcal{DAM}^0(+d,+n)}$

Therefore, the following holds.

Corollary 1. PSPACE $\subseteq \mathbf{PMC}_{\mathcal{DAM}^0(+d,+n)}$

In [1], a family $\mathbf{\Pi}$ of P systems from $\mathcal{DAM}^0(+d,+n)$ solving SAT problem in polynomial time and in a *semi-uniform* way (each P system of the family is associated with only one instance of the problem) was proposed. Recall that SAT is one of the most well known **NP**-complete problems [4]. Next, based on the solution of QSAT problem provided in [2], a family of P systems from $\mathcal{DAM}^0(+d,+n)$ solving SAT problem in polynomial time and in a *uniform* way is presented.

Theorem 3. $\texttt{SAT} \in \mathbf{PMC}_{\mathcal{DAM}^0(+d,+n)}$

Proof. Let φ be a propositional formula in conjunctive normal form such that:

- $\varphi = C_1 \wedge \ldots \wedge C_p$
- $C_i = y_1 \vee \ldots \vee y_{l_i}$, for $1 \leq i \leq p, y_j \in \{x_k, \overline{x}_k \,|\, 1 \leq k \leq n\}$ being n the number of variables occurring in the formula.

We construct $\Pi = (\Gamma, \Sigma, H, \mu, \mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_{2n+p+3}, \mathcal{R}, i_{in}, i_{out})$ that will solve all instances of formulas with n variables and p clauses, provided that the appropriate input multiset $cod(\varphi) = \{v_{i,j} | x_i \in C_j\} \cup \{v'_{i,j} | \neg x_i \in C_j\}$ is supplied to the system (through the corresponding input membrane):

- $\Gamma = \Sigma \cup \{d_i | 1 \leq i \leq 7n+2p+2\} \cup \{f_i, t_i, a_i | 1 \leq i \leq n\} \cup \{c_i | 1 \leq i \leq p\} \cup \{u_{i,j}, u'_{i,j} | 1 \leq i \leq n, 1 \leq j \leq p\} \cup \{t', f', z, z', T, F, \texttt{yes}, \texttt{no}\}$
- $\Sigma = \{v_{i,j}, v'_{i,j} | 1 \leq i \leq n, 1 \leq j \leq p\}$
- $H = \{0, 1, \ldots, 2n+p+3\}$
- $[\,[\,[\,\ldots[\,[\,\,]_0\,]_1\,\,\ldots\,]_{2n+p+1}]_{2n+p+2}]_{2n+p+3}$
- $\mathcal{M}_0 = \mathcal{M}_{2n+p+2} = d_0, \mathcal{M}_i = \emptyset, i \notin \{0, 2n+p+2\}$
- $i_{in} = 0, i_{out} = env$

Rules are distributed as follow:

- **Generation Stage**

$$\left.\begin{array}{l}[d_{2i} \to a_{i+1}\ d_{2i+1}]_0 \\ [d_{2i+1} \to d_{2i+2}]_0\end{array}\right\} 1 \leq i < n$$
$$[a_i]_0 \to [t_i]_0 [f_i]_0, 1 \leq i \leq n$$
$$[[\,\,]_i[\,\,]_i]_{i+1} \to [[\,\,]_i]_{i+1} [[\,\,]_i]_{i+1}, 0 \leq i < 2n+p$$
$$[d_{2n+i} \to d_{2n+i+1}]_0, 0 \leq i \leq 2n+p$$
$$[d_{4n+p+1}]_0 \to T$$

$[d_i \to d_{i+1}]_{2n+p+2}, 0 \leq i \leq 7n + 2p + 1$

In $4n + p + 1$ steps, we expand the membrane structure in a tree-like fashion, preparing for the checking stage. First, we use $2n$ steps to generate 2^n copies of membrane 0, each one of them encoding a different truth assignment. Then, some more non-elementary divisions take place in the following $2n+p+1$ steps, in such a way that we get 2^n copies of a linear nested structure composed by membranes j, for $0 \leq j \leq 2n + p + 1$.
After $4n+p+1$ steps, all the contents of membranes labelled by 0 are released into their corresponding parent membranes (labelled by 1).

- **Assignments Stage**

$$\left.\begin{array}{l} [t_i \to t']_{2i-1} \\ [t']_{2i-1} \to z \\ [f_i]_{2i-1} \to f' \\ [f' \to z]_{2i} \\ [z]_{2i} \to z' \end{array}\right\} 1 \leq i \leq n$$

$$\left.\begin{array}{l} [v_{i,j} \to u_{i,j}]_{2i-1} \\ [v'_{i,j} \to u'_{i,j}]_{2i-1} \end{array}\right\} 1 \leq i \leq n, 1 \leq j \leq p$$

$$\left.\begin{array}{l} [u'_{i,j} \to \lambda]_{2i-1} \\ [u_{i,j} \to c_j]_{2i-1} \\ [u_{i,j} \to \lambda]_{2i} \\ [u'_{i,j} \to c_j]_{2i} \end{array}\right\} 1 \leq i \leq n, 1 \leq j \leq p$$

We have to see whether each truth assignment makes true φ or not. The formula φ has been satisfied if and only if objects c_i, with all $i \in \{1, \ldots, n\}$ have been created. After $3n$ steps, all membranes labelled by j with $0 \leq j \leq 2n$ have been dissolved, and their contents are gathered into membranes labelled by $2n + 1$.

- **Checking Stage**

$[c_i]_{2n+i} \to z', 1 \leq i \leq p$
$[T]_{2n+p+1} \to T$

That means, if the truth assignment satisfies all clauses of the formula φ, then we have that φ is satisfied, so we can proceed to the output stage.

- **Output Stage**

$[d_{7n+2p+2}]_{2n+p+2} \to F$
$[T]_{2n+p+2} \to T$
$[T \to T']_{2n+p+3}$
$[T']_{2n+p+3} \to \texttt{yes}[\ \ ]_{2n+p+3}$
$[F]_{2n+p+3} \to \texttt{no}[\ \ ]_{2n+p+3}$

After $7n+2p+4$ steps, we obtain an object `yes` or an object `no`, but not both, in the environment, and that is the solution for the `SAT` instance that is being analyzed.

□

Let us notice that from Theorem 1 and Corollary 1 we have:

- $\mathbf{P} = \mathbf{PMC}_{\mathcal{DAM}^0(-d,+n)}$.
- $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{DAM}^0(+d,+n)}$.

Therefore, in the framework of polarizationless P systems with active membranes making use of division rules for elementary and non-elementary membranes, dissolution rules provide a frontier of the efficiency, that is, in that framework passing from forbidden to allowed dissolution rules amounts to passing from non–efficiency to efficiency, assuming that $\mathbf{P} \neq \mathbf{PSPACE}$.

At the beginning of 2005, Gh. Păun proposed a problem (problem **F** from [11]) which can be formally formulated as follows:

"Is the complexity class $\mathbf{PMC}_{\mathcal{DAM}^0(+d,-n)}$ *equal to* $\mathbf{P}$*?"*

The so-called *Păun conjecture* is $\mathbf{PMC}_{\mathcal{DAM}^0(+d,-n)} = \mathbf{P}$, and until now it has not been proved. Nevertheless, in [5] a partial affirmative answer was given when such membrane systems make no use of dissolution rules ($\mathbf{PMC}_{\mathcal{DAM}^0(-d,+n)} = \mathbf{P}$), and assuming that $\mathbf{P} \neq \mathbf{NP}$, a partial negative answer was given when division rules both for elementary and non-elementary membranes are permitted in such membrane systems ($\mathbf{NP} \cup \mathbf{co}-\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{DAM}^0(+d,+n)}$).

5 On efficiency of membrane systems from $\mathcal{DAM}^0_{mc}(-d,-n)$

Dissolution rules play a relevant role in the efficiency of polarizationless P systems which make use of division rules both for elementary and non-elementary membranes. In this section, we show that the syntactical ingredient of minimal cooperation in polarizationless P systems with active membranes (without dissolution and allowing only division for elementary membranes) is enough to solve computationally hard problems in an efficient way. That is, in the previous framework efficiency is reached by trading minimal cooperation for dissolution.

Next, a polynomial time solution to `SAT` problem, by a family $\mathbf{\Pi} = \{\Pi(t) \mid t \in \mathbb{N}\}$ of recognizer P systems from $\mathcal{DAM}^0_{mc}(-d,-n)$ is provided. Each system $\Pi(t)$ will process all Boolean formulas φ in conjunctive normal form with n variables and p clauses, where $t = \langle n,p\rangle$, provided that the appropriate input multiset $cod(\varphi)$ is supplied to the system (through the corresponding input membrane).

Let us recall that the polynomial–time computable function (the *pair function*) $\langle n,p\rangle = ((n+p)(n+p+1)/2)+n$ is a primitive recursive and bijective function

from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. Then, for each $n, p \in \mathbb{N}$, we consider the recognizer P system of degree 2 from $\mathcal{DAM}^0_{mc}(-d, -n)$

$$\Pi(\langle n, p\rangle) = (\Gamma, \Sigma, H, \mu, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, i_{in}, i_{out})$$

defined as follows:

(1) Working alphabet:

$\Gamma = \Sigma \cup \{\texttt{yes}, \texttt{no}, \alpha, \beta', \beta'', \gamma, \gamma', \gamma'', \#\} \cup \{a_{i,k} \mid 1 \leq i \leq n, 1 \leq k \leq i\} \cup$
$\{\beta_k \mid 0 \leq k \leq n + 2p\} \cup \{t_{i,k}, f_{i,k} \mid 1 \leq i \leq n-1, i \leq k \leq n-1\} \cup$
$\{T_{i,j}, F_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq p\} \cup \{c_{j,k} \mid 1 \leq j \leq p-1, j \leq k \leq p-1\} \cup$
$\{c_j \mid 1 \leq j \leq p\} \cup \{d_j \mid 2 \leq j \leq p\}$

where the input alphabet is $\Sigma = \{x_{i,j}, \overline{x}_{i,j}, x^*_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq p\}$.

(2) $H = \{1, 2\}$.

(3) Membrane structure: $\mu = [\ [\ \ ]_2]_1$, that is, $\mu = (V, E)$ where $V = \{1, 2\}$ and $E = \{\{1, 2\}\}$.

(4) Initial multisets: $\mathcal{M}_1 = \{\alpha, \beta_0\}$ and $\mathcal{M}_2 = \{a_{1,1}, \cdots, a_{n,1}\}$.

(5) The set $\mathcal{R}$ of rules consists of the following rules:

1.1 Rules to produce an affirmative answer.

$[\ \alpha\ \gamma\ \longrightarrow\ \gamma'\ ]_1$

$[\ \gamma'\ \longrightarrow\ \gamma''\ ]_1$

$[\ \gamma''\ ]_1\ \longrightarrow\ \texttt{yes}\ [\ \ ]_1$

1.2 Rules to produce a negative answer.

$[\ \beta_k\ \longrightarrow\ \beta_{k+1}\ ]_1$, for $0 \leq k \leq n + 2p - 1$

$[\ \beta_{n+2p}\ \longrightarrow\ \beta'\ ]_1$

$[\ \alpha\ \beta'\ \longrightarrow\ \beta''\ ]_1$

$[\ \beta''\ ]_1\ \longrightarrow\ \texttt{no}\ [\ \ ]_1$

2.1 Rules to generate truth assignments.

$[\ a_{i,i}\ ]_2 \longrightarrow [\ t_{i,i}\ ]_2\ [\ f_{i,i}\ ]_2$, for $1 \leq i \leq n-1$

$[a_{k,i} \longrightarrow\ a_{k,i+1}\ ]_2$, for $2 \leq k \leq n, 1 \leq i \leq k-1$

$[\ a_{n,n}\ ]_2 \longrightarrow [\ T_{n,1}\ ]_2\ [\ F_{n,1}\ ]_2$

2.2 Rules of synchronization.

$$\left.\begin{array}{l}[t_{i,k} \longrightarrow\ t_{i,k+1}\ ]_2 \\ [f_{i,k} \longrightarrow\ f_{i,k+1}\ ]_2\end{array}\right\} 1 \leq i \leq n-2, i \leq k \leq n-2$$

$$\left.\begin{array}{l}[t_{i,n-1} \longrightarrow\ T_{i,1}\ ]_2 \\ [f_{i,n-1} \longrightarrow\ F_{i,1}\ ]_2\end{array}\right\} 1 \leq i \leq n-1$$

2.3 Rules to check clauses.

$$\left.\begin{array}{l}
[T_{i,j}\ x_{i,j} \longrightarrow T_{i,j+1}\ c_{j,j}\ ]_2 \\
[T_{i,j}\ \overline{x}_{i,j} \longrightarrow T_{i,j+1}\ ]_2 \\
[T_{i,j}\ x^*_{i,j} \longrightarrow T_{i,j+1}\ ]_2 \\
[F_{i,j}\ x_{i,j} \longrightarrow F_{i,j+1}\ ]_2 \\
[F_{i,j}\ \overline{x}_{i,j} \longrightarrow F_{i,j+1}\ c_{j,j}\ ]_2 \\
[F_{i,j}\ x^*_{i,j} \longrightarrow F_{i,j+1}\ ]_2
\end{array}\right\} 1 \leq i \leq n, 1 \leq j \leq p-1$$

$$\left.\begin{array}{l}
[T_{i,p}\ x_{i,p} \longrightarrow c_p\ ]_2 \\
[T_{i,p}\ \overline{x}_{i,p} \longrightarrow \#\ ]_2 \\
[T_{i,p}\ x^*_{i,p} \longrightarrow \#\ ]_2 \\
[F_{i,p}\ x_{i,p} \longrightarrow \#\ ]_2 \\
[F_{i,p}\ \overline{x}_{i,p} \longrightarrow c_p\ ]_2 \\
[F_{i,p}\ x^*_{i,p} \longrightarrow \#\ ]_2
\end{array}\right\} 1 \leq i \leq n$$

$$[c_{i,j} \longrightarrow c_{i,j+1}\ ]_2\ 1 \leq i \leq j \leq p-2$$

$$[c_{i,p-1} \longrightarrow c_i\ ]_2\ 1 \leq i \leq p-1$$

2.4 Rules to detect if a truth assignment makes true the input formula.

$$[c_1\ c_2 \longrightarrow d_2\ ]_2$$

$$[d_j\ c_{j+1} \longrightarrow d_{j+1}\ ]_2\,, \text{ for } 1 \leq j \leq p-1$$

$$[\ d_p\ ]_2 \longrightarrow \gamma\ [\ \ ]_2$$

(6) The input membrane is membrane labelled by 2 ($i_{in} = 2$) and the output region is the environment ($i_{out} = env$).

Let us notice that for each $t \in \mathbb{N}$, the system $\Pi(t)$ is deterministic.

6 A formal verification

Let $\varphi = C_1 \wedge \cdots \wedge C_p$ an instance of the SAT problem consisting of p clauses $C_j = l_{j,1} \vee \cdots \vee l_{j,r_j}$, $1 \leq j \leq p$, where $Var(\varphi) = \{x_1, \cdots, x_n\}$, and $l_{j,k} \in \{x_i, \neg x_i \mid 1 \leq i \leq n\}$, $1 \leq j \leq p, 1 \leq k \leq r_j$. Let us assume 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

$$cod(\varphi) = \{x_{i,j} \mid\ x_i \in C_j\} \ \cup\ \{\overline{x}_{i,j} \mid \neg x_i \in C_j\} \ \cup\ \{x^*_{i,j} \mid x_i \notin C_j, \neg x_i \notin C_j\}$$

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:

$$cod(\varphi) = \begin{pmatrix} x_{1,1} & x_{2,1} & \overline{x}_{3,1} & x^*_{4,1} \\ x^*_{1,2} & \overline{x}_{2,2} & x^*_{3,2} & x_{4,2} \\ x^*_{1,3} & \overline{x}_{2,3} & x_{3,3} & \overline{x}_{4,3} \end{pmatrix}$$

That is, j-th row ($1 \leq j \leq p$) represents the j-th clause C_j of φ. We denote $(cod(\varphi))_j^p$ 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,

$$cod(\varphi)_2^p = \begin{pmatrix} x_{1,2}^* \ \overline{x}_{2,2} \ x_{3,2}^* \ x_{4,2} \\ x_{1,3}^* \ \overline{x}_{2,3} \ x_{3,3} \ \overline{x}_{4,3} \end{pmatrix}$$

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 division rules, all truth assignments for the variables $\{x_1, \ldots, x_n\}$ associated with φ are produced. Specifically, 2^n membranes labelled by 2 are generated, each of them encoding a truth assignment. This stage spends n computation steps exactly, 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 p steps, being 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 $p - 1$ steps, being 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 exactly 4 steps.

6.1 Generation stage

At this stage, all truth assignments for the variables associated with the Boolean formula $\varphi(x_1, \ldots, x_n)$ are going to be generated, by applying division rules from **2.1** in membranes labelled by 2. In such manner that in the i–th step ($1 \leq i \leq n-1$) of this stage, division rule associated with object $a_{i,i}$ is triggered, producing objects $t_{i,1}, f_{i,1}$ in the new created membranes labelled by 2. In the last step of this stage the objects produced are $T_{n,1}$ and $F_{n,1}$, respectively.

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) For each i $(1 \leq i \leq n-1)$ at configuration $\mathcal{C}_i$ we have the following:

- $\mathcal{C}_i(1) = \{\alpha\,,\, \beta_i\}$.
- *There are 2^i membranes labelled by 2 such that each of them contains*
  - ⋆ *the input multiset $cod(\varphi)$;*
  - ⋆ *objects $a_{i+1,i+1}, \ldots, a_{n,i+1}$; and*

⋆ *a different subset* $\{r_{1,i}, \ldots, r_{i,i}\}$, *being* $r \in \{t\,,\, f\}$.

(b) $\mathcal{C}_n(1) = \{\alpha\,,\, \beta_n\}$, *and in* $\mathcal{C}_n(2)$ *there are* 2^n *membranes labelled by* 2 *such that each of them contains the input multiset* $cod(\varphi)$, *as well as a different subset* $\{R_{1,1}, \ldots, R_{n,1}\}$, *being* $R \in \{T, F\}$.

Proof. (a) is going to be demonstrated by induction on i.

- The base case $i = 1$ is trivial because at the initial configuration $\mathcal{C}_0$ we have: $\mathcal{C}_0(1) = \{\alpha\,,\, \beta_0\}$ and there exists a single membrane labelled by 2 containing $cod(\varphi)$ and the set $\{a_{1,1}, \ldots, a_{n,1}\}$. Then, configuration $\mathcal{C}_0$ yields configuration $\mathcal{C}_1$ by applying the rules:
  $[\ a_{1,1}\ ]_2 \ \rightarrow\ [\ t_{1,1}\ ]_2\ [\ f_{1,1}\ ]_2$
  $[\ a_{i,1}\ \rightarrow\ a_{i,2}\ ]_2$, for $2 \leq i \leq n$
  $[\ \beta_0\ \rightarrow\ \beta_1\ ]_1$
  Thus, $\mathcal{C}_1(1) = \{\alpha\,,\, \beta_1\}$ and in $\mathcal{C}_1$ there exist two membranes labelled by 2 such that their contents is $cod(\varphi)$ and the set $\{a_{2,2}, \ldots, a_{n,2}\}$. Also, one of those membranes contains object $t_{1,1}$ and the other one object $f_{1,1}$. Hence, the result holds for $i = 1$.
- Supposing that, by induction, result is true for i $(1 \leq i < n-1)$; that is,
  - $\mathcal{C}_i(1) = \{\alpha\,,\, \beta_i\}$.
  - There are 2^i membranes labelled by 2 such that each of them contains
    ⋆ the input multiset $cod(\varphi)$;
    ⋆ objects $a_{i+1,i+1}, \ldots, a_{n,i+1}$; and
    ⋆ a different subset $\{r_{1,i}, \ldots, r_{i,i}\}$, being $r \in \{t\,,\, f\}$.

  Then, configuration $\mathcal{C}_i$ yields configuration $\mathcal{C}_{i+1}$ by applying the rules:
  $[\ t_{k,i}\ \rightarrow\ t_{k,i+1}\ ]_2$, for $1 \leq k \leq i$
  $[\ a_{i+1,i+1}\ ]_2\ \rightarrow\ [\ t_{i+1,i+1}\ ]_2\ [\ f_{i+1,i+1}\ ]_2$
  $[\ a_{k,i+1}\ \rightarrow\ a_{k,i+2}\ ]_2$, for $i+2 \leq k \leq n$
  $[\ \beta_i\ \rightarrow\ \beta_{i+1}\ ]_1$
  Therefore, the following holds:
  - $\mathcal{C}_{i+1}(1) = \{\alpha\,,\, \beta_{i+1}\}$.
  - There are 2^{i+1} membranes labelled by 2 such that each of them contains
    ⋆ the input multiset $cod(\varphi)$;
    ⋆ objects $a_{i+2,i+2}, \ldots, a_{n,i+2}$; and
    ⋆ a different subset $\{r_{1,i+1}, \ldots, r_{i+1,i+1}\}$, being $r \in \{t\,,\, f\}$.

  Hence, the result holds for $i + 1$.

In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{n-1}$ holds:

- $\mathcal{C}_{n-1}(1) = \{\alpha\,,\, \beta_{n-1}\}$.
- There are 2^{n-1} membranes labelled by 2 such that each of them contains
  ⋆ the input multiset $cod(\varphi)$;
  ⋆ object $a_{n,n}$; and
  ⋆ a different subset $\{r_{1,n-1}, \ldots, r_{n-1,n-1}\}$, being $r \in \{t\,,\, f\}$.

On the other hand, configuration $\mathcal{C}_{n-1}$ yields configuration $\mathcal{C}_n$ by applying the rules:

$[\ t_{k,n-1}\ \rightarrow\ T_{k,1}\ ]_2$, for $1 \leq k \leq n-1$
$[\ a_{n,n}\ ]_2\ \rightarrow\ [\ T_{n,1}\ ]_2\ [\ F_{n,1}\ ]_2$
$[\ \beta_{n-1}\ \rightarrow\ \beta_n\ ]_1$

Then, we have $\mathcal{C}_n(1) = \{\alpha\,,\,\beta_n\}$, and in $\mathcal{C}_n(2)$ there are 2^n membranes labelled by 2 such that each of them contains the input multiset $cod(\varphi)$, as well as a different subset $\{R_{1,1},\ldots,R_{n,1}\}$, being $R \in \{T,F\}$.

□

6.2 First Checking stage

At this stage, we try to determine the clauses satisfied for the truth assignment encoded by each membrane labelled by 2. For that, rules from **2.3** will be applied in such manner that in the j-th step ($1 \leq j \leq p$) of this stage, clause j is checked and an object c_j is produced in the case that clause is satisfied.

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) For each i $(1 \leq i \leq p-1)$ at configuration $\mathcal{C}_{n+i}$ we have the following:
- *$\mathcal{C}_{n+i}(1) = \{\alpha\,,\,\beta_{n+i}\}$.*
- *There are 2^n membranes labelled by 2 such that each of them contains*
  - ⋆ *the input multiset $cod(\varphi)^p_{i+1}$ corresponding to the clauses $c_{i+1},\ldots,c_p$;*
  - ⋆ *a different subset $\{R_{1,i+1},\ldots,R_{n,i+1}\}$, being $R \in \{T\,,\,F\}$ encoding a truth assignment for the variables $\{x_1,\ldots,x_n\}$; and*
  - ⋆ *objects $c_{j,i}$ $(1 \leq j \leq i)$ such that clause C_j is satisfied by the truth assignment encoded by such a membrane.*

(b) $\mathcal{C}_{n+p}(1) = \{\alpha\,,\,\beta_{n+p}\}$, and in $\mathcal{C}_{n+p}(2)$ there are 2^n membranes labelled by 2 such that each of them contains objects c_j such that clause C_j is satisfied by the truth assignment encoded by such a membrane. Besides, the multiplicity of object c_j represents the number of values of the truth assignment making true C_j.

Proof. (a) is going to be demonstrated by induction on i.

- In order to prove the base case $i = 1$ let us notice that from the previous proposition we deduce that configuration $\mathcal{C}_n$ verifies: $\mathcal{C}_n(1) = \{\alpha\,,\,\beta_n\}$ and in $\mathcal{C}_n(2)$ there are 2^n membranes labelled by 2 such that each of them contains the input multiset $cod(\varphi)$, as well as a different subset $\{R_{1,1},\ldots,R_{n,1}\}$, being $R \in \{T,F\}$. Besides, configuration $\mathcal{C}_n$ yields configuration $\mathcal{C}_{n+1}$ by applying rule $[\ \beta_n\ \rightarrow\ \beta_{n+1}\ ]_1$ and rules:

$$\left.\begin{array}{l}
[T_{i,1}\ x_{i,1} \rightarrow\ T_{i,2}\ c_{1,1}\ ]_2 \\
[T_{i,1}\ \overline{x}_{i,1} \rightarrow\ T_{i,2}\ ]_2 \\
[T_{i,1}\ x^*_{i,1} \rightarrow\ T_{i,2}\ ]_2 \\
[F_{i,1}\ x_{i,1} \rightarrow\ F_{i,2}\ ]_2 \\
[F_{i,1}\ \overline{x}_{i,1} \rightarrow\ F_{i,2}\ c_{1,1}\ ]_2 \\
[F_{i,1}\ x^*_{i,1} \rightarrow\ F_{i,2}\ ]_2
\end{array}\right\} 1 \leq i \leq n, 1 \leq j \leq p-1$$

Thus, the following holds for configuration $\mathcal{C}_{n+1}$:

- $\mathcal{C}_{n+1}(1) = \{\alpha\,,\, \beta_{n+1}\}$.
- There are 2^n membranes labelled by 2 such that each of them contains
  - ⋆ the input multiset $cod(\varphi)^p_2$ corresponding to the clauses $c_2, \ldots, c_p$;
  - ⋆ a different subset $\{R_{1,2}, \ldots, R_{n,2}\}$, being $R \in \{T\,,\, F\}$ encoding a truth assignment for the variables $\{x_1, \ldots, x_n\}$; and
  - ⋆ objects $c_{1,1}$ such that clause C_1 is satisfied by the truth assignment encoded by such a membrane.

Hence, the result holds for $i = 1$.

- Let us assume that by induction hypothesis, the result holds for i $(1 \leq i < p-1)$; that is,
  - $\mathcal{C}_{n+i}(1) = \{\alpha\,,\, \beta_{n+i}\}$.
  - There are 2^n membranes labelled by 2 such that each of them contains
    - ⋆ the input multiset $cod(\varphi)^p_{i+1}$ corresponding to the clauses $c_{i+1}, \ldots, c_p$;
    - ⋆ a different subset $\{R_{1,i+1}, \ldots, R_{n,i+1}\}$, being $R \in \{T\,,\, F\}$ encoding a truth assignment for the variables $\{x_1, \ldots, x_n\}$; and
    - ⋆ objects $c_{j,i}$ $(1 \leq j \leq i)$ such that clause C_j is satisfied by the truth assignment encoded by such a membrane.

  Besides, configuration $\mathcal{C}_{n+i}$ yields configuration $\mathcal{C}_{n+(i+1)}$ by applying rule $[\ \beta_{n+i}\ \rightarrow\ \beta_{n+(i+1)}\ ]_1$ and rules:

$$\begin{array}{l}
[T_{i,i+1}\ x_{i,i+1} \rightarrow\ T_{i,i+2}\ c_{i+1,i+1}\ ]_2 \\
[T_{i,i+1}\ \overline{x}_{i,i+1} \rightarrow\ T_{i,i+2}\ ]_2 \\
[T_{i,i+1}\ x^*_{i,i+1} \rightarrow\ T_{i,i+2}\ ]_2 \\
[F_{i,i+1}\ x_{i,i+1} \rightarrow\ F_{i,2}\ ]_2 \\
[F_{i,i+1}\ \overline{x}_{i,i+1} \rightarrow\ F_{i,i+2}\ c_{i+1,i+1}\ ]_2 \\
[F_{i,i+1}\ x^*_{i,i+1} \rightarrow\ F_{i,i+2}\ ]_2 \\
[c_{j,i}\ \longrightarrow\ c_{j,i+1}\ ]_2\ :\ 1 \leq j \leq i
\end{array}$$

Thus, the following holds for configuration $\mathcal{C}_{n+(i+1)}$:

- $\mathcal{C}_{n+(i+1)}(1) = \{\alpha\,,\, \beta_{n+(i+1)}\}$.
- There are 2^n membranes labelled by 2 such that each of them contains
  - ⋆ the input multiset $cod(\varphi)^p_{i+2}$ corresponding to the clauses $c_{i+2}, \ldots, c_p$;
  - ⋆ a different subset $\{R_{1,i+2}, \ldots, R_{n,i+2}\}$, being $R \in \{T\,,\, F\}$ encoding a truth assignment for the variables $\{x_1, \ldots, x_n\}$; and
  - ⋆ objects $c_{j,i+1}$ $(1 \leq j \leq i+1)$ such that clause C_j is satisfied by the truth assignment encoded by such a membrane.

Hence, the result holds for $i + 1$.

In order to prove (b) it is enough to notice that, on the one hand, from (a) configuration $\mathcal{C}_{n+p-1}$ verifies the following:

- $\mathcal{C}_{n+p-1}(1) = \{\alpha\,,\,\beta_{n+p-1}\}$.
- There are 2^n membranes labelled by 2 such that each of them contains
  - ⋆ the input multiset $cod(\varphi)_p^p$ corresponding to the clause c_p;
  - ⋆ a different subset $\{R_{1,p},\ldots,R_{n,p}\}$, being $R \in \{T\,,\,F\}$ encoding a truth assignment for the variables $\{x_1,\ldots,x_n\}$; and
  - ⋆ objects $c_{j,p-1}$ $(1 \leq j \leq p-1)$ such that clause C_j is satisfied by the truth assignment encoded by such a membrane.

On the other hand, configuration $\mathcal{C}_{n+p-1}$ yields configuration $\mathcal{C}_{n+p}$ by applying rule $[\ \beta_n\ \rightarrow\ \beta_{n+1}\ ]_1$ and rules:

$$\left.\begin{array}{l}[T_{i,p}\ x_{i,p} \rightarrow\ c_p\ ]_2\\ [T_{i,p}\ \overline{x}_{i,p} \rightarrow\ \#\ ]_2\\ [T_{i,p}\ x^*_{i,p} \rightarrow\ \#\ ]_2\\ [F_{i,p}\ x_{i,p} \rightarrow\ \#\ ]_2\\ [F_{i,p}\ \overline{x}_{i,p} \rightarrow\ c_p\ ]_2\\ [F_{i,p}\ x^*_{i,p} \rightarrow\ \#\ ]_2\end{array}\right\} 1 \leq i \leq n$$

$$[c_{j,p-1}\ \longrightarrow\ c_j\ ]_2\ :\ 1 \leq j \leq p-1$$

Thus, configuration $\mathcal{C}_{n+p}$ holds: $\mathcal{C}_{n+p}(1) = \{\alpha\,,\,\beta_{n+p}\}$ and in $\mathcal{C}_{n+p}(2)$ there are 2^n membranes labelled by 2 such that each of them contains objects c_j such that clause C_j is satisfied by the truth assignment encoded by such a membrane. Besides, the multiplicity of object c_j represents the number of values of the truth assignment making true C_j.

□

6.3 Second Checking stage

At this stage, we try to determine if some truth assignment encoded by a membrane labelled by 2 satisfied all clauses of the input formula. For that, rules from **2.4** will be applied in such manner that object d_j $(2 \leq j \leq p)$ is produced in the case clauses $c_1,\ldots,c_j$ all satisfied. Then, the input formula is satisfied by the truth assignment encoded by a membrane labelled by 2 if and only if object d_p appears in that membrane. This stage spends $p-1$ computation steps.

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 i $(1 \leq i \leq p-1)$ at configuration $\mathcal{C}_{n+p+i}$ we have the following:

- $\mathcal{C}_{n+p+i}(1) = \{\alpha\,,\,\beta_{n+p+i}\}$.
- *There are 2^n membranes labelled by 2 such that each of them contains objects d_{i+1} if and only if the truth assignment encoded in that membrane, makes true clauses $C_1,\ldots,C_{i+1}$.*

(b) φ is satisfiable if and only if at configuration $\mathcal{C}_{n+2p-1}$ there exists some membrane labelled by 2 which contains some object d_p.

Proof. (a) is going to be demonstrated by induction on i.

- In order to prove the base case $i = 1$, let us notice that from the previous proposition we deduce that configuration $\mathcal{C}_{n+p}$ verifies: $\mathcal{C}_{n+p}(1) = \{\alpha\,,\, \beta_{n+p}\}$ and in $\mathcal{C}_{n+p}(2)$ there are 2^n membranes labelled by 2 such that each of them contains objects c_j such that clause C_j is satisfied by the truth assignment encoded by such a membrane. Besides, the multiplicity of object c_j represents the number of values of the truth assignment making true C_j.
  Configuration $\mathcal{C}_{n+p}$ yields configuration $\mathcal{C}_{n+p+1}$ by applying the rules:
  $[\ c_1\ c_2\ \rightarrow\ d_2\ ]_2$
  $[\ \beta_{n+p}\ \rightarrow\ \beta_{n+p+1}\ ]_1$
  Thus, in configuration $\mathcal{C}_{n+p+1}$ the following holds:
  - $\mathcal{C}_{n+p+1}(1) = \{\alpha\,,\, \beta_{n+p+1}\}$.
  - There are 2^n membranes labelled by 2 such that each of them contains objects d_2 if and only if the truth assignment encoded in that membrane, makes true clauses C_1 and C_2.

  Hence, the result holds for $i = 1$.
- Supposing that, by induction, result is true for i $(1 \leq i < n-1)$; that is,
  - $\mathcal{C}_{n+p+i}(1) = \{\alpha\,,\, \beta_{n+p+i}\}$.
  - There are 2^n membranes labelled by 2 such that each of them contains objects d_{i+1} if and only if the truth assignment encoded in that membrane, makes true clauses $C_1, \ldots, C_{i+1}$.

  Then, configuration $\mathcal{C}_{n+p+i}$ yields configuration $\mathcal{C}_{n+p+(i+1)}$ by applying the rules:
  $[\ c_{i+1}\ c_{i+2}\ \rightarrow\ d_2\ ]_2$
  $[\ \beta_{n+p+i}\ \rightarrow\ \beta_{n+p+(i+1)}\ ]_1$
  Thus, in configuration $\mathcal{C}_{n+p+(i+1)}$ the following holds:
  - $\mathcal{C}_{n+p+(i+1)}(1) = \{\alpha\,,\, \beta_{n+p+(i+1)}\}$.
  - There are 2^n membranes labelled by 2 such that each of them contains objects d_{i+2} if and only if the truth assignment encoded in that membrane, makes true clauses $C_1, \ldots, C_{i+2}$.

  Hence, the result holds for $i + 1$.

In order to proof (b), let us note that formula φ is satisfiable if and only if there exists a truth assignment σ making true φ, that is, making true clauses $C_1, \ldots, C_p$. From (a) we deduce that φ is satisfiable if and only at configuration $\mathcal{C}_{n+2p-1}$ there exists some membrane labelled by 2 which contains some object d_p.

□

6.4 Output stage

The output phase starts at the $(n + 2p)$-th step, and takes exactly four steps.

- *Affirmative answer*: if the input formula φ of SAT problem is satisfiable then at least one of the truth assignments from a membrane with label 2 has satisfied all clauses. Thus, a copy of object d_p will appear in that membrane at configuration $\mathcal{C}_{n+2p-1}$. Then, by applying the last rule from **2.4** and rule $[\,\beta_{n+2p-1} \longrightarrow \beta_{n+2p}\,]_1$, objects γ and β_{n+2p} are produced in the skin membrane. At the next step, by applying rules $[\,\alpha\,\gamma \longrightarrow \gamma'\,]_1$ and $[\,\beta_{n+2p} \longrightarrow \beta'\,]_1$, objects γ' and β' are produced in the skin membrane. At the next step, by applying rule $[\,\gamma' \longrightarrow \gamma''\,]_1$, object γ'' is produced in the skin membrane (let us notice that object β' cannot interact with α). Finally, at the step $n + 2p + 3$ by applying rule $[\,\gamma''\,]_1 \longrightarrow \texttt{yes}[\ \ ]_1$, object yes is sent out to the environment and the computation halts.
- *Negative answer*: if the input formula φ of SAT problem is not satisfiable then none of the truth assignments encoded by a membrane with label 2 makes the formula φ true. Thus, object d_p does not appear in any membrane with label 2. Thus, at step $n + 2p$, only rule $[\,\beta_{n+2p-1} \longrightarrow \beta_{n+2p}\,]_1$ is applicable to $\mathcal{C}_{n+2p-1}$. Then, $\mathcal{C}_{n+2p}(1) = \{\alpha\,,\,\beta_{n+2p}\}$. At the next step, by applying rule $[\,\beta_{n+2p} \longrightarrow \beta'\,]_1$ we have $\mathcal{C}_{n+2p+1}(1) = \{\alpha\,,\,\beta'\}$. Then rule $[\,\alpha\,\beta' \longrightarrow \beta''\,]_1$ produces an object β'' in the skin membrane. Finally, at step $n + 2p + 3$ by applying rule $[\,\beta''\,]_1 \longrightarrow \texttt{no}[\ \ ]_1$ releases an object no at the environment. Then, the computation halts and the answer of the computation is no.

6.5 Result

Theorem 4. $\texttt{SAT} \in \mathbf{PMC}_{\mathcal{DAM}^0_{mc}(-d,-n)}$.

Proof: The family of P systems previously constructed verifies the following:

(a) Every system of the family $\mathbf{\Pi}$ is a recognizer P system from $\mathcal{DAM}^0_{mc}(-d,-n)$.
(b) 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: $5np + \frac{3n^2-5n+p^2-3p+6}{2} + n + 4p + 9 \in \Theta((max\{n,p\})^2)$.
   - Initial number of cells: $2 \in \Theta(1)$.
   - Initial number of objects in cells: $n + 2 \in \Theta(n)$.
   - Number of rules: $6np + \frac{3n^2+p^2+3n+5p}{2} + 6 \in \Theta((max\{n,p\})^2)$.
   - Maximal number of objects involved in any rule: $4 \in \Theta(1)$.
(c) The pair (cod, s) of polynomial–time computable functions defined fulfill the following: for each input formula φ of SAT problem, $s(\varphi)$ is a natural number, $cod(\varphi)$ is an input multiset of the system $\Pi(s(\varphi))$, and for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set.
(d) The family $\mathbf{\Pi}$ is polynomially bounded: indeed for each input formula φ of SAT problem, the deterministic P system $\Pi(s(\varphi)) + cod(\varphi)$ takes exactly, in $n + 2p + 3$ steps, being n the number of variables of φ and p the number of clauses.

(e) The family $\mathbf{\Pi}$ is sound with regard to (X, cod, s): indeed for each input formula φ, if the computation of $\Pi(s(\varphi)) + cod(\varphi)$ is an accepting computation, then φ is satisfiable (see Section 6).
(f) The family $\mathbf{\Pi}$ is complete with regard to (X, cod, s): indeed, for each input formula φ such that it is satisfiable, the accepting computation of $\Pi(s(\varphi)) + cod(\varphi)$ is an accepting computation (see Section 6).

Therefore, the family $\mathbf{\Pi}$ of P systems previously constructed solves SAT problem in polynomial time and in a uniform way, according to Definition 5.

□

Corollary 2. $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{DAM}^0_{mc}(-d,-n)}$

Proof: It suffices to notice that SAT problem is a **NP**-complete problem, SAT$\in \mathbf{PMC}_{\mathcal{DAM}^0_{mc}(-d,-n)}$, and the complexity class $\mathbf{PMC}_{\mathcal{DAM}^0_{mc}(-d,-n)}$ is closed under polynomial-time reduction and under complement.

□

7 Limits on efficient computations in $\mathcal{SAM}^0_{mc}(+d, +n)$

In this section we study the computational efficiency of polarizationless P systems with active membranes, dissolution rules and minimal cooperation when separation rules (for elementary and non-elementary membranes) are considered as a mechanism to generate an exponential workspace in linear time. Specifically, we will show that these kind of P systems can only solve problems in class **P** in an efficient way. The proof is inspired on a similar result, obtained in the framework of cell-like P systems with symport/antiport rules and cell separation [7].

Let $\Pi = (\Gamma, \Gamma_0, \Gamma_1, \Sigma, H, H_0, H_1, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$ be a recognizer P system from $\mathcal{SAM}^0_{mc}(+d, +n)$. In what follows we use the concepts of notation from [15].

- We denote by $p(i)$ (resp., $ch(i)$) the label of the parent (resp., a child) of the membrane labelled by i, the parent of the skin membrane is the environment (we write $p(1) = 0$). We denote by $\mathcal{R}_E$ (resp., $\mathcal{R}_C$, $\mathcal{R}_D$ and $\mathcal{R}_S$) the set of evolution rules (resp., communication, dissolution and separation rules) of Π. We will fix total orders in $\mathcal{R}_E$, $\mathcal{R}_C$, $\mathcal{R}_D$ and $\mathcal{R}_S$.
- Let $\mathcal{C}$ be a computation of Π, and $\mathcal{C}_t$ an arbitrary configuration of $\mathcal{C}$. With respect to the number of objects of the system, let us notice that by applying a single rule, this number remains unchanged or decreases by one. Thus, the total number of objects in $\mathcal{C}_t$ is, at most, M, being $M = |\mathcal{M}_0 + ... + \mathcal{M}_q|$.
  With respect to the number of membranes of the system, by applying a separation rule for elementary membranes, an object is removed from the system, no new objects are produced and a new membrane is created. Thus, at most M membranes can be produced by means of this process. Also, by applying a separation rule for non-elementary membranes, the number of objects remains

unchanged but a new membrane is created (when such a rule is applied to a non-elementary membrane, it cannot be applied to that membrane anymore). In this way, no more than $q-2$ new membranes can be generated. Consequently, $q+M+(q-2)=M+2q-2$ is an upper bound of the total number of membranes at $\mathcal{C}_t$.

- In order to identify the membranes created by the application of a separation rule, we modify the labels of the new membranes in the following recursive manner:
  - The label of a membrane will be a pair (i,σ) where $0 \leq i \leq q$ and $\sigma \in \{0,1\}^*$. At the initial configuration, the labels of the membranes are $(1,\lambda),\ldots,(q,\lambda)$. The label of the environment is denoted by $(0,\lambda)$.
  - If a separation rule is applied to a membrane labelled by (i,σ), then the new created membranes will be labelled by $(i,\sigma 0)$ and $(i,\sigma 1)$, respectively. Membrane $(i,\sigma 0)$ will only contain the objects of membrane (i,σ) which belong to Γ_0, and membrane $(i,\sigma 1)$ will only contain the objects of membrane (i,σ) which belong to Γ_1. Only elementary membranes can be separated, so if a membrane i is non-elementary then we denote it by the label (i,λ).
  - If an object evolution rule or a communication rule is applied to a membrane labelled by (i,σ), then after the application of the rule, the membrane keeps its label.
- Let us notice that the number of labels we need to identify all membranes appearing along any computation of a P system from $\mathcal{SAM}^0_{mc}(+d,+n)$ is of the order $O(M+q)$.
- A configuration $\mathcal{C}_t$ of a P system from $\mathcal{SAM}^0_{mc}(+d,+n)$ is described by the current membrane structure and the multisets of labelled objects of the type

$$\{(a,i,\sigma) \,:\, a \in \Gamma, 0 \leq i \leq q, \sigma \in \{0,1\}^*\}$$

  The expression $(a,i,\sigma) \in \mathcal{C}_t$ means that object a belongs to membrane labelled by (i,σ).
- Let $r=[ab \to c]_h \in \mathcal{R}$ be an object evolution rule of Π. We denote by $n \cdot LHS(r,(i,\sigma))$, $n \in \mathbf{N}$, the multiset of labelled objects $(a,i,\sigma)^n(b,i,\sigma)^n$ We denote by $n \cdot RHS(r,(i,\sigma))$ the multiset of labelled objects $(c,i,\sigma)^n$ produced by applying n times rule r over membrane (i,σ). Similarly these concepts are defined for object evolution rules of the forms $[\,ab \to cd\,]_h$ and $[\,a \to c\,]_h$.
- Let $r=[a]_h \to b[\ \ ]_h \in \mathcal{R}$ be a send-out communication rule of Π. We denote by $LHS(r,(i,\sigma))$ the labelled object (a,i,σ). We denote by $RHS(r,(i,\sigma))$ the labelled object $(b,p(i),\tau)$ produced by applying rule r over membrane (i,σ), where $(p(i),\tau)$ is the parent of membrane (i,σ).
- Let $r=a[\ \ ]_h \to [b]_h \in \mathcal{R}$ be a send-in communication rule of Π. We denote by $LHS(r,(i,\sigma))$ the labelled object $(a,p(i),\tau)$, where $(p(i),\tau)$ is the parent of membrane (i,σ). We denote by $RHS(r,(i,\sigma))$ the labelled object (b,i,σ) produced by applying rule r over membrane (i,σ).
- Let $\mathcal{C}_t$ is a configuration of Π, we denote by $\mathcal{C}_t + \{(x,i,\sigma)/\sigma'\}$ the multiset obtained by replacing in $\mathcal{C}_t$ every occurrence of (x,i,σ) by (x,i,σ'). Besides,

$\mathcal{C}_t + m$ (resp., $\mathcal{C}_t \setminus m$) is used to denote that a multiset m of labelled objects is added (resp., removed) to the configuration.

Next, we provide a deterministic algorithm $\mathcal{A}$ working in polynomial time that receives as input a recognizer P system Π from $\mathcal{SAM}^0_{mc}(+d,+n)$ together with an input multiset m of Π. Then algorithm $\mathcal{A}$ reproduces the behaviour of a single computation of such system.

The pseudocode of the algorithm $\mathcal{A}$ is described as follows:

Input: A P system Π from $\mathcal{SAM}^0_{mc}(+d,+n)$ and an input multiset m of Π
  Initialization stage: the initial configuration $\mathcal{C}_0$ of $\Pi + m$
  $t \leftarrow 0$
  while $\mathcal{C}_t$ is a non-halting configuration **do**
    Selection stage: Input $\mathcal{C}_t$, Output $(\mathcal{C}'_t, A)$
    Execution stage: Input $(\mathcal{C}'_t, A)$, Output $\mathcal{C}_{t+1}$
    $t \leftarrow t + 1$
  end while
Output: *Yes* if $\mathcal{C}_t$ is an accepting configuration, *No* otherwise

The selection stage and the execution stage implement a transition step of a recognizer P system Π. Specifically, the selection stage receives as input a configuration $\mathcal{C}_t$ of Π at an instant t. The output of this stage is a pair $(\mathcal{C}'_t, A)$, where A encodes a multiset of rules selected to be applied to $\mathcal{C}_t$, and $\mathcal{C}'_t$ is the configuration obtained from $\mathcal{C}_t$ once the labelled objects corresponding to the application of rules from A have been consumed. The execution stage receives as input the output $(\mathcal{C}'_t, A)$ of the selection stage, and the output is the next configuration $\mathcal{C}_{t+1}$ of $\mathcal{C}_t$. Specifically, at this stage, configuration $\mathcal{C}'_t$ yields configuration $\mathcal{C}_{t+1}$ by adding the labelled objects produced by the application of rules from A.

Next, selection stage and execution stage are described in detail.

Selection stage.

Input: A configuration $\mathcal{C}_t$ of Π at instant t
  $\mathcal{C}'_t \leftarrow \mathcal{C}_t$; $A \leftarrow \emptyset$; $B \leftarrow \emptyset$
  for each membrane (i,σ) of $\mathcal{C}'_t$ according to the lexicographical order **do**
    for each $r \in \mathcal{R}_E$ according to the order chosen **do**
      $n_r \leftarrow$ maximum number of times that r is applicable to (i,σ)
      if $n_r > 0$ **then**
        $\mathcal{C}'_t \leftarrow \mathcal{C}'_t \setminus n_r \cdot LHS(r,(i,\sigma))$
        $A \leftarrow A \cup \{(r, n_r, (i,\sigma))\}$
      end if
    end for
    for each $r \in \mathcal{R}_C$ according to the order chosen **do**
      if $(i,\sigma) \notin B$ and r is applicable to (i,σ) in $\mathcal{C}'_t$ **then**
        $\mathcal{C}'_t \leftarrow \mathcal{C}'_t \setminus LHS(r,(i,\sigma))$
        $A \leftarrow A \cup \{(r, 1, (i,\sigma))\}$
        $B \leftarrow B \cup \{(i,\sigma)\}$
      end if
    end for

```
    for each r ≡ [ a ]_i → b ∈ R_D according to the order chosen do
      if (i,σ) ∉ B and r is applicable to (i,σ) in C'_t then
        C'_t ← C'_t \ {(a,(i,σ))}
        A ← A ∪ {(r,1,(i,σ))}
        B ← B ∪ {(i,σ)}
      end if
    end for
    for r ∈ R_S according to the order chosen do
      if (i,σ) ∉ B and r is applicable to (i,σ) in C'_t then
        C'_t ← C'_t \ LHS(r,(i,σ))
        A ← A ∪ {(r,1,(i,σ))}
        B ← B ∪ {(i,σ)}
      end if
    end for
  end for
```

This algorithm is deterministic and works in polynomial time. Indeed, the cost in time is polynomial in the size of Π because the number of cycles of the external main **for** loop is of order $O(M+q)$, and the number of cycles of the three internal main **for** loops are of order $O(|R|)$. Besides, the cost of each internal loops is of the order $O(M+q)$.

Let us notice that the number of tuples in set A is of the order $O(M)$ because each object in the system can be involved in, at most, one rule and at any configuration $\mathcal{C}_t$ the total number of objects is upper bounded by M. In set A an order is considered in a natural way (a product order concerning the rules, natural numbers and labels).

In order to complete the simulation of a computation step of the system Π, the execution stage takes care of the effects of applying the rules selected in the previous stage: updating the objects according to the RHS of the rules.

Execution stage.

```
Input: The output C'_t and A of the selection stage
    for each (r,n_r,(i,σ)) ∈ A according to the order chosen do
      if r ∈ R_E then
        C'_t ← C'_t + n_r · RHS(r,(i,σ))
      if r ∈ R_C then
        C'_t ← C'_t + RHS(r,(i,σ))
      if r ∈ R_D then
        C'_t ← C'_t + RHS(r,(p(i),σ))
        C'_t ← C'_t + {(x,(p(i),σ)) | x is in membrane (i,σ) in C'_t}
        Update the parent function by removing the membrane (i,σ)
      else if r ∈ R_S then
        C'_t ← C'_t + {(λ,i,σ)/σ0}
        C'_t ← C'_t + {(λ,i,σ1)}
        for each (x,i,σ) ∈ C'_t according to the lexicographical order do
          if x ∈ Γ_0 then
            C'_t ← C'_t + {(x,i,σ)/σ0}
```

```
            else
                C'_t ← C'_t + {(x, i, σ)/σ1}
            end if
        end for
        for each (j, τ) ∈ C'_t do
            if p(j, τ) = (i, σ) and j ∈ H_0 then p(j, τ) = p(i, σ0)
            else if p(j, τ) = (i, σ) and j ∈ H_1 then p(j, τ) = p(i, σ1)
            end if
        end for
    end if
end for
C_{t+1} ← C'_t
```

This algorithm is deterministic and works in polynomial time. Indeed, on the one hand, the number of cycles of the main **for** loop is of order $O(M)$. On the other hand, each cycle of the main **for** loop takes $O(|R|)$ steps plus the number of steps spend by the two secondary **for** loops: the first takes $O(M(M+q))$ steps and the second takes $O(M+q)$ steps.

Theorem 5. $\mathbf{P} = \mathbf{PMC}_{\mathcal{SAM}^0_{mc}(+d,+n)}$.

Proof: It suffices to prove that $\mathbf{PMC}_{\mathcal{SAM}^0_{mc}(+d,+n)} \subseteq \mathbf{P}$. For that, let $X = (I_X, \theta_X)$ be a decision problem in $\mathbf{PMC}_{\mathcal{SAM}^0_{mc}(+d,+n)}$. Let $\{\Pi(n) \mid n \in \mathbb{N}\}$ be a family of P systems from $\mathcal{SAM}^0_{mc}(+d,+n)$ solving X, according to Definition 5. Let (cod, s) be a polynomial encoding associated with that solution. Let us recall that instance $u \in I_X$ of the problem X is processed by the system $\Pi(s(u))+cod(u)$.

Let us consider the following deterministic algorithm $\mathcal{A}'$:

Input: `an instance` u `of the decision problem` X
`Construct the system` $\Pi(s(u)) + cod(u)$
`Run algorithm` $\mathcal{A}$ `with input the system` $\Pi(s(u)) + cod(u)$
Output: *Yes* `if` $\Pi(s(u))+cod(u)$ `has an accepting computation,` *No* `otherwise`

Given an instance u of the decision problem $X = (I_X, \theta_X)$, the following assertions are equivalent:

1. $\theta_X(u) = 1$, that is, the answer of problem X to instance u is affirmative.
2. Every computation of $\Pi(s(u)) + cod(u)$ is an accepting computation.
3. The output of the algorithm with input u is *Yes*.

Therefore, algorithm $\mathcal{A}'$ provides a solution of the decision problem X. Bearing in mind that $\mathcal{A}'$ works in polynomial time, we finally deduce that $X \in \mathbf{P}$.

□

8 Conclusions

The classical definition of polarizationless P systems with active membranes makes use of non-cooperative rules and their object evolution rules are of the form $[\, a \rightarrow u\,]_h$, where a is an object and u is a finite multiset of objects. In that context, the capability of these membrane systems to create an exponential workspace in linear time is implemented by means of division rules (for both elementary and non-elementary membranes). It is well known [5] that only tractable problems can be solved in an efficient way by families of such kind of P systems which do not make use of dissolution rules, that is, $\mathbf{P} = \mathbf{PMC}_{\mathcal{DAM}^0(-d,+n)}$ (in the notation from [5], $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(-d,+n)}$).

In this paper, two new variants are considered. First, by using separation rules inspired on the membrane fission mechanism, instead of division rules in order to create an exponential workspace in linear time. Second, minimal cooperation in object evolution rules is incorporated in polarizationless P systems with active membranes making use of division or separation rules. Object evolution rules with minimal cooperation are of the forms $[a \rightarrow c]_h$, $[ab \rightarrow c]_h$ or $[ab \rightarrow cd]_h$.

The computational efficiency of these models is studied and two main results have been obtained. On the one hand, a polynomial-time and uniform solution to SAT problem by a family of polarizationless P systems with active membranes, minimal cooperation in object evolution rules, without dissolution rules and using only division for elementary membranes, is provided. On the other hand, the limits on efficient computations of polarizationless P systems with active membranes, minimal cooperation in object evolution rules, and using separation rules for elementary membranes and non-elementary membranes, has been established, in the sense that only problems in class $\mathbf{P}$ can be solved by families of such kind of membrane systems in an efficient way.

Consequently, in the framework of polarizationless P systems with active membranes and without dissolution rules, two frontiers of the efficiency have been presented.

- If these membrane systems make use of division rules then passing from non-cooperation to minimal cooperation in object evolution rules amounts passing from non-efficiency to efficiency, that is, $\mathbf{P} = \mathbf{PMC}_{\mathcal{DAM}^0(-d,+n)}$ and $\mathtt{SAT} \in \mathbf{PMC}_{\mathcal{DAM}^0_{mc}(-d,-n)}$
- If these membrane systems make use of minimal cooperation in object evolution rules then passing from separation rules to division rules amounts passing from non-efficiency to efficiency, that is, that is, $\mathbf{P} = \mathbf{PMC}_{\mathcal{SAM}^0_{mc}(+d,+n)}$ and $\mathtt{SAT} \in \mathbf{PMC}_{\mathcal{DAM}^0_{mc}(-d,-n)}$.

It is worth pointing out some remarks regarding to the Păun's conjecture, $\mathbf{P} = \mathbf{PMC}_{\mathcal{DAM}^0(+d,-n)}$. In [5] a key role of the –apparently "innocent"– operation of dissolution rules has been highlighted in the context of computational efficiency of polarizationless P systems with active membranes, assuming that $\mathbf{P} \neq \mathbf{NP}$. Therefore, bearing in mind that $\mathtt{SAT} \in \mathbf{PMC}_{\mathcal{DAM}^0_{mc}(-d,-n)}$, the role

of dissolution rules is now not relevant because in the sense that computationally hard problems can be solved in an efficient way without using these kind of rules. On the other hand, assuming that $\mathbf{P} \neq \mathbf{NP}$, a new partial negative answer to the Păun's conjecture has been obtained

As future work, we propose several research lines related to the computational efficiency of new variants of polarizationless P systems with active membranes.

(a) Membrane systems with membrane separation which make use of classical object evolution rules.
(b) Membrane systems that incorporate minimal cooperation in object evolution rules, removing the restriction about the length of the right-hand side of the rules.
(c) Membrane systems that incorporate an environment with an active role in polarizationless P systems with active membranes through a distinguished alphabet $\mathcal{E}$ similarly to the considered in cell-like P systems with symport/antiport rules (see [7, 8] for details). Then two kind of semantics can be considered: the classical semantics of active membranes or a semantics based on maximal parallelism of the rules except for division or separation rules. Is relevant the role of the environment from a computational complexity point of view?

References

1. A. Alhazov, L. Pan, G. Păun. Trading polarizations for labels in P systems with active membranes, *Acta Informatica*, **41** (2004), 111–144.
2. A. Alhazov, M.J. Pérez–Jiménez. Uniform solution of QSAT using polarizationless active membranes. *Lecture Notes in Computer Science*, **4664** (2007), 122-133.
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, New York, 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. *Lecture Notes in Computer Science*, **3850** (2006), 224–240.
6. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez, F.J. Romero–Campero, A. Romero–Jiménez. Characterizing tractability by cell–like membrane systems. In K.G. Subramanian, K. Rangarajan, M. Mukund (eds.) *Formal models, languages and applications*, World Scientific, Singapore, 2006, pp. 137–154.
7. L.F. Macías-Ramos, M.J. Pérez-Jiménez, A. Riscos-Núñez, L. Valencia-Cabrera. Membrane fission versus cell division: When membrane proliferation is not enough. *Theoretical Computer Science*, **608** (2015), 57–65.
8. L.F. Macías-Ramos, B. Song, L. Valencia-Cabrera, L. Pan, M.J. Pérez-Jiménez. Membrane fission: A computational complexity perspective. *Complexity*, online version 2015 (doi: 10.1002/cplx.21691).
9. 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.

10. Gh. Păun. P systems with active membranes: Attacking **NP**–complete problems. *Journal of Automata, Languages and Combinatorics*, **6**, 1 (2001), 75–90.
11. Gh. Păun. Further twenty six open problems in membrane computing. In M.A. Gutiérrez, A. Riscos, F.J. Romero, D. Sburlan (eds.) *Proceedings of the Third Brainstorming Week on Membrane Computing*, Report RGNC 01/04, Fénix Editora, Sevilla, 2005, pp. 249–262.
12. Gh. Păun, G. Rozenberg, A. Salomaa (eds.). *The Oxford Handbook of Membrane Computing*, Oxford University Press, Oxford, 2010.
13. M.J. Pérez-Jiménez. An approach to computational complexity in Membrane Computing. In G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (eds.). *Membrane Computing, International Workshop, WMC5, Milano, Italy, 2004, Selected Papers, Lecture Notes in Computer Science*, **3365** (2005), 85-109.
14. 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. A preliminary version in E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszil (eds.) *Proceedings of the Fifth International Workshop on Descriptional Complexity of Formal Systems*, DCFS 2003, Budapest, Hungary, July 12-14, 2003, pp. 284-294.
15. M.J. Pérez-Jiménez. A Computational Complexity Theory in Membrane Computing. (invited talk). Membrane Computing, 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009, Revised Selected and Invited Papers. *Lecture Notes in Computer Science*, **5957** (2010), 125-148.
16. A.E. Porreca, G. Mauri, C. Zandron. Complexity classes for membrane systems. *Informatique théorique et applications*, **40**, 2 (2006), 141–162.

Individual memory about the 14^{th} Brainstorming Week on Membrane Computing

Ana Ventura

Universitat de Barcelona
Email: a.venturabarroso@gmail.com

Two weeks ago I didn't even know the existence about this branch of computing, right now I can ay a little bit more but I still know nothing. I think membrane computing is a science which is inspired in the cell (objects that pass trough the membranes). The most important and useful characteristic (especially in physics) is its maxim parallelism, it reduces the computation time because can operate lots of rules at the same time.

If we drew a comparison with quantum, we can identify the objects with the particles, and use the membranes as we want (e.g. as a device) with the appropriate rules. So we can solve a problem with N particles (N tends to infinite) more quickly than with conventional computing.

I have discovered an exciting world. I mean, when I decided to study physics was because of the particles. After learning computational physics I realized that I really like computing, but after this week in Sevilla I open my mind and I think that the possibilities of model with computers are unlimited, and I really want to learn more about it. And I want to link it with particles and quantum world.

I can sincerely say that this week has been one of the most important and amazing weeks in my life, not only for the knowledge, because I have realized what I really want to do in my life. I love learning and improve myself every day, and I can only reach this by working as a researcher. Also, the experience has been enriching for me. I met the most important researchers in this field, I could speak with them and asked questions, I also joked with some of them and I have learned a lot of important skills (especially work as a team, with people that I have never seen before).

In the morning sessions we had provocative presentations, where the researchers presents their projects and its difficulties to go on with it, all the assistants tried to help and to solve this problems. Some of this talks were interesting for me but other didn't because I couldn't understand anything.

One of the topics that I found really interesting is P-Systems with a quantum like-behaviour. How I found this? This began with our project (apply P-Systems to physics), it consists in simulate the Urainum 238 decay experiment. We use the simplest model, membranes (as a Stern-Gerlach device), objects (as a particles) and rules (as probabilities of being up or down). But, I was thinking that with this model the interactions are not considered, so I wanted to know if exists a model for it. By chance, we found a paper named P Systems with a Quantum-Like Behaviour: Background, Definition, and computational Power, before reading it I realized that the author is Alberto Leporati, one of the assistants to the brainstorming, so I asked him about this. This happened the last day so I had not got time to assimilate it and ask questions so now I am reading the paper and trying to understand it, I haven't finished it yet.

What I can understand for the moment is that, this systems is based on the exchange of a quantum of energy among two quantum systems, using the operator creation and annihilation. Each object have associated an amount of energy that can be use to transform objects using rules, which are realized trough linear operators. Its difficulty is avoiding undesired exchanges of energy among the objects, that yield the system to unintended states. I don't know how is visually this model and how it works but I find it useful in the way to model particles and their interactions.

I have been thinking about what P-systems can do in physics and I have some ideas, but I don't know if they are possible.

- Related with quantum behavior, I would like to develop a simulator of particles collision (simulate what occurs in the particles accelerators).
- Another idea that we develop in Sevilla is to use membrane computing for solving continuous problems. By constructing a net it's possible to approximate this problems to a discrete problems, so it's possible to solve with differential equations, for example standing distribution of temperatures (Poisson's equation). Because of membrane computing's maxim parallelism (lots of operations at the same time) it's possible to reduce the computing time. The problem is that in this model is not implemented the relative position (which is fundamental in physics). But I asked Sergiu Ivanov (one of the assistants) and he said that he is developing another model of P system, Automata P-System that may could be useful to solve this kinds of problems. If it's possible to implement the position, membrane computing can be very useful to simulate meteorological models.
- Another idea is related with quantum computer and cryptography. I don't know too much about quantum computer. I have read that the elementary units that compose these parts are two-level quantum system called qubits. The mathematical description of a single qubit is based on the two-dimensional complex Hilbert space C^2. Qubits are thus the quantum extension of the classical notion of bit, but whereas bits can only take two different values 0,1, qubits are

not confined to their two basis (pure) states, $|0\rangle$ and $|1\rangle$, but can also exist in states which are coherent superpositions. Performing a measurement of the state alters it. Indeed, performing a measurement on a qubit in the above superposition will return 0 or 1 with different probabilities. In cryptography it's necessary to do a lots of combinations to find the correct result. With the computation used until now it lasts a lot. But with membrane computing and its max parallelism it could be used as a quantum computer and reduce time of computing. I don't know how to do it yet, but with time and information maybe it's possible.

Author Index