Rudolf Freund and Sergiu Ivanov (eds.)

Proceedings of the

Workshop on Membrane Computing at UCNC 2018

Fontainebleau, France

June 25^{th}, 2018

Technical Report TU Wien

Workshop on Membrane Computing 2018

Preface

The Workshop on Membrane Computing 2018 was collocated with the 17^{th} International Conference on Unconventional Computation and Natural Computation (UCNC 2018), taking place in Fontainebleau, France, June 25^{th} to 29^{th}, 2018; the talks of the Workshop on Membrane Computing were presented on June 25^{th}, 2018.

The aim of the Workshop on Membrane Computing at UCNC 2018 was to bring together researchers working in membrane computing and related fields of unconventional and natural computation, in a friendly atmosphere enhancing communication and cooperation. The Workshop focused on important new theoretical and experimental results in membrane computing and their impact on related fields.

We thank all our colleagues having agreed to join the program committee:

- Artiom Alhazov (Institute of Mathematics and Computer Science, Moldova)
- Lucie Ciencialová (Silesian University in Opava, Czech Republic)
- Erzsébet Csuhaj-Varjú (Eötvös Loránd University, Hungary)
- Rudolf Freund (TU Wien, Austria)
- Sergiu Ivanov (Université Évry, France)
- Raluca Lefticaru (University of Bradford, UK)
- Luca Manzoni (Università degli Studi di Milano-Bicocca, Italy)
- Mario de Jesús Pérez-Jiménez (University of Seville, Spain)
- Antonio Enrico Porreca (Università degli Studi di Milano-Bicocca, Italy)
- Agustín Riscos Núñez (University of Seville, Spain)
- György Vaszil (University of Debrecen, Hungary)
- Gexiang Zhang (Southwest Jiaotong University & Xihua University, China)

The final program consisted of two invited talks given by

- Artiom Alhazov (Institute of Mathematics and Computer Science, Moldova): *Matter-Antimatter Annihilation Rules in Membrane Computing*
  and
- Lucie Ciencialová (Silesian University in Opava, Czech Republic): *APCol Systems with Agent Creation*

as well as a regular contribution refereed by two members of the program committee, presented by David Orellana-Martín,

- David Orellana-Martín, Luis Valencia-Cabrera, Mario J. Pérez-Jiménez: *The Factorization Problem: A New Approach Through Membrane Systems*

and two overview talks given by Sergiu Ivanov and Rudolf Freund:

- Sergiu Ivanov: *A Note on Polymorphic P Systems*
  and
- Artiom Alhazov, Rudolf Freund, Sergiu Ivanov: *Unfair P Systems*.

We thank all the speakers for coming to Fontainebleau and giving their presentations at the Workshop. Moreover, we are very much indebted to Sergey Verlan, Université Paris Est Créteil, the main organizer of UCNC 2018, for providing us with the opportunity to hold our workshop at the IUT of Fontainebleau.

Rudolf Freund (TU Wien, Austria)

Sergiu Ivanov (Université Évry, France)

co-chairs of the Workshop on Membrane Computing at UCNC 2018

Fontainebleau, June 25^{25}, 2018

Workshop on Membrane Computing 2018

Contents

Matter-Antimatter Annihilation Rules in Membrane Computing*

Artiom Alhazov[1], Rudolf Freund[2]

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

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

Abstract. We describe research carried out on matter-antimatter annihilation rules in membrane computing, as an elegant tool of restricted cooperation. While the concept of annihilation rules in membrane computing originates in Spiking Neural P systems, here we mainly focus on two other models: transitional P systems (where this is usually the only source of direct or indirect object-to-object cooperation, occasionally combined with a catalyst), and P systems with active membranes (where these rules often eliminate the need for polarizations and membrane dissolution).
The topics addressed include: computational completeness, deterministic acceptance, small universal systems, uniform families efficiently solving intractable problems, strong NP-completeness, simulating R systems, annihilation without priority, P completeness without membrane division, $P^{\#P}$ characterization with elementary membrane division, and PSPACE characterization with membrane creation.

1 Introduction

Antimatter (e.g., see [125]) is material composed of antiparticles, which have the same mass as particles of ordinary matter but have opposite charge. Encounters between particles and antiparticles lead to the annihilation of the objects, giving energy proportional to the total matter and antimatter mass, in accordance with the mass-energy equivalence equation, $E = mc^2$.

The term *antimatter* was first used by Arthur Schuster in 1898, (see [114]). He hypothesized antiatoms, as well as whole antimatter solar systems, and discussed the possibility of matter and antimatter annihilating each other. The modern theory of antimatter began in 1928, with the papers [34, 35] by Paul Dirac. Dirac realized that the relativistic version of the Schrödinger wave equation for

* The work is supported by National Natural Science Foundation of China (61320106005, 61033003, and 61772214) and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012).

electrons predicted the possibility of antielectrons. These were discovered by Carl D. Anderson in 1932 [21] and named positrons (a contraction of "positive electrons").

In Membrane Computing, the notion of antimatter has first been associated to anti-spikes in the framework of *spiking neural P systems*, introduced as an additional control tool for the flow of spikes in spiking neural P systems, for example, see [92] and [82, 115, 123]. In this context, when one spike and one anti-spike appear in the same neuron, the annihilation occurs and both, spike and anti-spike, disappear. During the Brainstorming Week 2014 in Sevilla the concept of antimatter was further developed for transitional P systems, e.g., see [4] and [5], and later for P systems with active membranes, for example, see [39]. Currently this is an active research area for multiple models of membrane systems.

It turned out that combining annihilation rules, which are a specific form of cooperative erasing, with non-cooperative rules in transitional P systems yields an elegant computationally complete model. Note that immediate annihilation precisely corresponds to *weak priority* of annihilation. It has been shown that this priority may be removed at the price of adding *one* catalyst. Then, it has also been shown that P systems with non-cooperative rules and matter/antimatter annihilation are computationally complete even in the deterministic case. A variant with annihilation generating energy was considered, too.

The work of [4] has been continued in [2]. In particular, the computational completeness results were generalized to computing vectors over $\mathbb{Z}$ instead of $\mathbb{N}$, as well as to computing languages, or even subsets of groups (as languages over symbols and anti-symbols). A number of universality results involving *small* computing devices was obtained in [5], in particular, a universal accepting P system with 53 rules, simulating a model called generalized counter automata introduced there for that purpose.

Besides being studied for computational completeness and universality results involving small computing devices, matter/antimatter annihilation rules have been considered in the model of P systems with active membranes, for instance, see [41]. Under the basic settings, i.e., with weak priority of the matter/antimatter annihilation rules over all the other rules, uniform families of recognizer P systems with active membranes solve Subset Sum, a well-known weakly **NP**-complete problem, and in [38] even a solution for SAT, the famous *strongly* **NP**-complete problem, has been described. Recently it has been shown in [36] that without the weak priority of the matter/antimatter annihilation rules over all the other rules, only the complexity class **P** is characterized within the framework of recognizer P systems.

We would also like to point out a certain informal resemblance with the *Geffert normal form* [57] (in the case of sequential string rewriting), where computational completeness is reached by an elegant combination of context-free rules and a specific kind of erasing rules.

2 Computation Theory Remarks

A computation is a sequence of configurations which starts from an initial configuration. A configuration describes the current status of the computing machine; this may include instances of objects, instances of membranes, and any other entity bearing information. A computation step consists of transformations of symbols by applying specific kinds of rules. Clearly, computations using rules without cooperation of symbols are quite limited in power; for example, it is known that $E0L$-behavior (i.e., the parallel use of non-cooperating rules as in Lindenmayer systems) with standard halting yields $PsREG$ (i.e., semi-linear sets), and accepting P systems are considerably more degenerate.

In this sense, interaction of symbols is a fundamental part of membrane computing, or of theoretical computer science in general. Various ways of interaction of symbols have been studied in membrane computing. For the models with active membranes, the most commonly studied ways are various rules changing polarizations (or even sometimes labels) and membrane dissolution rules. One object may engage such a rule, which would affect the *context* (polarization or label) of other objects in the same membrane, thus affecting the behavior of the latter, e.g., in case of dissolution, such objects find themselves in the parent membrane, which usually has a different label.

In the literature on P systems with active membranes, often only the rules with at most one object on the left side have been studied. Recently, the model with matter/antimatter annihilation rules, e.g., see [2] and [5], have attracted the attention of researchers. It provides a form of *direct* object-object interaction, albeit in a rather restricted way (i.e., by erasing a pair of objects that are in a bijective relation). Although it is known that non-cooperative P systems with antimatter are universal, studying their efficiency turned out to be an interesting line of research. So how does matter/antimatter annihilation compare to other ways of organizing interaction of objects?

First, all known solutions of **NP**-complete (or more difficult) problems in membrane computing rely on the possibility of P systems to obtain *exponential space* in polynomial time; note that object replication alone does not count as building exponential space, since an exponential number can be written, e.g., in binary, in polynomial space. Such a possibility to obtain exponential space in polynomial time is provided by either of membrane division rules, membrane separation rules, see [14, 91, 93], membrane creation rules, see [88], or else by string replication rules, but string-objects lie outside of the scope of the current paper. In tissue P systems, one may apply a similar approach to cells instead of membranes.

Note that in case of cell-like P systems, membrane creation alone (unlike the other types of rules mentioned above) makes it also possible to construct a hierarchy of membranes, let us refer to it as *structured workspace*, which is used to solve **PSPACE**-complete problems. The structured workspace can be alternatively created by elementary membrane division plus non-elementary membrane division (plus membrane dissolution if we have no polarizations).

Besides creating workspace, to solve **NP**-complete problems we need to be able to effectively use that workspace by making objects interact. For instance, it is known that even with membrane division, without polarizations and without dissolution, only problems in **P** may be solved. However, already with two polarizations (the smallest non-degenerate value) P systems can solve **NP**-complete problems. What can be done without polarizations?

One solution is to use the power of switching the context by membrane dissolution. Coupled with non-elementary division, a suitable membrane structure can be constructed so that the needed interactions can be performed solving **NP**-complete or even **PSPACE**-complete problems [16]. It is not difficult to realize that elementary and non-elementary division rules can be replaced by membrane creation rules, or elementary division rules can be replaced by separation rules.

Finally, an alternative way of interaction of objects considered in this paper following [4] is matter/antimatter annihilation. What are the strengths and the weaknesses of these ingredients (the weaker is a combination of ingredients, the stronger is the result, while sometimes weaker ingredients do not let us do what stronger ones can do)?

Using matter/antimatter annihilation makes it possible to carry out multiple simultaneous interactions (for example, the checking phase in our solution for SAT is constant-time instead of linear with respect to the number of clauses), and it is a direct object-object interaction.

The power of dissolution and polarizations is the possibility of mass action (not critical for studying computational efficiency within **PSPACE** as all multiplicities are bounded with respect to the problem size) by changing context.

Using non-elementary division lets us build structured workspace (probably necessary for **PSPACE** if membrane creation is not used instead of membrane division, unless $\mathbf{P}^{\mathbf{PP}}$=**PSPACE**, see [74]), and change non-local context (e.g., the label of the parent membrane).

In [41] it is shown that antimatter is a frontier of tractability in Membrane Computing (for P systems with active membranes without polarizations and without membrane dissolution). It is well known that the polynomial complexity class of recognizer P systems with active membranes without polarizations, without dissolution and with elementary and non-elementary membrane division is exactly the complexity class **P** (see [61], Th. 2). On the other side, it has been proved that if the described P systems model is endowed with dissolution rules, then **NP**-complete problems can be solved even without non-elementary membrane division, the result recently having been improved to exactly characterize $\mathbf{P}^{\#\mathbf{P}}$, see [74]. Even more, with non-elementary membrane division, **PSPACE**-complete problems can be solved, see [16], so exactly **PSPACE** is characterized, see [121]. In this way, dissolution is a frontier of tractability.

The polynomial complexity class of recognizer P systems with active membranes without polarizations, *without* dissolution and with elementary and non-elementary membrane division (i.e., the class which is equal to **P**) is considered with adding antimatter and the corresponding annihilation rules. In this new model, a **uniform** family of P systems is described which solves the Subset Sum

problem, even without non-elementary membrane division. Since the Subset Sum Problem is **NP**-complete, this P systems family shows that antimatter is a new frontier for tractability in Membrane Computing.

In [39] the focus is put on using antimatter and matter/antimatter annihilation rules and on the significantly less power coming up when removing weak priority of these rules over all the other rules.

3 Overview of Results

In [2] the reader can find multiple developments in the area of P systems with antimatter:

> Computational completeness can be obtained with using only non-cooperative rules besides these matter/anti-matter annihilation rules if these annihilation rules have priority over the other rules. Without this priority condition, in addition catalytic rules with one single catalyst are needed to get computational completeness. Even deterministic systems are obtained in the accepting case. Allowing anti-matter objects as input and/or output, we even get a computationally complete computing model for computations on integer numbers. Interpreting sequences of symbols taken in from and/or sent out to the environment as strings, we get a model for computations on strings, which can even be interpreted as representations of elements of a group based on a computable finite presentation.

In [5] small universal P systems with antimatter are explicitly presented.

Theorem 1. *[5] There exist small universal P systems with non-cooperative rules and matter/anti-matter annihilation rules – with* 9 *annihilation rules and, in total,* 53 *rules in the* accepting *case,* 59 *rules in the* generating *case, and* 57 *rules in the* computing *case.*

P systems with active membranes have been shown to be computationally efficient, even without polarizations, without dissolution and without non-elementary membrane division, when enhanced by matter-antimatter annihilation rules.

Theorem 2. *[41]* $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}^0_{-d,+e,-ne,+ant}}$.

In [39] this result has been re-proved using `SAT`, a strongly-**NP**-complete problem instead of `Subset-Sum`.

However, even with non-elementary membrane division, P systems with active membranes only characterize complexity class **P** in case of not having priority of annihilation rules over other rules.

Theorem 3. *[39]* $\mathbf{PMC}_{\mathcal{AM}^0_{-d,+ne,+ant\_NoPri}} = \mathbf{P}$.

The proof follows the one exhibited in [36]. The technique of dependency graphs is used. Since membrane systems of this class of recognizer P systems are required to be confluent by definition, for the proof one can choose any computation, e.g., one where non-cooperative rules have weak priority over annihilation rules, i.e., annihilation would be only applied to objects that do not have non-cooperative rules associated with them. Then, annihilation is useless, and we are left with a P system where there is no interaction between different objects, except membranes sequentialize usage of rules (other than type (a)) associated to them. Yet, for any path from (a, i) to (b, env) in the dependency graph, a suitable computation exists transforming and moving the object accordingly, and all this can be precomputed in $\mathbf{P}$.

In [3], it is shown how P systems with antimatter can simulate R systems with different time and descriptional complexity, depending on whether the underlying R system is simple or general, and how multiplicities are treated (resetting multiplicities to one, obtaining the last multiplicity or multiplicative effect), see Table 1. Here, n is the number of rules, k is the number of objects in the underlying R system, and $k'' \leq k$, $\bar{k} \leq k$.

P	R	mult	steps	$\lvert O\rvert$	$\lvert R_1\rvert$
Π_{13}	s	M	2	$2n+k$	$3n+k$
Π_{14}	s	1	4	$2n+5k+4$	$3n+5k+4$
Π_{15}	g	L	3	$2n+4k+3$	$3n+3k+k''+\bar{k}+3$
Π_{16}	g	1	5	$2n+8k+5$	$3n+7k+k''+\bar{k}+5$

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

What problems can be efficiently solved by P systems with antimatter without using *any* membrane division (or creation or separation)? Is it not known to be within $\mathbf{P}$? In [69] it has been shown that this model characterizes the entire class $\mathbf{P}$, by solving `Horn-SAT`, a known $\mathbf{P}$-complete problem, by P systems with active membranes without creating any membranes.

Using membrane creation, it has previously been shown (using either polarization or dissolution) that appropriate membrane structures can be created to solve any problem in **PSPACE**. In [55] a characterization of **PSPACE** has been shown also for P systems with antimatter (antimatter removing the need for polarizations and dissolution).

It has been shown previously by the Milano group that P systems with active membranes without polarizations and without non-elementary membrane division characterize $\mathbf{P^{PP}}$. A similar result has also been shown in [75] for P system with antimatter, improving previously known bounds of $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP}$.

4 Definitions and Remarks

In the following, the reader is assumed to be familiar with the definitions of transitional membrane systems and membrane systems with active membranes, as well as with register machines.

The main idea of the model with antimatter is the following. For any object a we consider anti-object a^-, as well as a corresponding annihilation rule $aa^- \to \lambda$, which is assumed to exist in all membranes; this annihilation rule could be assumed to remove a pair a, a in zero time, but here we use these annihilation rules as special cooperative rules having priority over all other rules in the sense of weak priority.

Remark 1. Assuming weak priority of all annihilation rules over all other rules, it makes no difference whether the pair of objects is erased in zero time or in one step. Indeed, it would only make a difference when objects would be produced on the right side of the annihilation rule, yet this is not the case.

Remark 2. Annihilation rules are not a feature that can be specified in a concrete P system in a different way, they are always only deduced from the alphabet and the matter-antimatter relation, which relation is a bijection over the alphabet, with the condition that a is equal to its own inverse, i.e., $(a^-)^- = a$, and it has no fixpoint, i.e., $a^- \neq a$ for every a.

Remark 3. The weak priority of annihilation rules, i.e., the priority of all annihilation rules over all other rules, is a feature which is either present for the whole system or not used at all.

Remark 4. For getting better descriptional complexity results, we may omit anti-objects never appearing in the initial configuration and in the possible input of the system, and also never appearing in the right side of any rule, i.e., we remove them from the working alphabet of the system. In addition, we may omit the corresponding annihilation rules. This has no affect on the behaviour of the P system. For example, then we only needed 9 annihilation rules in small computationally complete P systems with antimatter.

Pretty much all computational completeness proofs and universality constructions are based on simulating register machines. We would like to recall that for computational completeness it suffices to have two decrementable registers, plus (also decrementable) input registers if any, and output registers which do not need to be decrementable. For small universality, 8 decrementable registers are used in the strongly universal register machine of Korec, and only 7 needed for weak universality. Moreover, for improving descriptional complexity results, a generalization of register machines has been introduced in [5], essentially embedding ADD-instructions into SUB-instructions; this often lead to not needing any additional rules to simulate ADD-instructions. However, attention needs to be paid to the output, since in membrane computing it is usually assumed that no non-output objects should be present in the output membrane upon halting.

5 Computational Completeness

Theorem 4. *[2] For any $n \geq 1$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$ and $Z \in \{Fun, Rel\}$,*

$$Ps_\delta OP_n\,(ncoo, antim/pri) = PsRE \quad and$$
$$ZPs_\alpha OP_n\,(ncoo, antim/pri) = ZPsRE.$$

Proof. (sketch) Since addition naturally corresponds to non-cooperative rules, we only need to discuss subtract instructions. For each register-object a_r, there are rules $a_r^- \to \#^-$, $a_r a_r^-$, as well as rules $\#\#^- \to \lambda$, $\#^- \to \#\#$ and $\# \to \#\#$.

Then instruction $l_1 : (SUB(r), l_2, l_3)$ can be simulated by rule $l_1 \to l_2 a_r^-$ (decrement case) or by rules $l_1 \to l_1' a_r^-$, $l_1' \to \# l_3$ (zero-test case).

Indeed, decrement is successful if and only if a_r^- annihilates with one register object a_r, and does not produce $\#^-$. On the other hand, zero-test is successful if and only if $\#$ is annihilated with $\#^-$ produced from a_r^- in the absence of a_r. □

The next result from [2] replaces priority by one catalyst. There, rule $ca_r^- \to c\#^-$ is catalytic, and for the zero-test case an additional dummy-object is produced, keeping the catalyst busy for one step before giving it a chance to process a_r^-. This turn is necessary and sufficient for object a_r^- to annihilate with the corresponding register-object if this register is not zero.

Theorem 5. *[2] For any $n \geq 1$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$ and $Z \in \{Fun, Rel\}$,*

$$Ps_\delta OP_n\,(cat_1, antim) = PsRE \quad and$$
$$ZPs_\alpha OP_n\,(cat_1, antim) = ZPsRE.$$

Interestingly, acceptance and computing functions can be done in a deterministic way.

Theorem 6. *[2] For any $n \geq 1$, $k \geq 0$, and $Y \in \{N, Ps\}$,*

$$Y_{detacc} OP_n\,(cat(k), antim/pri) = YRE \quad and$$
$$FunY_{detacc} OP_n\,(cat(k), antim/pri) = FunYRE.$$

Proof. We only need to 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:

For every register r, let $B_-(r) = \{l \mid l : (SUB(r), l', l'') \in P\}$, and the rule

$$a_r^- \to \prod_{l \in B_-(r)} \tilde{l}^- \ \prod_{l \in B_-(r)} \hat{l};$$

moreover, we take the annihilation rules $a_r a_r{}^- \to \lambda$ as well as $\hat{l}\hat{l}^- \to \lambda$ and $\tilde{\tilde{l}}\tilde{l}^- \to \lambda$ for all $l \in B_-(r)$.

Any SUB-instruction $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in B_-(r)$, $l_2, l_3 \in B$, $1 \leq r \leq m$, is simulated by the rules

$$\begin{aligned}
&l_1 \to \bar{l}_1 {a_r}^-, \\
&\bar{l}_1 \to \hat{l}_1{}^- \textstyle\prod_{l \in B_-(r) \setminus \{l_1\}} \tilde{l}, \\
&\hat{l}_1{}^- \to l_2 \textstyle\prod_{l \in B_-(r) \setminus \{l_1\}} \tilde{l}^-, \text{ and} \\
&\tilde{l}_1{}^- \to l_3 \textstyle\prod_{l \in B_-(r) \setminus \{l_1\}} \hat{l}^-.
\end{aligned}$$

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 (which indicates that the register is empty). □

Finally, in [2] the results were generalized to g enerate energy for measuring the time complexity, generating vectors over all integers, generating languages of strings, and even languages over computable finite presentations of groups.

6 Small universality

Theorem 7. *[5] There exist small universal P systems with non-cooperative rules and matter/anti-matter annihilation rules – with* 9 *annihilation rules and, in total,* 53 *rules in the* accepting *case,* 59 *rules in the* generating *case, and* 57 *rules in the* computing *case.*

Proof. We start with a slightly changed variant of the P system from Theorem 4 in [48] (obtained from the universal register machine U_{32} machine in [70]). This modified sequential antiport P system with forbidden contexts can be written with the instructions of a generalized counter machine as follows:

$$\begin{array}{ll}
1 : (q_1, \langle 1 \rangle, \{\}, \langle 7 \rangle, q_1), & 10 : (q_{18}), \langle 5^3 \rangle, \{\}, \langle 4 \rangle, q_{18}), \\
2 : (q_1, \langle\rangle, \{1\}, \langle 6 \rangle, q_4), & 11 : (q_{18}, \langle\rangle, \{5,3\}, \langle 0 \rangle, q_1), \\
3 : (q_4, \langle 5 \rangle, \{\}, \langle 6 \rangle, q_4), & 12 : (q_{18}, \langle 5^2, 0 \rangle, \{5,2\}, \langle\rangle, q_1), \\
4 : (q_4, \langle 6 \rangle, \{5\}, \langle 5 \rangle, q_{10}), & 13 : (q_{18}, \langle 5^2, 2 \rangle, \{5\}, \langle\rangle, q_1), \\
5 : (q_{10}, \langle 7, 6 \rangle, \{\}, \langle 1, 5 \rangle, q_{10}), & 14 : (q_{18}, \langle 5^2 \rangle, \{5,2,0\}, \langle\rangle, q_1) \\
6 : (q_{10}, \langle 7 \rangle, \{6\}, \langle 1 \rangle, q_4), & 15 : (q_{18}, \langle 3 \rangle, \{5\}, \langle\rangle, q_{32}), \\
7 : (q_{10}, \langle\rangle, \{6,7\}, \langle\rangle, q_1), & 16 : (q_{18}, \langle 5 \rangle, \{5\}, \langle 2,3 \rangle, q_{32}), \\
8 : (q_{10}, \langle 6,4 \rangle, \{7\}, \langle\rangle, q_1), & 17 : (q_{32}, \langle 4 \rangle, \{\}, \langle\rangle, q_1), \\
9 : (q_{10}, \langle 6,5 \rangle, \{7,4\}, \langle\rangle, q_{18}), & 18 : (q_{32}, \langle\rangle, \{4\}, \langle\rangle, q_h).
\end{array}$$

For a generalized counter automaton $M = (m, B, l_0, q_h, P)$, let

$$k = 1 + \max_{i:(q,M_-,N,M_+,q') \in P} (|M_-|, |N|).$$

We consider the following rules (common for different instructions of M):

$$\#^- \to \#^k,\ \# \to \#^k,\ \#\#^- \to \lambda,\ a_r \to \#^-,\ a_r a_r^- \to \lambda,\ r \in R.$$

Now we present the simulation of instruction $i : (q, M_-, N, M_+, q') \in P$. First we consider the case when M_- and N have no common elements, and moreover, we also assume that M_- does not overlap with M_+ (otherwise such an instruction can be split into two instructions; notice that this condition is already satisfied in the rules given above).

$$q \to l_i \prod_{r\in N} {a_r}^-,\ l_i \to q'(\prod_{r\in N} \#)(\prod_{r\in M_-} {a_r}^-) \prod_{r\in M_+} a_r.$$

Indeed, the zero-test is successful if *none* of the objects a_r^- generated in the first step annihilates with the corresponding register symbols a_r; they have to change into objects $\#^-$ to annihilate with the same number of objects $\#$ produced in the next step. The decrement is successful if *all* objects ${a_r}^-$ generated in the second step annihilate with the corresponding register symbols a_r. If either decrement or zero-test fail, then at least either one $\#$ or one $\#^-$ will be produced without its annihilation partner, leading to producing objects $\#$ in a geometric progression, ensuring that such computations do not produce any result (notice that no objects $\#$ or $\#^-$ are produced in the first step of the simulation of any instruction).

If the zero-test set N is empty, then the first step is a simple renaming, and thus can be combined with the second step, yielding just one rule

$$q \to q'(\prod_{r\in M_-} {a_r}^-) \prod_{r\in M_+} a_r.$$

Clearly, if M_- and N overlap, such an instruction can be broken down into two subsequent instructions of the generalized counter automaton. However, a more efficient solution with only three rules exists:

$$q \to l_i \prod_{r\in M_-} {a_r}^-,\ l_i \to l_i' \prod_{r\in N} {a_r}^-,\ l_i' \to q(\prod_{r\in N} \#^-) \prod_{r\in M_+} a_r.$$

The generalized counter automaton obtained by rewriting the sequential antiport P system with inhibitors from [48] (with the modifications described above) has 18 instructions, out of which only 4 have overlaps between the decrement multiset and the zero-test set, and other 5 have empty zero-test sets. Hence, applying the constructions described above we get a universal P system with anti-matter having $(18 \times 2 + 4 - 5) + 8 + 2 + (8 + 1) = 54$ rules, i.e., 45 non-cooperative rules and 9 model-defined annihilation rules:

$$\begin{aligned}
&\Pi = (O, [\ \ ]_1, q_1, R_1, 1, 1) \text{ where}\\
&O = \{l_2, l_4, l_6, l_7, l_8, l_9, l_{11}, l_{12}, l_{12}', l_{13}, l_{13}', l_{14}, l_{14}', l_{15}, l_{16}, l_{16}', l_{18}\}\\
&\quad \cup \{q_1, q_4, q_{10}, q_{18}, q_{32}, q_h\} \cup \{a, a^- \mid a \in \{a_j \mid 0 \le j \le 7\} \cup \{\#\}\}
\end{aligned}$$

and R_1 contains the following rules:

$q_1 \to q_1 a_1{}^- a_7$,
$q_1 \to l_2 a_1{}^-$, $l_2 \to q_4 \# a_6$,
$q_4 \to q_4 a_5{}^- a_6$,
$q_4 \to l_4 a_5{}^-$, $l_4 \to q_{10} \# a_6{}^- a_5$,
$q_{10} \to q_{10} a_7{}^- a_6{}^- a_1 a_5$,
$q_{10} \to l_6 a_6{}^-$, $l_6 \to q_4 \# a_7{}^- a_1$,
$q_{10} \to l_7 a_6{}^- a_7{}^-$, $l_7 \to q_1 \# \#$,
$q_{10} \to l_8 a_7{}^-$, $l_8 \to q_1 \# a_6{}^- a_4{}^-$,
$q_{10} \to l_9 a_7{}^- a_4{}^-$, $l_9 \to q_{18} \# \# a_6{}^- a_5{}^-$,
$q_{18} \to q_{18} a_5{}^- a_5{}^- a_5{}^- a_4$,
$q_{18} \to l_{11} a_5{}^- a_3{}^-$, $l_{11} \to q_1 \# \# a_0$,
$q_{18} \to l_{12} a_5{}^- a_5{}^- a_0^-$, $l_{12} \to l'_{12} a_5{}^- a_2{}^-$, $l'_{12} \to q_1 \# \#$,
$q_{18} \to l_{13} a_5{}^- a_5{}^- a_2{}^-$, $l_{13} \to l'_{13} a_5{}^-$, $l'_{13} \to q_1 \#$,
$q_{18} \to l_{14} a_5{}^- a_5{}^-$, $l_{14} \to l'_{14} a_5{}^- a_2{}^- a_0{}^-$, $l'_{14} \to q_1 \# \# \#$,
$q_{18} \to l_{15} a_5{}^-$, $l_{15} \to q_{32} \# a_3{}^-$,
$q_{18} \to l_{16} a_5{}^-$, $l_{16} \to l'_{16} a_5{}^-$, $l'_{16} \to q_{32} \# a_2 a_3$,
$q_{32} \to q_1 a_4{}^-$,
$q_{32} \to l_{18} a_4{}^-$, $l_{18} \to q_h \#$,
$\#^- \to \#^4$, $\# \to \#^4$, $(\# \#^- \to \lambda)$,
$a_r \to \#^-$, $(a_r a_r{}^- \to \lambda)$, $0 \le r \le 7$.

As the rules with l_7 and l'_{12} on the left side have the same right side, we can replace l'_{12} by l_7, thus decreasing the number of non-cooperative rules down to 44. In sum, we finish with 53 rules in the *accepting case*. In the *computing case*, we have to "clean" registers 1 and 6 and add the following four rules and the state q'_h:

$$q_h \to q_h a_1{}^-,\ q_h \to q_h a_6{}^-,\ q_h \to q'_h a_1{}^- a_6{}^-,\ q'_h \to \# \#.$$

The P system now halts with the skin membrane only containing copies of the symbol a_2 representing the output value. Finally, in the *generating case*, we start with the new initial state q_0 and add the two rules $q_0 \to a_2 q_0$ and $q_0 \to q_1$, which allows us to produce, in a non-deterministic way, an input for U_{32} simulating the identity function on the domain of the set to be generated by the P system. □

7 Simulating R Systems

R systems differ from P systems in the following ways. First, all positive multiplicities collapse into one object. Second, all individually applicable rules are applied simultaneously instead of non-determinism. Third, the next configuration consists of only the objects produced in the current step; the idle objects do not persist. These features correspond perfectly to TVDH1-systems, but the nature of objects is atomic, and rule (A, B, C) can be viewed as $A \to C|_{\{\neg b | b \in B\}}$, where A, B, C are sets of objects.

Here we present one construction from [3]. Consider the general case of simulating an R system $S = (V, w_0, R)$ with $R = \{(A_i, B_i, C_i) \mid 1 \leq i \leq n\}$. The simulating P system is given below.

$$
\begin{aligned}
\Pi_{15} &= (O, w_1 = w_0 I_1, R_1 = R_{1,1} \cup R_{1,2} \cup R_{1,3}) \text{ where} \\
O &= V \cup \{a', (a')^-, a'' \mid a \in V\} \cup \{d_i, d_i^- \mid 1 \leq i \leq n\} \cup \{I_1, I_2, I_3\}, \\
R_{1,1} &= \{I_1 \to I_2 d_1 \cdots d_n \prod\nolimits_{a \in V} a'\} \cup \{a \to (a')^- a'' \mid a \in V\}, \\
R_{1,2} &= \{I_2 \to I_3\} \cup \{a'(a')^- \to \lambda,\ (a')^- \to \lambda \mid a \in V\} \\
&\quad \cup \{a' \to \prod\nolimits_{a \in A_i} d_i^- \mid a \in A_i \text{ for some } i,\ 1 \leq i \leq n\} \\
&\quad \cup \{b'' \to \prod\nolimits_{b \in B_i} d_i^- \mid b \in B_i \text{ for some } i,\ 1 \leq i \leq n\}, \\
R_{1,3} &= \{I_3 \to I_1\} \cup \{d_i d_i^- \to \lambda,\ d_i \to \prod\nolimits_{c \in C_i} c,\ d_i^- \to \lambda \mid 1 \leq i \leq n\}.
\end{aligned}
$$

Symbols from C_i are produced from d_i if and only if d_i has not been annihilated, i.e., neither a' nor b'' should produce d_i^- for any $a \in A_i$ and $b \in B_i$. 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 A_i are present and all symbols from the second set B_i are absent. The simulation takes three steps, using an alphabet of $2n + 4k + 3$ symbols and a set of $3n + 3k + k'' + \bar{k} + 3$ rules, where now $\bar{k}$ now denotes the number of symbols appearing in some inhibitor set of any rule.

8 Solving SAT

Propositional Satisfiability is the problem of determining, for a formula of the propositional calculus, if there is an assignment of truth values to its variables for which that formula evaluates to true. By SAT we mean the problem of propositional satisfiability for formulas in conjunctive normal form (CNF).

In the following, we describe the construction of a uniform family of deterministic recognizer P systems with active membranes, without polarizations, without non-elementary membrane division and without dissolution, yet with matter/antimatter annihilation rules, for solving SAT:

Theorem 8. *[39]* $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}^0_{-d,+e,+ant}}$.

Proof. As usual, we will address the resolution via a brute force algorithm, which consists of the following stages (some of the ideas for the design are taken from [31] and [110]):

- *Generation and Evaluation Stage:* All possible assignments associated with the formula are created and evaluated (in this paper we have subdivided this group into *Generation* and *Input processing* groups of rules, which take place in parallel).

– *Checking Stage:* In each membrane we check whether or not the formula evaluates to true for the assignment associated with it.
– *Output Stage:* The system sends out the correct answer to the environment.

Let us consider the pairing function $\langle\ ,\ \rangle$ defined by

$$\langle n, m\rangle = ((n+m)(n+m+1)/2) + n.$$

This function is polynomial-time computable (it is primitive recursive and bijective from $\mathbb{N}^2$ onto $\mathbb{N}$). For any given formula in CNF, $\varphi = C_1 \wedge \cdots \wedge C_m$, with m clauses and n variables $Var(\varphi) = \{x_1, \ldots, x_n\}$ we construct a P system $\Pi(\langle n, m\rangle)$ solving it, where the multiset encoding the problem to be the input of $\Pi(\langle n, m\rangle)$ (for the sake of simplicity, in the following we will omit m and n) is

$$cod(\varphi) = \{x_{i,j} :\ x_j \in C_i\} \cup \{y_{i,j} :\ \ \neg x_j \in C_i\}.$$

For solving SAT by a uniform family of deterministic recognizer P systems with active membranes, without polarizations, without non-elementary membrane division and without dissolution, yet with matter/antimatter annihilation rules, we now construct the members of this family as follows:

$$\begin{aligned}
&\Pi = (O, \Sigma, H = \{1,2\}, \mu = [\ [\ \ ]_2\ ]_1, w_1, w_2, R, i_{in} = 2), \text{ where}\\
&\Sigma = \{x_{i,j}, y_{i,j} \mid 1 \le i \le m,\ 1 \le j \le n\},\\
&O = \{d, t, f, F, \overline{F}, T, \overline{no}_{n+5}, \overline{F}_{n+5}, \overline{yes}_{n+6}, yes_{n+6}, no_{n+6}, yes, no\}\\
&\quad \cup \{x_{i,j}, y_{i,j} \mid 1 \le i \le m,\ -1 \le j \le n\} \cup \{\overline{x}_{i,-1}, \overline{y}_{i,-1} \mid 1 \le i \le m\}\\
&\quad \cup \{c_i, \overline{c}_i \mid 1 \le i \le m\} \cup \{e_j \mid 1 \le j \le n+3\}\\
&\quad \cup \{yes_j, no_j, F_j \mid 0 \le j \le n+5\},\\
&w_1 = no_0\ yes_0\ F_0,\ w_2 = d^n\ e_1,
\end{aligned}$$

and the rules of the set R are given below, presented in the groups Generation, Input Processing, Checking, and Output, together with explanations about how the rules in the groups work.

Generation

G1. $[\ d\ ]_2 \to [\ t\ ]_2 [\ f\ ]_2$;
G2. $[\ t \to \overline{y}_{1,-1} \cdots \overline{y}_{m,-1}\ ]_2$;
G3. $[\ f \to \overline{x}_{1,-1} \cdots \overline{x}_{m,-1}\ ]_2$;
G4. $[\ \overline{x}_{i,-1} \to \lambda\ ]_2$, $1 \le i \le m$;
G5. $[\ \overline{y}_{i,-1} \to \lambda\ ]_2$, $1 \le i \le m$.

In each step j, $1 \le j \le n$, every elementary membrane is divided, one new membrane corresponding with assigning *true* to variable j and the other one with assigning *false* to it. One step later, proper objects are produced to annihilate the input objects associated to variable j: in the *true* case, we introduce the antimatter object for the negated variable, i.e., it will annihilate the corresponding negated variable, and in the *false* case, we introduce the antimatter object for the variable itself, i.e., it will annihilate the corresponding variable. Remaining barred (antimatter) objects not having been annihilated with the input objects, are erased in the next step.

Input Processing

I1. $[\ x_{i,j} \to x_{i,j-1}\ ]_2$, $1 \le i \le m$, $0 \le j \le n$;
I2. $[\ y_{i,j} \to y_{i,j-1}\ ]_2$, $1 \le i \le m$, $0 \le j \le n$;
I3. $[\ x_{i,-1}\ \overline{x}_{i,-1} \to \lambda\ ]_2$, $1 \le i \le m$;
I4. $[\ y_{i,-1}\ \overline{y}_{i,-1} \to \lambda\ ]_2$, $1 \le i \le m$;
I5. $[\ x_{i,-1} \to c_i\ ]_2$, $1 \le i \le m$;
I6. $[\ y_{i,-1} \to c_i\ ]_2$, $1 \le i \le m$.

Input objects associated with variable j decrement their second subscript during $j+1$ steps to -1. The variables not representing the desired truth value are eliminated by the corresponding antimatter object generated by the rules G2 and G3, whereas any of the input variables not annihilated then, shows that the associated clause i is satisfied, which situation is represented by the introduction of the object c_i.

Checking

C1. $[\ e_j \to e_{j+1}\ ]_2$, $1 \le j \le n+1$;
C2. $[\ e_{n+2} \to \overline{c}_1 \cdots \overline{c}_m e_{n+3}\ ]_2$;
C3. $[\ c_i\ \overline{c}_i \to \lambda\ ]_2$, $1 \le i \le m$;
C4. $[\ \overline{c}_i \to F\ ]_2$, $1 \le i \le m$;
C5. $[\ e_{n+3} \to \overline{F}\ ]_2$;
C6. $[\ F\ \overline{F} \to \lambda\ ]_2$, $1 \le i \le m$;
C7. $[\ \overline{F}\ ]_2 \to [\ \ ]_2 T$.

It takes $n+2$ steps to produce objects c_i for every satisfied clause, possibly multiple times. Starting from object e_1, we have obtained the object e_{n+2} until then; from this object e_{n+2}, at step $n+2$ one anti-object is produced for each clause. If any of these clause anti-objects $\overline{c}_i$ is not annihilated, then it is transformed into F, showing that the chosen variable assignment did not satisfy the corresponding clause. It remains to notice that object T is sent to the skin (at step $n+4$) if and only if an object $\overline{F}$ did not get annihilated, i.e., no clause failed to be satisfied.

Output

O1. $[\ yes_j \to yes_{j+1}\ ]_1$, $0 \le j \le n+5$;
O2. $[\ no_j \to no_{j+1}\ ]_1$, $0 \le j \le n+5$;
O3. $[\ F_j \to F_{j+1}\ ]_1$, $0 \le j \le n+4$;
O4. $[\ T \to \overline{no}_{n+5} \overline{F}_{n+5}\ ]_1$;
O5. $[\ no_{n+5}\ \overline{no}_{n+5} \to \lambda\ ]_1$;
O6. $[\ no_{n+6}\ ]_1 \to [\ \ ]_1 no$;
O7. $[\ F_{n+5}\ \overline{F}_{n+5} \to \lambda\ ]_1$;
O8. $[\ F_{n+5} \to \overline{yes}_{n+6}\ ]_1$;
O9. $[\ yes_{n+6}\ \overline{yes}_{n+6} \to \lambda\ ]_1$;
O10. $[\ yes_{n+6}\ ]_1 \to [\ \ ]_1 yes$.

If no object T has been sent to the skin, then the initial *no*-object can count up to $n+6$ and then send out the negative answer *no*, while the initial object F counts up to $n+5$, generates the antimatter object for the *yes*-object at stage $n+6$ and annihilates with the corresponding object *yes* at stage $n+6$. On the other hand, if (at least one) object T arrives in the skin, then the object *no* is annihilated at stage $n+5$ before it would be sent out in the next step, and the object F is annihilated before it could annihilate with the object *yes*, so that the positive answer *yes* can be sent out in step $n+6$.

Finally, we notice that the solution is uniform, deterministic, and uses only rules of types (a_0), (c_0), (e_0) as well as matter/antimatter annihilation rules. The result is produced in $n+6$ steps. □

9 Discussion

We would like to encourage further research on antimatter in membrane computing. Below we list some open problems/potential research directions that we think of being interesting/challenging:

- different models (other than transitional, active membranes and spiking);
- different ingredients that may influence the power/efficiency of the model (other than catalysts and membrane creation);
- different modes (other than maximal parallelism);
- different subsets of pairs of symbols that can annihilate (other than disjointly partitioning the alphabet in O and $\bar{O}$ and a bijection between them);
- different semantics (other than either priority of all annihilation rules over all other rules or else no such priority at all);
- different restrictions (other than no dissolution, no polarizations, and/or forbidding other cooperating rules than annihilation rules);
- different measures of descriptional complexity (other than the total number of rules), and different kinds of complexity (other than descriptional);
- different systems to model (other than R systems);
- etc.

We also refer to the conclusions given in [2], [3], [4], [5], [6], [12], [13], [36], [37], [38], [39], [41], [52], [55], [69], and [75].

References

1. A. Alhazov: P Systems without Multiplicities of Symbol-Objects. *Information Processing Letters* **100** (3), 2006, 124–129.
2. A. Alhazov, B. Aman, R. Freund: P Systems with Anti-matter. In: M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron (Eds.): *Membrane Computing - 15th International Conference, CMC 2014, Prague, 2014, Revised Selected Papers*, Lecture Notes in Computer Science **8961**, Springer, 2014, 66–85.

3. A. Alhazov, B. Aman, R. Freund, S. Ivanov: Simulating R Systems by P Systems. In: A. Leporati, G. Rozenberg, A. Salomaa, C. Zandron (Eds): *Membrane Computing. 17th International Conference*, CMC 2016, Milan, Revised Selected Papers. Lecture Notes in Computer Science **10105**, Springer, 2017, 51–66.
4. A. Alhazov, B. Aman, R. Freund, Gh. Păun: Matter and Anti-matter in Membrane Systems. In: L.F. Macías-Ramos, M.A. Martínez-del-Amor, Gh. Păun, A. Riscos-Núñez, L. Valencia-Cabrera (Eds.): *Twelfth Brainstorming Week on Membrane Computing*. RGNC report 1/2014, University of Sevilla, Fénix Editora, 2014, 1–26.
5. A. Alhazov, B. Aman, R. Freund, Gh. Păun: Matter and Anti-matter in Membrane Systems. In: H. Jürgensen, J. Karhumäki, A. Okhotin (Eds.): *Descriptional Complexity of Formal Systems 2014*, Lecture Notes in Computer Science **8614**, Springer, 2014, 65–76.
6. A. Alhazov, B. Aman, R. Freund, Gh. Păun: P Systems with Matter and Anti-Matter. *Computability in Europe*, Budapest, 2014.
7. A. Alhazov, S. Cojocaru, A. Colesnicov, L. Malahova, M. Petic: A P System for Annotation of Romanian Affixes. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin, G. Rozenberg, A. Salomaa, (Eds.): *Membrane Computing – 14th International Conference*, CMC 2013, Chişinău, Revised Selected Papers, *Lecture Notes in Computer Science* **8340**, Springer, 2013, 80–87.
8. A. Alhazov, 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, Gy. Vaszil, (Eds.): *Membrane Computing – 13th International Conference*, CMC 2012, Budapest. Revised Selected Papers, *Lecture Notes in Computer Science* **7762**, Springer, 2012, 101–111.
9. A. Alhazov, R. Freund: P Systems with Toxic Objects. In: M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron (Eds.): *Membrane Computing - 15th International Conference*, CMC 2014, Prague, Lecture Notes in Computer Science **8961**, Springer, 2014, 99–125.
10. A. Alhazov, R. Freund: Small Catalytic P Systems. In: M. Dinneen (Ed.): Proceedings of the *Workshop on Membrane Computing 2015* (WMC2015), (Satellite workshop of UCNC2015), Research Report CDMTCS-487, Centre for Discrete Mathematics and Theoretical Computer Science, Department of Computer Science, University of Auckland, 2015.
11. A. Alhazov, R. Freund: Variants of Small Universal P Systems with Catalysts. *Fundamenta Informaticae* **138** (1-2), 2015, 227–250.
12. A. Alhazov, R. Freund, P. Sosík: Small P Systems with Catalysts or Anti-Matter Simulating Generalized Register Machines and Generalized Counter Automata. *The Computer Science Journal of Moldova* **23** (3), 2015, 304–328.
13. A. Alhazov, R. Freund, P. Sosík: Small P Systems with Catalysts or Anti-Matter Simulating Generalized Register Machines and Generalized Counter Automata. In: S. Cojocaru, C. Gaindric (Eds), Proceedings of *Workshop on Foundations of Informatics*, FOI-2015, Chişinău, Institute of Mathematics and Computer Science, Valinex, 2015, 304–328.
14. A. Alhazov, Ts.-O. Ishdorj: Membrane Operations in P Systems with Active Membranes. In: Gh. Păun, A. Riscos-Núñez, A. Romero-Jimenez, F. Sancho-Caparrini (Eds.): *Second Brainstorming Week on Membrane Computing*. RGNC report 01/2004, University of Sevilla, 2004, 37–44.
15. A. Alhazov, L. Pan, Gh. Păun: Trading Polarizations for Labels in P Systems with Active Membranes. *Acta Informatica* **41** (2-3), 2004, 111–144.

16. A. Alhazov, M.J. Pérez-Jiménez: Uniform Solution of QSAT using Polarizationless Active Membranes. In: J.O. Durand-Lose, M. Margenstern (Eds.): *Machines, Computations, Universality,5th International Conference, MCU 2007, Orléans, 2007. Proceedings*, Lecture Notes in Computer Science **4664**, Springer, 2007, 122–133.
17. A. Alhazov, Yu. Rogozhin, S. Verlan: On Small Universal Splicing Systems. *International Journal of Foundations of Computer Science* **23** (7), 2012, 1423–1438.
18. A. Alhazov, D. Sburlan: Static Sorting P Systems. In: G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.): Applications of Membrane Computing. *Natural Computing Series*, Springer, 2005, 215–252.
19. A. Alhazov, S. Verlan: Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. *Theoretical Computer Science* **412** (17), 2011, 1581–1591.
20. B. Aman, E. Csuhaj-Varjú, R. Freund: Red-Green P Automata. In: M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron (Eds.): *Membrane Computing. 15th International Conference*, CMC 2014, Prague, Lecture Notes in Computer Science **8961**, Springer, 2014, 139–157.
21. C.D. Anderson: The Positive Electron. *Physical Review* **43**, 1933, 491–494.
22. L. Berman: The Complexity of Logical Theories. *Theoretical Computer Science* **11**, 1980, 71–77.
23. W.D. Blizard: Negative Membership. *Notre Dame Journal of Formal Logic* **31** (3), 1990, 346–368.
24. R. Brijder, A. Ehrenfeucht, M.G. Main, G. Rozenberg: A Tour of Reaction Systems. *International Journal of Foundations of Computer Science* **22** (7), 2011, 1499–1517.
25. P. Budnik: What Is and What Will Be. Mountain Math Software, 2006.
26. C.S. Calude, Gh. Păun: Bio-steps Beyond Turing. *Biosystems* **77**, 2004, 175–194.
27. C.S. Calude, L. Staiger: A Note on Accelerated Turing Machines. *Mathematical Structures in Computer Science* **20** (6), 2010, 1011–1017.
28. M. Cavaliere, R. Freund, A. Leitsch, Gh. Păun: Event-Related Outputs of Computations in P Systems. *Journal of Automata, Languages and Combinatorics* **11** (3), 2006, 263–278.
29. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.): *Applications of Membrane Computing*. Springer, 2006.
30. S.A. Cook: The Complexity of Theorem-proving Procedures. In Proceedings of the *Third Annual ACM Symposium on Theory of Computing, STOC '71*, ACM, New York, NY, 1971, 151–158.
31. A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F. Sancho-Caparrini: A Prolog Simulator for Deterministic P Systems with Active Membranes. *New Generation Computing* **22** (4), 2004, 349–363.
32. 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 Workshop, WMC-CdeA 2002 Curtea de Argeş. Revised Papers*, Lecture Notes in Computer Science **2597**, Springer, 2003, 219–233.
33. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer, 1989.
34. P.A.M. Dirac: The Quantum Theory of the Electron. I. *Proceedings of the Royal Society A* **117** (778). 1928, 610–624.
35. P.A.M. Dirac: The Quantum Theory of the Electron. II. *Proceedings of the Royal Society A* **118** (779). 1928, 351–361.

36. D. Díaz-Pernil, R. Freund, M.A. Gutiérrez-Naranjo, A. Leporati: On the Semantics of Annihilation Rules in Membrane Computing. In: J. Sempere et al. (Eds.): Proceedings of the *16th International Conference on Membrane Computing*, CMC 2015, Valencia, 2015, 91–101.
37. D. Díaz-Pernil, A. Alhazov, R. Freund, M.A. Gutiérrez-Naranjo: Solving SAT with Antimatter in Membrane Computing. In: L.F. Macías-Ramos, Gh. Păun, A. Riscos-Núñez, L. Valencia-Cabrera (Eds.): *Thirteenth Brainstorming Week on Membrane Computing*. RGNC Report 1/2015, University of Sevilla, Fénix Editora, 2015, 121–130.
38. D. Díaz-Pernil, A. Alhazov, R. Freund, M.A. Gutiérrez-Naranjo: Solving SAT with Antimatter in Membrane Computing. In: M.J. Dinneen (Ed.): Proceedings of the *Workshop on Membrane Computing 2015* (WMC2015), (Satellite workshop of UCNC2015), Research Report Series CDMTCS-487, University of Auckland, 48–58.
39. D. Díaz-Pernil, A. Alhazov, R. Freund, M.A. Gutiérrez-Naranjo, A. Leporati: Recognizer P Systems with Antimatter. *Romanian Journal of Information Science and Technology* **18** (3), 2015, 201–217.
40. D.Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: A Logarithmic Bound for Solving Subset Sum with P Systems. In: G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, A. Salomaa (Eds.): *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Revised Selected and Invited Papers*, Lecture Notes in Computer Science **4860**, Springer, 2007, 257–270.
41. D. Díaz-Pernil, F. Peña-Cantillana, A. Alhazov, R. Freund, M.A. Gutiérrez-Naranjo: Antimatter as a Frontier of Tractability in Membrane Computing. *Fundamenta Informaticae* **134** (1-2), 2014, 83–96.
42. W. Dowling, J. Gallier: Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *The Journal of Logic Programming* **3**, 1984, 267–284.
43. A. Ehrenfeucht, G. Rozenberg: Reaction Systems. *Fundamenta Informaticae* **75** (1), 2007, 263–280.
44. R. Freund: Purely Catalytic P Systems: Two Catalysts Can Be Sufficient for Computational Completeness. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin (Eds.): *CMC14 Proceedings – The 14th International Conference on Membrane Computing, Chişinău, 2013*. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, 2013, 153–166.
45. R. Freund, S. Ivanov, L. Staiger: Going Beyond Turing with P Automata: Regular Observer ω-Languages and Partial Adult Halting. *International Journal of Unconventional Computing* **12** (1), 2016, 51–69.
46. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient. *Theoretical Computer Science* **330**, Springer, 2005, 251–266.
47. R. Freund, M. Oswald: A Short Note on Analysing P Systems. *Bulletin of the EATCS* **78**, 2002, 231–236.
48. R. Freund, M. Oswald: A Small Universal Antiport P System with Forbidden Context. In: H. Leung, G. Pighizzini (Eds.): *8th International Workshop on Descriptional Complexity of Formal Systems - DCFS 2006*, Las Cruces, NM. Proceedings DCFS, New Mexico State University, Las Cruces, NM, 2006, 259–266.
49. R. Freund, M. Oswald: Catalytic and Purely Catalytic P Automata: Control Mechanisms for Obtaining Computational Completeness. In: S. Bensch, F. Drewes, R. Freund, F. Otto (Eds.): *Fifth Workshop on Non-Classical Models of Automata and Applications* (NCMA 2013), OCG, Wien, 2013, 133–150.

50. R. Freund, M. Oswald, L. Staiger: ω-P Automata with Communication Rules. In: C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing, International Workshop, WMC 2003, Tarragona, Lecture Notes in Computer Science **2933**, Springer, 2004, 203–217.
51. R. Freund, Gh. Păun: How to Obtain Computational Completeness in P Systems with One Catalyst. In: T. Neary, M. Cook (Eds.): Proceedings of *Machines, Computations and Universality 2013*, MCU 2013, Zürich, *EPTCS* **128**, 2013, 47–61.
52. R. Freund, Gh. Păun: P Systems with Anti-Matter. In: C. Calude, R. Freivalds, I. Kazuo (Eds.): *Computing with New Resources*. Lecture Notes in Computer Science **8808**, Springer, 2014, 409–420.
53. R. Freund, I. Pérez-Hurtado, A. Riscos-Núñez, S. Verlan: A Formalization of Membrane Systems with Dynamically Evolving Structures. *International Journal of Computer Mathematics* **90** (4), 2013, 801–815.
54. 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.
55. Zs. Gazdag, M.A. Gutiérrez-Naranjo: A Characterization of PSPACE with Antimatter and Membrane Creation. In: L.F. Macías-Ramos, Gh. Păun, A. Riscos-Núñez, L. Valencia-Cabrera (Eds.): *Thirteenth Brainstorming Week on Membrane Computing*. RGNC Report 1/2015, University of Sevilla, Fénix Editora, 2015, 159–178.
56. Zs. Gazdag, G. Kolonits: A New Approach for Solving SAT by P Systems with Active Membranes. In: E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, Gy. Vaszil (Eds.): *International Conference on Membrane Computing. 13th International Conference, CMC 2012, Budapest, Revised Selected Papers*. Lecture Notes in Computer Science **7762**, Springer, 2012, 195–207.
57. V. Geffert: Normal Forms for Phrase-Structure Grammars. RAIRO - *Informatique Théorique et Applications* **25** (5), 1991, 473–496.
58. M. Gheorghe, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Frontiers of Membrane Computing: Open Problems and Research Topics. *International Journal of Foundations of Computer Science* **24** (5), 2013, 547–623.
59. J. Gott: Time Travel in Einstein's Universe: The Physical Possibilities of Travel Through Time. Houghton Mifflin, 2001.
60. J. Gruska: Descriptional Complexity of Context-free Languages. Proceedings of *Symposium on Mathematical Foundations of Computer Science*, High Tatras, 1973, 71–83.
61. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero: On the Power of Dissolution in P Systems with Active Membranes. In: R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *Workshop on Membrane Computing. 6th International Workshop, WMC 2005, Vienna, Revised Selected and Invited Papers*. Lecture Notes in Computer Science **3850**, Springer, 2005, 224–240.
62. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A Linear Solution of Subset Sum Problem by Using Membrane Creation. In: J.Mira, J.R. Álvarez (Eds): Mechanisms, Symbols, and Models Underlying Cognition: *First International Work-Conference on the Interplay Between Natural and Artificial Computation*, IWINAC 2005, Las Palmas, Canary Islands, Proceedings, Part I, Lecture Notes in Computer Science **3561**, Springer, 2005, 258–267.
63. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A Uniform Solution to SAT using Membrane Creation. *Theoretical Computer Science* **371** (1-2), 2007, 54–61.

64. D.F. Holt, B. Eick, E.A. O'Brien: *Handbook of Computational Group Theory*. CRC Press, 2005.
65. M. Ionescu, Gh. Păun, T. Yokomori: Spiking Neural P Systems. *Fundamenta Informaticae* **71** (2-3), 2006, 279–308.
66. Ts.-O. Ishdorj, A. Leporati: Uniform Solutions to SAT and 3-SAT by Spiking Neural P Systems with Pre-computed Resources. *Natural Computing* **7** (4), 2008, 519–534.
67. M. Ito, C. Martín-Vide, Gh. Păun: A Characterization of Parikh Sets of ET0L Languages in terms of P Systems. In: M. Ito, Gh. Păun, S. Yu (Eds.): *Words, Semigroups, and Transductions* - Festschrift in Honor of Gabriel Thierrin. World Scientific, 2001, 239–253.
68. S. Ivanov, E. Pelz, S. Verlan: Small Universal Non-deterministic Petri Nets with Inhibitor Arcs. In: H. Jürgensen, J. Karhumäki, A. Okhotin (Eds.): *16th International Workshop on Descriptional Complexity of Formal Systems*, DCFS 2014, Lecture Notes in Computer Science **8614**, Springer, 2014, 186–197.
69. G. Kolonits: A Solution of Horn-SAT with P Systems using Antimatter. In: G. Rozenberg, A. Salomaa, J. Sempere, C. Zandron (Eds): *Membrane Computing. 16th International Conference*, CMC 2015, Lecture Notes in Computer Science **9504**, Springer, 2015, 236–250.
70. I. Korec: Small Universal Register Machines. *Theoretical Computer Science* **168**, 1996, 267–301.
71. K. Krithivasan, V.P. Metta, D. Garg: On String Languages Generated by Spiking Neural P Systems with Anti-spikes. *International Journal of Foundations of Computer Science* **22** (1), 2011, 15–27.
72. J. van Leeuwen, J. Wiedermann: Computation as an Unbounded Process. *Theoretical Computer Science* **429**, 2012, 202–212.
73. A. Leporati, M.A. Gutiérrez-Naranjo: Solving Subset Sum by Spiking Neural P Systems with Pre-computed Resources. *Fundamenta Informaticae* **87** (1), 2008, 61–77.
74. A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron: Simulating Elementary Active Membranes – with an Application to the P Conjecture. In: M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron (Eds.): *Conference on Membrane Computing*. Lecture Notes in Computer Science **8961**, Springer, 2014, 284–299.
75. A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron: The Counting Power of P Systems with Antimatter. *Theoretical Computer Science* **701**, At the intersection of computer science with biology, chemistry and physics - In Memory of Solomon Marcus, 2017, 161–173.
76. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez: Uniform Solutions to SAT and Subset Sum by Spiking Neural P Systems. *Natural Computing* **8** (4), 2009, 681–702.
77. D. Loeb: Sets with a Negative Number of Elements. *Advances in Mathematics* **91**, 1992, 64–74.
78. P. Luisi: The Chemical Implementation of Autopoiesis. In: G. Fleischaker, S. Colonna, P. Luisi (Eds.): Self-Production of Supramolecular Structures, NATO ASI Series **446**, Springer Netherlands, 1994, 179–197.
79. W. Maass, C. Bishop, Eds.: *Pulsed Neural Networks*. MIT Press, Cambridge, 1999.
80. M. Margenstern, Yu. Rogozhin, S. Verlan: Time-Varying Distributed H Systems with Parallel Computations: The Problem is Solved. In: J. Chen, J.H. Reif (Eds.), *DNA Computing, 9th International Workshop on DNA Based Computers*, DNA9,

Madison, WI, 2003. Revised Papers, *Lecture Notes in Computer Science* **2943**, Springer, 2003, 48–53.
81. V.P. Metta, A. Kelemenovà: Universality of Spiking Neural P Systems with Anti-spikes. In: T.V. Gopal, M. Agrawal, A. Li, S.B. Cooper (Eds.): *Theory and Applications of Models of Computation*. TAMC 2014. Lecture Notes in Computer Science **8402**, Springer, 2014, 352–365.
82. V.P. Metta, K. Krithivasan, D. Garg: Computability of Spiking Neural P Systems with Anti-spikes. *New Mathematics and Natural Computation* **8** (3), 2012, 283–295.
83. V.P. Metta, K. Krithivasan, D. Garg: Some Characteristics of Spiking Neural P Systems with Anti-spikes. Proceedings of *11th International Conference on Membrane Computing*, Jena, 2010, 291–303.
84. V.P. Metta, K. Krithivasan, D. Garg: Modelling and Analysis of Spiking Neural P Systems with Anti-spikes using Pnet lab. *Nano Communication Networks* **1** (2), 2011, 141–149.
85. V.P. Metta, K. Krithivasan, D. Garg: Spiking Neural P Systems with Anti-spikes as Transducers. *Romanian Journal of Information Science and Tehnology* **14** (1), 2011, 20–30.
86. M. L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
87. N. Murphy, D. Woods: The Computational Complexity of Uniformity and Semi-uniformity in Membrane Systems. In: M.A. Martínez-del-Amor, E.F. Orejuela-Pinedo, Gh. Păun, I. Pérez-Hurtado, A. Riscos-N''uñez, (Eds.): *Seventh Brainstorming Week on Membrane Computing*, vol. II, Fénix Editora, Sevilla, 2009, 73–84.
88. M. Mutyam, K. Krithivasan: P Systems with Membrane Creation: Universality and Efficiency. In: M. Margenstern, Yu. Rogozhin (Eds.): Proceedings of *Machines, Computations, and Universality, MCU 2001, Chişinău, 2001*. Lecture Notes in Computer Science **2055**, Springer, 2001, 276–287.
89. A. Obtułowicz: Deterministic P Systems for Solving SAT Problem. *Romanian Journal of Information Science and Technology* **4** (1-2), 2001, 195–201.
90. L. Pan, A. Alhazov: Solving HPP and SAT by P Systems with Active Membranes and Separation Rules. *Acta Informatica* **43** (2), 2006, 131–145.
91. L. Pan, Ts.-O. Ishdorj: P Systems with Active Membranes and Separation Rules. *Journal of Universal Computer Science* **10** (5), 2004, 630–649.
92. L. Pan, Gh. Păun: Spiking Neural P Systems with Anti-spikes. *International Journal of Computers, Communications & Control* **IV** (3), 2009, 273–282.
93. L. Pan, M.J. Pérez-Jiménez: Computational Complexity of Tissue-like P Systems. *Journal of Complexity* **26** (3), 2010, 296–315.
94. A. Păun, Gh. Păun: Small Universal Spiking Neural P Systems. *BioSystems* **90**, 2007, 48–60.
95. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61** (1), 2000, 108–143 (and Turku Center for Computer Science-TUCS Report 208, 1998, `www.tucs.fi`).
96. Gh. Păun: DNA Computing Based on Splicing: Universality Results. *Theoretical Computer Science* **231** (2), 2000, 275–296.
97. Gh. Păun: Four (Somewhat Nonstandard) Research Topics. In: L.F. Macías-Ramos, M.A. Martínez-del-Amor, Gh. Păun, A. Riscos-Núñez, L. Valencia-Cabrera (Eds.): Twelfth Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, 2014, 305–309.

98. Gh. Păun: *Membrane Computing. An Introduction*. Springer, 2002.
99. Gh. Păun: P Systems with Active Membranes: Attacking NP-Complete Problems. *Journal of Automata, Languages and Combinatorics* **6** (1), 2001, 75–90.
100. Gh. Păun, M.J. Pérez-Jiménez: Towards Bridging Two Cell-Inspired Models: P Systems and R Systems. *Theoretical Computer Science* **429**, 2012, 258–264.
101. Gh. Păun, G. Rozenberg, A. Salomaa: Computing by splicing. *Theoretical Computer Science* **168** (2), 1996, 321–336.
102. Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing – New Computing Paradigms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1998.
103. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
104. Gh. Păun: Some Quick Research Topics. In: L.F. Macías-Ramos, Gh. Păun, A. Riscos-Núñez, L. Valencia-Cabrera (Eds.): *Thirteenth Brainstorming Week on Membrane Computing*. RGNC Report 1/2015, University of Sevilla, Fénix Editora, 2015, 245–250.
105. M.J. Pérez-Jiménez: A Bioinspired Computing Approach to Model Complex Systems. In: M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron (Eds.): Membrane Computing. 16th International Conference, CMC 2014, Prague, Lecture Notes in Computer Science **8961**, Springer, 2014, 20–34.
106. 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.): *Workshop on Membrane Computing. 5th International Workshop, WMC 2004, Milan, Revised Selected and Invited Papers*. Lecture Notes in Computer Science **3365**, Springer, 2004, 85–109.
107. M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving the Subset-Sum Problem by P Systems with Active Membranes. *New Generation Computing* **23** (4), 2005, 339–356.
108. M.J. Pérez-Jiménez, A. Riscos-Núñez, Á. Romero-Jiménez, D. Woods: Complexity – Membrane Division, Membrane Creation. In: Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010, 302–336.
109. 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.
110. M.J. Pérez-Jiménez, Á. Romero-Jiménez, F. Sancho-Caparrini: Complexity Classes in Models of Cellular Computing with Membranes. *Natural Computing* **2** (3), 2003, 265–285.
111. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: Sublinear-Space P Systems with Active Membranes. In: E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, Gy. Vaszil (Eds.): 13th International Conference on Membrane Computing, CMC 2012, Lecture Notes in Computer Science **7762**, Springer, 2013, 342–357.
112. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages,* 3 volumes. Springer, 1997.
113. D. Sburlan: Further Results on P Systems with Promoters/Inhibitors. *International Journal of Foundations of Computer Science* **17** (1), 2006, 205–221.
114. A. Schuster: Potential Matter. A Holiday Dream. *Nature* **58**, (367) 1898.
115. T. Song, Y. Jiang, Xi. Shi, Xi. Zeng: Small Universal Spiking Neural P Systems with Anti-spikes. *Journal of Computational and Theoretical Nanoscience* **10** (4), 2013, 999–1006.

116. T. Song, L. Luo, J. He, Z. Chen, K. Zhang: Solving Subset Sum Problems by Time-free Spiking Neural P Systems. *Applied Mathematics & Information Sciences* **8** (1), 2014, 327–332.
117. T. Song, L. Pan, J. Wang, I. Venkat, K.G. Subramanian, R. Abdullah: Normal Forms for Spiking Neural P Systems with Anti-spikes. *IEEE Transactions on Nanobioscience* **22** (4), 2012, 352–359.
118. T. Song, X. Wang, Z. Zhang, Z. Chen: Homogeneous Spiking Neural P Systems with Anti-spikes. *Neural Computing and Applications* **24** (7-8), 2014, 1833–1841.
119. P. Sosík, M. Langer: Improved Universality Proof for Catalytic P Systems and a Relation to Non-Semilinear Sets. In: S. Bensch, R. Freund, F. Otto (Eds.): *Sixth Workshop on Non-Classical Models of Automata and Applications* (NCMA 2014), books@ocg.at, BAND 304, 2014, 223–233.
120. P. Sosík, M. Langer: Small (Purely) Catalytic P Systems Simulating Register Machines. *Theoretical Computer Science* **623**, 2016, 65–74.
121. P. Sosík, A. Rodríguez-Patón: Membrane Computing and Complexity Theory: A Characterization of PSPACE. *Journal of Computer and System Sciences* **73** (1), 2007, 137–152.
122. P. Sosík, O. Valík: On Evolutionary Lineages of Membrane Systems. In: R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing. 6th International Workshop, WMC 2005, Vienna. Lecture Notes in Computer Science **3850**, Springer, 2006, 67–78.
123. G. Tan, T. Song, Zh. Chen, Xi. Zeng: Spiking Neural P Systems with Anti-spikes and Without Annihilating Priority Working in a 'Flip-flop' Way. *International Journal of Computing Science and Mathematics* **4** (2), 2013, 152–162.
124. S. Verlan: Look-Ahead Evolution for P Systems. In: Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, A. Salomaa (Eds.): *Membrane Computing. 10th International Workshop*, WMC 2009, Curtea de Argeş, Lecture Notes in Computer Science **5957**, Springer, 2010, 479–485.
125. Wikipedia Antimatter Page: `https://en.wikipedia.org/wiki/Antimatter`
126. The P Systems Website: `http://ppage.psystems.eu`

APCol Systems with Agent Creation

Lucie Ciencialová

Institute of Computer Science
Silesian University in Opava, Czech Republic
lucie.ciencialova@fpf.slu.cz

Abstract. We introduce a specific type of rules for APCol systems (Automaton-like P colonies), variants of P colonies where the environment of the agents is given by a string and during functioning the agents change their own states and process the string similarly to automata. These rules enrich the actioning of APCol systems by agent creation. Finally, we show that even APCol systems with agent creation, systems without inner structure, can solve 3SAT in linear time.

1 Introduction

APCol systems are variants of P colonies (introduced in [5]) – very simple membrane systems inspired by colonies of formal grammars. The APCol system were introduce in 2014 in [1]. The interested reader is referred to [9] for detailed information on P systems (membrane systems) and to [6] and [3] for more information on grammar systems theory; for more details on P colonies consult [4] and [2].

An APCol system consists of a finite number of agents – finite collections of objects in a cell – and a shared environment. The agents have programs consisting of rules. These rules are of two types: they may change the objects of the agents and they can be used for interacting with the joint shared environment – a string.

The computation in APCol systems starts with an input string, representing the environment, and with each agent in its initial state.

Every computational step means a maximally parallel action of the active agents: an agent is active if it is able to perform at least one of its programs, and the joint action of the agents is maximally parallel if no more active agent can be added to the synchronously acting agents.

The computation ends if there are no more applicable programs in the system.

We equip the agents with agent creation programs. They are applicable when a special object appears inside the agent. An agent with the special object can make one copy of itself containing objects specified by the program.

In membrane computing, the notion of creation is not new in P systems. It was introduced in [7] and in [8]. The membrane creation by membrane division is the frequently investigated way for obtaining an exponential working space in a linear time, and on this basis solving hard problems, typically NP-complete problems, in polynomial (often, linear) time. Details can be found in [[10, 8]]. Recently, PSPACE-complete problems were also attacked in this way (see [12]).

In this paper, we recall the definition of APCol systems and the notion of accepting, generating and verifying mode of computation and we introduce the computing mode. Then we define agent creation actions and show that APCol system with agent creation can solve 3SAT in polynomial time.

2 Preliminaries and Basic Notions

Throughout the paper we assume the reader to be familiar with the basics of the formal language theory and membrane computing [11, 9].

For an alphabet Σ, the set of all words over Σ, including the empty word, ε, is denoted by Σ^*. We denote the length of a word $w \in \Sigma^*$ by $|w|$ and the number of occurrences of the symbol $a \in \Sigma$ in w by $|w|_a$.

For every string $x \in \Sigma^*$, $perm(x)$ denotes the set of all permutations of x and $pref(x)$ denotes the set of prefixes of x.

A multiset of objects M is a pair $M = (O, f)$, where O is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : O \to N$; f assigns to each object in O its multiplicity in M. Any multiset of objects M with the set of objects $O = \{x_1, \ldots x_n\}$ can be represented as a string w over alphabet O with $|w|_{x_i} = f(x_i)$; $1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters can also represent the same multiset M, and ε represents the empty multiset.

2.1 SAT

A SAT problem is represented using n propositional variables $x_1, x_2, \ldots, x_n$, which can be assigned truth values 0 (false) or 1 (true). A literal l is either a variable x_i (i.e., a positive literal) or its complement $\neg x_i$ (i.e., a negative literal). A clause α is a disjunction of literals and a CNF formula φ is a conjunction of clauses. A literal l_j of a clause α that is assigned truth value 1 satisfies the clause, and the clause is said to be satisfied. If the literal is assigned truth value 0 then it can be removed from the clause. A clause with a single literal is said to be a unit and its literal has to be assigned value 1 for the clause to be satisfied. The derivation of an empty clause indicates that the formula is unsatisfied for the given assignment. The formula is satisfied if all its clauses are satisfied. The SAT problem consists of deciding whether there exists a truth assignment to the variables such that the formula becomes satisfied. Determining the satisfiability of a formula in the conjunctive normal form where each clause is limited to at most three literals is NP-complete, too; this problem is called 3-SAT.

2.2 APCol Systems

APCol system is a kind of P colonies. It is formed from agents with capacity 2 processing the string. They act according to programs as it is usual in P colonies. Each program is formed from two rules of two types. The first type called rewriting is of the form $a \to b$ and by execution of this rule, the object a

inside the agent is rewritten to the object b. The second type of rules is called replacing. The rewriting rules are of the form $c \leftrightarrow d$ and by use of them, the agent replaces the symbol d in the string by object c initially placed inside the agent. If $c = e$ $(e \leftrightarrow d)$ it means that agent erase symbol d from the string. If $d = e$ $(c \leftrightarrow e)$ it means that agent insert symbol d to the string.

Let us make a few comments about the application of the programs.

1. Agent can act in only one place in the string in one step of computation.
2. $\langle a \leftrightarrow b; c \leftrightarrow e\rangle \quad \Rightarrow \quad b \to ac$ in the string,
   $\langle c \leftrightarrow e; a \leftrightarrow b\rangle \quad \Rightarrow \quad b \to ca$ in the string,
   $\langle a \to b; c \leftrightarrow e\rangle \quad \Rightarrow \quad \varepsilon \to c$ in the string, insert c anywhere to the string, and rewrite a to b inside agent
   $\langle a \leftrightarrow b; e \leftrightarrow d\rangle \quad \Rightarrow \quad d \to \varepsilon$ in the string, delete one d from the string, and rewrite a to b inside agent

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

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

The APCol system is called restricted if its each programs consist of one evolution rule $(a \to b)$ and one communication rule $(a \leftrightarrow b)$.

The computation starts in the initial configuration where all agents contain their initial multiset of objects and there is an input string over the alphabet T on the APCol system. Consequently, an initial configuration of the APCol system is an $(n + 1)$-tuple $c = (\omega; \omega_1, \ldots, \omega_n)$ where w is the initial state of the environment and the other n components are multisets of strings of objects, given in the form of strings, the initial states of the agents.

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

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

By executing programs, the APCol system passes from one configuration to another configuration. A sequence of configurations started from the initial configuration is called a computation. The computation halts when no agent has an applicable program.

Accepting mode In the accepting mode a halting computation is called accepting if and only if at least one agent is in final state and the string to be processed is ε. Hence, the string ω is accepted by the APCol system Π if there exists a computation by Π such that it starts in the initial configuration $(\omega; \omega_1, \ldots, \omega_n)$ and the computation ends by halting in the configuration $(\varepsilon; w_1, \ldots, w_n)$, where at least one of $w_i \in F_i$ for $1 \leq i \leq n$.

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

Verifying mode An input string is verified by the APCol system if the computation process is halting, and moreover, for every i, $1 \leq i \leq m$ - supposed that the length of the input string is m -, each agent rewrites some symbol at position i in some of the environmental strings occurring in the computation process. This means that the agents "visit" eavery position (of the input string or that of its descendants), i.e., they verify the environment.

Computing mode Inspired by computations of a Turing machine we introduce the computing mode. In the computing mode, a computation starts in an initial configuration with an input string possibly different from ε. An input string is accepted if and only if there exists a halting computation starting in a corresponding initial configuration and at least one agent is in a final state. We can also say that an APCol system computes a string that can be found in the environment after a computation halts and at least one agent is in a final state.

3 APCol Systems with Agent Creation

In this section, we introduce the programs for agent creation. For this purpose, we define a new special object @. If an agent contains such an object, the agent makes a copy of itself. This action is done by executing a program formed from two rewriting rules. Let $a@$ be a contents of agent A_1 with program $p_1 = \langle @ \to b; a \to c \rangle$. After execution of the program p_1 there is one new child-agent in the APCol system with the same label and the same set of programs as the parent-agent A_1 has. The contents of the parent-agent after the execution of the program is bc while the contents of the child-agent is ba.

If the parent-agent has a program $p_2 = \langle a \to c; @ \to b \rangle$, then after the execution of the program p_2 the contents of the parent-agent after the execution of the program is bc and the contents of the child-agent is bc, too.

The order of rules determines whether the rewriting rule without @ is used before or after the creation of the child-agent.

Definition 2. *An APCol system with agent creation is a construct*

$$\Pi = (O, e, @, A_1, \ldots, A_n),$$

where

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

    if @ *appears on the left side of the rule in the program, both rules of this program must be rewriting;*
  - $F_i \subseteq O^*$ *is a finite set of final states (contents) of agent* A_i*.*

When an agent obtains the object @ by the execution of a rewriting or a communication rule in the program, the agent must create a new agent in the next step of the computation in the way described in the above definition.

4 Solving 3-SAT

Let φ be a formula in CNF such that every clause α_j in it has at most 3 literals. Let n be a number of its variables and m be the number of its clauses.

As it is usual for APCol systems we add the special symbol \$ to the environmental string as a prefix of the formula.

Lemma 1. *Let* φ *be formula in CNF as mentioned above. Then there exists an APCol system that encodes the string* $\$\varphi$ *into a string of the form*

$$\wedge_1 \boxed{x_{1_1} x_{1_2} x_{1_3}} \wedge_2 \boxed{x_{2_1} x_{2_2} x_{2_3}} \cdots \wedge_m \boxed{x_{m_1} x_{m_2} x_{m_3}},$$

where x_{i_j} *is* l_{i_j} *for a positive literal,* $\overline{l_{i_j}}$ *for a negative literal or* ε*, in linear time.*

The idea of the proof is that there are two agents in APCol system. They cooperate in replacing triplets of literals by one complex symbol, they erase parentheses and add indices to $\wedge$-symbols. One agent consumes at least one unread symbol in every second step of the computation; hence, the number of steps of each computation is at most $2 \cdot (8 \cdot m - 1)$.

Theorem 1. *To every string corresponding to a formula in CNF with at most three literals in each clause (encoded by an APCol system from Lemma 1), there exists a APCol system with agent creation that can determine if it is satisfiable or not.*

The idea of the proof: We construct a APCol system with agent creation with two distinct groups of agents.

The first group will generate agents with an object corresponding to the combination of values of variables inside agents.

The second group of agents will generate agents containing an object corresponding to some triplet-object in a string.

Both groups generate 2^n agents in $2 \cdot n$ steps.

The special agent from the second group puts object false into the string.

After the agents from both groups have been created, the agents from the first group put their contents into the string. Then the second group starts to consume them in such a way that they consume at one moment the triplet-symbols corresponding to the first clause. If the value of the clause is false, the agent rewrites the symbol of the combination of the values of variables into false and continues with consuming triplet-objects. After all triplet-objects have been consumed (this takes m steps), the agents that have not object false inside generate object true and send it to the string.

The same agent that sends the false object to the string can consume a false and a true symbol and replace them by one true symbol.

Every computation with correct input string is halting, and the special agent is in a final state. The environmental string in a halting configuration is formed from one symbol false or at least one symbol true. No other symbols are present in a string at the end of a successful computation.

5 Conclusions

We have introduced the new type of APCol systems that contain programs for agent creation. We have shown that such APCol systems can solve the 3-SAT problem in linear time.

In future research, we will focus on the use of agent creation programs in 2D P colonies – P colonies where agents are placed in an environment having the form of a 2D grid of cells. We will investigate the capability of such a kind of P colonies to simulate, for example, the spread of an infection.

References

1. L. Cienciala, L. Ciencialová, E. Csuhaj-Varjú: Towards P Colonies Processing Strings. In *Proc. BWMC 2014*, Sevilla, 2014, Fénix Editora, Sevilla, Spain, 102–118 (2014).
2. L. Ciencialová, E. Csuhaj-Varjú, L. Cienciala, P. Sosík: P colonies. *Bulletin of the International Membrane Computing Society* **1** (2), 119–156 (2016).
3. E. Csuhaj-Varjú, J. Kelemen, Gh. Păun, J. Dassow (eds.): *Grammar Systems: A Grammatical Approach to Distribution and Cooperation.* Gordon and Breach Science Publishers, Inc., Newark, NJ, USA (1994).
4. A. Kelemenová: P Colonies. In: [9], Chapter 23.1, 584–593.

5. J. Kelemen, A. Kelemenová, Gh. 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)*, 82–86. Boston, Mass (2004).
6. J. Kelemen, A. Kelemenová: A Grammar-Theoretic Treatment of Multiagent Systems. *Cybern. Syst.* **23** (6), 621–633 (1992),
7. S. N. Krishna, R. Rama: A variant of P systems with active membranes: Solving NP-complete problems. *Romanian J. of Information Science and Technology* **2** (4), 357–367 (1999).
8. M. Mutyam, K. Krithivasan: P Systems with Membrane Creation: Universality and Efficiency. In M. Margenstern and Yu. Rogozhin (eds.): Machines, Computations, and Universality, MCU 2001. *Lecture Notes in Computer Science* **2055**. Springer, Berlin, Heidelberg, 276–287 (2001).
9. Gh. Păun, G. Rozenberg, A. Salomaa (eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA (2010).
10. Gh. Păun: P Systems with Active Membranes: Attacking NP-Complete Problems. *Journal of Automata, Languages and Combinatorics* **6** (1), 75–90 (2001).
11. G. Rozenberg, A. Salomaa (eds.): *Handbook of Formal Languages* I-III. Springer, 1997.
12. P. Sosík: Solving a PSPACE-Complete Problem by P Systems with Active Membranes. In: M. Cavaliere, C. Martín-Vide, and Gh. Păun (eds.): *Proceedings of the Brainstorming Week on Membrane Computing*, Report GRLMC 26/03, 305–312 (2012).

The Factorization Problem: A New Approach Through Membrane Systems

David Orellana-Martín, Luis Valencia-Cabrera, and 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
{dorellana, lvalencia, marper}@us.es

Abstract. The *factorization problem* (given a natural number which is the product of two prime numbers, find its decomposition) is conjectured to be intractable and for that it has been used as the key to have secure current cryptosystems. Due to its relevance, this problem has been studied in various computational paradigms, in particular in *membrane computing*. In this framework, *recognizer* P systems were introduced to deal with *decision problems*, that is, problems whose solution/answer is either "`yes`" or "`no`". The factorization problem is a *search problem* (also called *function problem*), where the question is to identify/find *one* solution to the set of possible solutions associated with each instance. In this work, membrane systems *computing partial functions* are shown to (efficiently) solve the factorization problem, improving the previous solutions given in the framework of membrane computing. Specifically, a family of computing polarizationless P systems with active membranes using minimal cooperation and minimal production in object evolution rules, is provided to give a polynomial-time solution to the factorization problem.

1 Introduction

Membrane computing is a computing paradigm inspired by the structure and functioning of living cells. It was introduced in [10] by Gheorghe Păun, describing the basic behavior of these kinds of systems, called *membrane systems* or *P systems*. As a fields within *Natural Computing*, we take inspiration from nature in order to define the semantics of the computational model. Membrane computing takes the minimal functions required to have a living being, that is, replication of DNA, synthesis of proteins, the use of energy to perform metabolic processes and methods to regulate itself (e.g., *apoptosis*). In this way, we can abstract these chemical reactions by means of mathematical tools, for instance, rewriting rules, in order to perform computations. There are basically three approaches to consider computational devices: cell–like membrane systems [10, 11], using the biological membranes arranged hierarchically, inspired by the structure of the cell; tissue–like membrane systems [7], using the biological membranes placed in the nodes of a graph, inspired by the cell inter-communication in tissues; and

neuron-like P systems [5], inspired by the neurophysiological behavior of neurons sending electrical impulses (spikes) along axons from presynaptic neurons to post–synaptic neurons in a distributed and parallel manner. In these variants, polynomial-time solutions of some computationally hard decision problems has been provided: SAT [13], HAM-CYCLE [12], 3-COL [2], KNAPSACK, SUBSET-SUM and PARTITION [15], among others.

Cryptography is a discipline concerning the security of the information in the presence of possible intruders. For this purpose, several *cryptosystems* have been developed, that is, several protocols of security that make harder to a third party to discover the information that two different parts want to interchange. It is important to take into account that the protocol itself does not have to be secret (for instance, for public-key cryptosystems). In fact, the keys of the most important cryptosystems are well-known. The difficulty that resides in them is intrinsic to the problem behind the systems themselves. That is because there is not known any efficient classical algorithm to solve them. For instance, the key to have secure cryptosystems such as *RSA*, introduced by R. Rivest, A. Shamir and L. Adleman in [16], is the following version of the *factorization problem*: *given a natural number n which is the product of two large primes, find its decomposition.* Such a number n is used as the *modulus* for both public and private keys. In order to attack it, we need to factorize n into its prime factors. Many systems, such as banks, medical databases and critical systems with confidential information keep their data secure thanks to this method. The factorization problem is conjectured to be computationally intractable, as some of the problems used in cryptography, such as KNAPSACK [1] in Merkle-Hellman cryptosystem [8] and DISCRETE LOGARITHM used in Diffie-Hellman key exchange [3], among others.

In the framework of membrane computing, there were attempts to give a solution to the factorization problem in [6, 9, 22] with membrane systems. In [6], a solution is given by using a family of P system with active membranes and electrical charges, using 4-division rules, that is, by applying this kind of rules a membrane produces four new membranes, and object evolution rules whose left-hand side and right-hand sides can have three objects. In [9], a solution is given by means of a family of asynchronous P systems with active membranes and electrical charges which use cooperation in object evolution rules and division rules for non-elementary membranes. In [22], a solution is given by using a family of tissue P systems with cell division which use symport/antiport rules of length at most 4.

In this work, a solution of the factorization problem is given by means of a version of polarizationless P system with active membranes using 2-division rules only for elementary membranes (that is, by applying this kind of rule an elementary membrane produces only two new membranes) without dissolution rules. Specifically, these systems use *minimal cooperation* (the left-hand sides have at most two objects) and *minimal production* (the right-hand sides have only one object) in objects evolution rules. In order to develop simulators running on real computers, this kind of membrane system is interesting, for instance, from

[1] That is, in fact, an **NP**-complete problem.

the GPU computing point of view, just because the algorithm proposed seems easily translatable to this parallel computing paradigm.

The solution provided in this paper improves the previous ones given in [6, 9, 22]. In this regard, it should be recalled that families of polarizationless P system with active membranes using division rules and without dissolution rules only can (efficiently) solve problems in class **P** [4].

The paper is organized as follows. The next section briefly describes some basic aspects in order to make the work self-contained. In Section 3 we define the syntax and semantics of polarizationless P systems with active membranes by using membrane division rules and minimal cooperation in evolution rules. Next, *computing* membrane systems are introduced in Section 4. Section 5 is devoted to define the family of P systems that return the factorization of a given number, followed by an overview of the computation to know what is happening in each step. The paper ends with some open problems and concluding remarks.

2 Preliminaries

In order to have a precise definition of all the terms that are going to be used later, we are going to introduce them here.

2.1 Partial Functions

A function f is a set whose elements are ordered pairs verifying the following: $\forall x \forall y \forall z\, [(x, y) \in f \wedge (x, z) \in f \rightarrow y = z]$. The set $\{x \mid \exists y\, (x, y) \in f\}$ is called the *domain* of f and it is denoted by $dom(f)$. The set $\{x \mid \exists y\, (y, x) \in f\}$ is called the *range* of f and it is denoted by $rang(f)$.

Given two sets A, B, a *partial function* f from A onto B is a set verifying that $f \subseteq A \times B$, $dom(f) \subseteq A$ and $rang(f) \subseteq B$. In the case $dom(f) = A$ the function is called a *total function* from A onto B.

2.2 Alphabets and Multisets

An *alphabet* Γ is a non-empty set. A *multiset* over an alphabet Γ is an ordered pair (Γ, f) where f is a total function from Γ onto the set of natural numbers $\mathbb{N}$. The support of a multiset $\mathcal{M} = (\Gamma, f)$ is defined as $supp(\mathcal{M}) = \{x \in \Gamma \mid f(x) > 0\}$. A multiset is finite (respectively, empty) if its support is a finite (resp., empty) set. If Γ is a finite set then each multiset $\mathcal{M} = (\Gamma, f)$ is finite and it will be represented by $\{a^{f(a)} \mid a \in supp(\mathcal{M})\}$.

2.3 Graphs and Trees

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 [1] for details).

2.4 Binary Representation of Natural Numbers

For each natural number $n \in \mathbb{N}$ we denote $[0, 2^{n+1}) = \{x \in \mathbb{N} \mid 0 \leq x < 2^{n+1}\}$. Next, we define the binary representation of natural numbers. For each natural number $x \in \mathbb{N}$, $x \geq 1$, there exist a unique tuple $(x_0, \ldots, x_n) \in \{0,1\}^{n+1}$, with $n \in \mathbb{N}$ and $x_n = 1$, such that $x = x_0 \cdot 2^0 + x_1 \cdot 2^1 + \ldots + x_n \cdot 2^n$, that is, $x \in [0, 2^{n+1})$. We say that the finite sequence $x_n \cdots x_1 x_0$ or the tuple $(x_0, \ldots, x_n)$, is the *binary representation* of natural number x. We denote $k_x = n$, that is, $1 + k_x$ is the number of digits of natural number $x \geq 1$ in its binary representation.

Binary representation and, in general, representation of numbers different to unary one, is useful because the *size* of a natural number x (the number of bits used) in unary representation is x but in binary representation its size is $1 + \lfloor \log_2(x) \rfloor$. Consequently, the size of a natural number expressed in unary form is exponential in the size of that number expressed in binary form. It is worth pointing out that within the framework of Membrane Computing work is being carried out on multisets, and it is usual represents instances of abstract problems in unary representation (natural numbers are encoded by the multiplicities of some objects in a multiset).

Given a partial function f from $\mathbb{N}^q$ into $\mathbb{N}^r$, with $q \geq 1, r \geq 1$, for each $\boldsymbol{x} = (x_1, \ldots, x_q) \in \mathbb{N}^q$ and $\boldsymbol{y} = (y_1, \ldots, y_r) \in \mathbb{N}^r$ such that $f(\boldsymbol{x}) = \boldsymbol{y}$ there exists a unique natural number $k_{(\boldsymbol{x},\boldsymbol{y})}$ defined as follows:

$$k_{(\boldsymbol{x},\boldsymbol{y})} = \min\{k \in \mathbb{N} \mid [k \geq 1] \wedge [x_1, \ldots, x_q, y_1, \ldots, y_r \in [0, 2^k)]\}$$

That is, $k_{(\boldsymbol{x},\boldsymbol{y})}$ is the smallest natural number where natural numbers $x_1, \ldots, x_q$, $y_1, \ldots, y_r$ can be represented in binary form with, at most, $k_{(\boldsymbol{x},\boldsymbol{y})}$ digits.

3 Polarizationless P Systems with Active Membranes

In [11], P systems with active membranes are introduced as a universal computing model, that is, it has the same computational power than a Turing machine. There, a linear time solution to `SAT` is given, using P systems with active membranes with polarizations and membrane division. As polarizations seem to be a very powerful tool from the computational complexity point of view, a new framework not using them is created, the so-called *polarizationless* P systems with active membranes. In [4], a frontier of efficiency is given by means of dissolution rules. Passing from forbidding them to allowing them is the same as passing from non-efficiency to (strong) efficiency. In fact, not only **NP**-complete problems can be solved in an efficient way, but a solution to the problem `QSAT`, a

well-known **PSPACE**-complete problem [14], by means of recognizer polarizationless P systems with membrane division for elementary and non-elementary membranes and dissolution rules in linear time is given.

In P systems with active membranes, the rules are non-cooperative, that is, the left-hand side of the rules have only one object. In [18] and [19], a cooperative version of object evolution rules was introduced. In following investigations, more restrictions were added to these rules, by considering *minimal cooperation* (the left-hand side of the rules have exactly two objects) and *minimal production* (the right-hand side of the rules have only one object) in objects evolution rules. Even restricting these rules to this, a linear-time solution to `SAT` was provided in [20] when division rules only for elementary membranes are considered and dissolution rules are forbidden.

3.1 Syntax

Definition 1. *A polarizationless P system with active membranes of degree $p \geq 1$ that makes use of minimal cooperation and minimal production in object evolution rules is a tuple $\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_p, \mathcal{R}, i_{out})$, where:*

- *Γ is a finite alphabet;*
- *H is a finite alphabet such that $H \cap \Gamma = \emptyset$;*
- *μ is a labelled rooted tree with p nodes;*
- *$\mathcal{M}_1, \ldots, \mathcal{M}_p$ are multisets over Γ;*
- *$\mathcal{R}$ is a finite set of rules, of the following forms:*
  - *(a) $[\, a \to c\,]_h$ or $[\, a\, b \to c\,]_h$, for $h \in H$, $a, b, c \in \Gamma$ (object evolution rules).*
  - *(b) $a\,[\quad]_h \to [\, b\,]_h$, for $h \in H$, $a, b \in \Gamma$ (send-in communication rules).*
  - *(c) $[\, a\,]_h \to b\,[\quad]_h$, for $h \in H$, $a, b \in \Gamma$ (send-out communication rules).*
  - *(d) $[\, a\,]_h \to b$, for $h \in H$, $a, b \in \Gamma$ (dissolution rules).*
  - *(e) $[\, a\,]_h \to [\, b\,]_h [\, c\,]_h$, for $h \in H$, $a, b, c \in \Gamma$ (division rules for elementary membranes).*
  - *(f) $[\,[\quad]_{h_0}\,[\quad]_{h_1}\,]_h \to [\,[\quad]_{h_0}\,]_h\,[\,[\quad]_{h_1}\,]_h$, for $h, h_0, h_1 \in H$ (division rules for non-elementary membranes).*

A polarizationless P system with active membranes of degree p can be viewed as a set of p membranes, labelled by elements of H, arranged in a hierarchical structure μ given by a rooted tree (called membrane structure) whose root is called the *skin membrane*, such that: (a) $\mathcal{M}_1, \ldots, \mathcal{M}_p$ represent the finite multisets of *objects* initially placed in the p 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 zone. We use the term *zone 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*. In these kind of P systems where are mechanisms, implemented by division rules, able to generate an exponential workspace (in terms of number of membranes and objects) in polynomial time. This allows us to describe brute force algorithms in these systems in an "efficient" way.

3.2 Semantics

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 $[\, a \to c\,]_h$ (resp., $[\, a\, b \to c\,]_h$) is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$ which contains object a (resp., objects a and b). When applying such a rule, object a (resp., objects a and b) is consumed and object c is produced in that membrane.

A send-in communication rule $a\,[\quad]_h \to [\, b\,]_h$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$ 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 labelled by h.

A send-out communication rule $[\, a\,]_h \to b\,[\quad]_h$ is *applicable* to a configuration $\mathcal{C}_t$ at an instant t, if there exists a membrane labelled by h in $\mathcal{C}_t$ 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 (in the case that such membrane is the skin then object b is produced in the environment).

A dissolution rule $[\, a\,]_h \to b$ 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 zone, such that it contains object a. When applying such a rule, object a is consumed, membrane h is dissolved and its objects beside an object b are sent to the parent (or the first ancestor that has not been dissolved).

A division rule $[\, a\,]_h \to [\, b\,]_h [\, c\,]_h$ for elementary membrane 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 zone, such that it contains object a. When applying such a rule, 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 $[\,[\quad]_{h_0}\,[\quad]_{h_1}\,]_h \to [\,[\quad]_{h_0}\,]_h\,[\,[\quad]_{h_1}\,]_h$ for non-elementary membrane 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 zone, which contains a membrane labelled by h_0 and another membrane labelled by h_1. When applying such a rule, 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 that 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.

In polarizationless P systems with active membranes, the rules are applied according to the following principles:

- At one transition step, one object and one membrane can be used by only one rule, selected in a non-deterministic way.
- At one transition step, a *membrane* can be the subject of *only one* rule of types $(b) - (f)$, and then it is applied at most once.
- Object evolution rules can be simultaneously applied to a membrane with one rule of types $(b) - (f)$. In any case, 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) or (f) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, changing the objects, and then the division is produced. Of course, this process takes only one transition step.
- The skin membrane and the output membrane, if any, can never get divided nor dissolved.

Let us notice that in these kind of P systems the environment plays a passive role in the following sense: along any computation, the environment only can receive objects from the system but it cannot send objects into the system.

4 Computing Membrane Systems

Let us recall that *counting membrane systems*, was introduced as a framework where counting problems (a special case of *search problems*) can be solved in a natural way [21]. These systems are inspired from *counting Turing machines* introduced by L. Valiant [23] and from recognizer membrane systems where the Boolean answer of these systems is replaced by an answer encoded by a natural number expressed in a binary representation (placed in the environment associated with the halting configuration). On the other hand, the concept of *computing P system* was introduced in [17] providing devices in Membrane Computing to compute *partial functions* from $\mathbb{N}^q$ to $\mathbb{N}^r$, with $q \geq 1, r \geq 1$.

Inspired by the previous concepts, (*binary*) *computing membrane systems* is defined in order to compute partial function from $\mathbb{N}^q$ to $\mathbb{N}^r$ $(q \geq 1, r \geq 1)$ when the natural numbers are considered by means of the corresponding binary representation.

Definition 2. *A (binary) computing membrane system Π of degree (p, q, r), $p \geq 1, q \geq 1, r \geq 1$, and order $n \geq 1$ is a tuple $\Pi = (\Pi', \Sigma, \Phi, i_{in})$, where*

- *$\Pi' = (\Gamma', \mu', \mathcal{M}'_1, \ldots, \mathcal{M}'_p, \mathcal{R}')$ is a membrane system with external output of degree p.*
- *$\Sigma = \{a_{1,0}, \ldots, a_{1,n-1}, \ldots, a_{q,0}, \ldots, a_{q,n-1}\}$ is an ordered set (the input alphabet) strictly contained in Γ'.*
- *$\Phi = \{b_{1,0}, \ldots, b_{1,n-1}, \ldots, b_{r,0}, \ldots, b_{r,n-1}\}$ is an ordered set (the final alphabet) strictly contained in Γ' and $\Phi \cap \Sigma = \emptyset$.*
- *i_{in} is the label of a distinguished membrane of Π' (the input membrane).*

Given a (binary) computing membrane system $\Pi = (\Pi', \Sigma, \Phi, i_{in})$ of degree (p, q, r) and order n, for each tuple $\boldsymbol{x} = (x_1, \ldots, x_q) \in \mathbb{N}^q$ such that $x_i \in [0, 2^n)$, for $1 \leq i \leq q$, there are uniques $x_{i,j} \in \{0, 1\}$, for $1 \leq i \leq q, 0 \leq j \leq n-1$, verifying $x_i = \sum_{j=0}^{n-1} x_{i,j} \cdot 2^j$, we use the following notations:

- $cod(\boldsymbol{x})$ is the set $\{a_{1,0}^{x_{1,0}}, \ldots, a_{1,n-1}^{x_{1,n-1}}, \ldots, a_{q,0}^{x_{q,0}}, \ldots, a_{q,n-1}^{x_{q,n-1}}\}$.
- $\Pi + cod(\boldsymbol{x})$ is the membrane system Π whose the initial configuration is the tuple $(\mu', \mathcal{M}'_1, \ldots, \mathcal{M}'_{i_{in}} + cod(\boldsymbol{x}), \ldots \mathcal{M}'_p)$.

In a (binary) computing membrane system Π of degree (p, q, r) and order n, the following semantics conditions are required: for each natural number $\boldsymbol{x} = (x_1, \ldots, x_q) \in \mathbb{N}^q$ such that $x_i \in [0, 2^n)$, for $1 \leq i \leq q$,

- Either any computation of $\Pi + cod(\boldsymbol{x})$ is a non-halting computation, or all computations of $\Pi + cod(\boldsymbol{x})$ halt.
- If all computations of $\Pi + cod(\boldsymbol{x})$ halt, then there exists a tuple

$$(y_{1,0}, \ldots, y_{1,n-1}, \ldots, y_{r,0}, \ldots, y_{r,n-1}) \in \{0, 1\}^{r \cdot n}$$

  such that for any computation of $\Pi + cod(\boldsymbol{x})$, the subset of the final alphabet Φ contained in the environment associated with the corresponding halting configuration is $\{b_{1,0}^{y_{1,0}}, \ldots, b_{1,n-1}^{y_{1,n-1}}, \ldots, b_{r,0}^{y_{r,0}}, \ldots, b_{r,n-1}^{y_{r,n-1}}\}$.

According with this, the output of the membrane system $\Pi + cod(\boldsymbol{x})$, in the case that all their computations halt, denoted by $Output(\Pi + cod(\boldsymbol{x}))$, is the tuple

$$(y_{1,0}, \ldots, y_{1,n-1}, \ldots, y_{r,0}, \ldots, y_{r,n-1}) \in \{0, 1\}^{r \cdot n}$$

That is, the output of the membrane system $\Pi + cod(\boldsymbol{x})$ encodes a tuple $(y_1, \ldots, y_r) \in \mathbb{N}^r$ such that $y_l \in [0, 2^n)$, for $1 \leq l \leq r$, and $(y_{l,0}, \ldots, y_{l,n-1})$ is the binary representation of y_l.

Definition 3. *We say that a (binary) computing membrane system Π of degree (p, q, r) and order n, computes a partial function f from $[0, 2^n) \times \overset{(q \cdot n)}{\cdots} \times [0, 2^n)$ into $[0, 2^n) \times \overset{(r \cdot n)}{\cdots} \times [0, 2^n)$, if for each $\boldsymbol{x} = (x_1 \ldots x_n) \in [0, 2^n) \times \overset{(q \cdot n)}{\cdots} \times [0, 2^n)$, the following holds:*

- *$f(\boldsymbol{x})$ is defined, that is, $\boldsymbol{x} \in dom(f)$, if and only if all computations of system $\Pi + cod(\boldsymbol{x})$ halt.*
- *$f(\boldsymbol{x}) = \boldsymbol{y}$, with $\boldsymbol{y} = \sum_{j=0}^{n-1} y_{i,j} \cdot 2^j$, for $1 \leq i \leq r$, if and only if*

$$Output(\Pi + cod(\boldsymbol{x})) = (y_{1,0}, \ldots, y_{1,n-1}, \ldots, y_{r,0}, \ldots, y_{r,n-1})$$

Definition 4. *We say that a family $\{\Pi(k) \mid k \in \mathbb{N}\}$ of (binary) computing membrane systems computes a partial function f from $\mathbb{N}^q$ into $\mathbb{N}^r$ if the following holds:*

- *For each $k \in \mathbb{N}$, $\Pi(k)$ is a (binary) computing membrane system of order $k + 1$.*
- *For each $\boldsymbol{x} \in \mathbb{N}^q$, $f(\boldsymbol{x})$ is defined and $f(\boldsymbol{x}) = \boldsymbol{y}$, with $\boldsymbol{y} = \sum_{j=0}^{k_{(\boldsymbol{x}, \boldsymbol{y})}} y_{i,j} \cdot 2^j$, for $1 \leq i \leq r$, if and only if the output of the system $\Pi(k_{(\boldsymbol{x}, \boldsymbol{y})}) + cod(\boldsymbol{x})$ is the tuple $(y_{1,0}, \ldots, y_{1,k_{(\boldsymbol{x}, \boldsymbol{y})}}, \ldots, y_{r,0}, \ldots, y_{r,k_{(\boldsymbol{x}, \boldsymbol{y})}})$.*

5 Solving the FACTORIZATION problem by Computing Membrane Systems

Let us recall that the FACTORIZATION problem is the following: *given a natural number which is the product of two prime numbers, find its decomposition.* This problem can be characterized by a *partial function* FACT from $\mathbb{N}$ to $\mathbb{N}^2$ defined as follows: for each natural number x which is the product of two prime numbers y, z, with $y \geq z$, we have $\texttt{FACT}(x) = (y, z)$. In other words, to solve the FACTORIZATION problem is equivalent to compute the partial function FACT.

In this paper, a solution to the FACTORIZATION problem is presented by providing a family $\{\Pi(n) \mid n \in \mathbb{N}\}$ of (binary) computing polarizationless P systems with active membranes which make use of minimal cooperation and minimal production (without dissolution rules and without division rules for non-elementary membranes), that computes the partial function FACT from $\mathbb{N}$ to $\mathbb{N}^2$, previously defined. Specifically, an instance x of the FACTORIZATION problem will be processed by the membrane system $\Pi(k_x)$ with input multiset $cod(x)$, where $cod(x)$ encodes the binary representation of instance x through the input alphabet of $\Pi(k_x)$. Bearing in mind that $2 \leq y, z < x$ we have $k_x = k_{(x,y,z)}$, and so $x, y, z \in [0, 2)^{1+k_x}$. Besides, $1+k_x$ is the maximum number of digits of natural numbers $x, y, z \geq 1$ in its binary representation and the system $\Pi(k_x)$ has order $k_x + 1$. For each natural number $n \in \mathbb{N}$, we consider the computing polarizationless P system with active membranes which makes use of minimal cooperation and minimal production but without dissolution rules and without division for non-elementary membranes, $\Pi(n) = (\Gamma, \Sigma, \Phi, H, \mu, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, i_{in}, i_{out})$ of degree $(2, 1, 2)$ and order $n + 1$, defined as follows:

(1) Working alphabet:

$$
\begin{aligned}
\Gamma = \Sigma \cup \Phi \cup \{\#\} \cup \{\omega_{j,k} \mid 0 \leq j \leq n, 0 \leq k \leq 17n + 26\} \cup \\
\{\alpha_{1,j,s} \mid 0 \leq j \leq n, 0 \leq s < j\} \cup \\
\{\alpha_{2,j,s} \mid 0 \leq j \leq n, 0 \leq s < n + 1 + j\} \cup \\
\{T_{h,j}, \overline{T}_{h,j} \mid 1 \leq h \leq 2, 0 \leq j \leq n\} \cup \\
\{p_{j,k} \mid 0 \leq j \leq n, 0 \leq k \leq 5n + 10\} \cup \\
\{\beta_{h,j,s} \mid 1 \leq h \leq 2, 0 \leq j \leq n, 0 \leq s \leq 3n + 6\} \cup \\
\{\overline{P}_{j,k} \mid 0 \leq j \leq n, 0 \leq k \leq 5n + 8\} \cup \{X_j, \overline{X}_j \mid 0 \leq j \leq n\} \cup \\
\{t_{1,j,s}, \bar{t}_{1,j,s} \mid 0 \leq j \leq n, j \leq s \leq 3n + 5\} \cup \\
\{t_{2,j,s}, \bar{t}_{2,j,s} \mid 0 \leq j \leq n, n + 1 + j \leq s \leq 3n + 5\} \cup \\
\{T^*_{h,j}, \overline{T}^*_{h,j} \mid 1 \leq h \leq 2, 0 \leq j \leq n\} \cup \\
\{P_j, \overline{P}_j, P^*_j, \overline{P}^*_j \mid 0 \leq j \leq n\} \cup \{e_j, e^*_j \mid 1 \leq j \leq n\} \cup \\
\{d_j \mid 1 \leq j \leq 4n + 3\} \cup \{d^*_j \mid 2n + 2 \leq j \leq 4n + 2\} \cup \\
\{G_k \mid 1 \leq k \leq 8n + 6\} \cup \{T'_{1,j}, \overline{T}'_{1,j} \mid 0 \leq j \leq n\} \cup \\
\{C_{h,j,i} \mid 0 \leq h \leq 2, 0 \leq j \leq n, 0 \leq i \leq n - j\} \cup \\
\{C_{h,j} \mid 0 \leq h \leq 2, 0 \leq j \leq n\} \cup \\
\{T_{1,j,k}, \overline{T}_{1,j,k} \mid 0 \leq j \leq n, 0 \leq k \leq j\} \cup \\
\{T_{2,j,k}, \overline{T}_{2,j,k} \mid 0 \leq j \leq n, 0 \leq k \leq n + 1 + j\} \cup \\
\{y_j, \overline{y}_j, z_j, \overline{z}_j, y^*_j, z^*_j \mid 0 \leq j \leq n\}
\end{aligned}
$$

where the input alphabet is $\Sigma = \{a_i \mid 0 \leq i \leq n\}$ and the final alphabet is $\Phi = \{b_{1,j}, b_{2,j} \mid 0 \leq j \leq n\}$;

(2) $H = \{1, 2\}$

(3) Membrane structure: $\mu = [\,[\quad]_2\,]_1$, that is, $\mu = (V, E)$ where $V = \{1, 2\}$ and $E = \{(1, 2)\}$

(4) Initial multisets: $\mathcal{M}_1 = \{\omega^2_{j,0} \mid 0 \leq j \leq n\}$, $\mathcal{M}_2 = \{\overline{X}_j, \alpha_{1,j,0}, \alpha_{2,j,0}, T^{n+4}_{1,j}, \overline{T}^{n+4}_{1,j}, T^{n+4}_{2,j}, \overline{T}^{n+4}_{2,j}, p_{j,0}, \omega^2_{j,0}, \tau_{j,0}, \overline{z}_j, \beta^{n+1}_{1,j,0}, \beta^{n+1}_{2,j,0} \mid 0 \leq j \leq n\} \cup \{\overline{P}_{j,0} \mid 0 \leq i \leq 2n+1\}$

(5) The set of rules $\mathcal{R}$ consists of the following rules:

5.1 Counters

$$[\,a_j\,\overline{X}_j \to X_j\,]\ \text{, for } 0 \leq j \leq n$$
$$[\,\alpha_{1,j,s} \to \alpha_{1,j,s+1}\,]_2\ \text{, for } 0 \leq j \leq n \text{ and } 0 \leq s < j$$
$$[\,\alpha_{2,j,s} \to \alpha_{2,j,s+1}\,]_2\ \text{, for } 0 \leq j \leq n \text{ and } 0 \leq s < n+1+j$$
$$\left.\begin{array}{l}[\,\beta_{1,j,k} \to \beta_{1,j,k+1}\,]_2\\ [\,\beta_{2,j,k} \to \beta_{2,j,k+1}\,]_2\end{array}\right\} \text{for } 0 \leq j \leq n, 0 \leq k \leq 3n+5$$
$$[\,\overline{P}_{j,k} \to \overline{P}_{j,k+1}\,]_2\ \text{, for } 0 \leq j \leq 2n+1, 0 \leq k \leq 5n+7$$
$$[\,p_{i,j} \to p_{i,j+1}\,]_2\ \text{, for } 0 \leq i \leq n, 0 \leq j \leq 5n+9$$
$$[\,\tau_{j,k} \to \tau_{j,k+1}\,]_2\ \text{, for } 0 \leq j \leq n, 0 \leq k \leq 14n+11$$
$$[\,\omega_{j,k} \to \omega_{j,k+1}\,]_2\ \text{, for } 0 \leq j \leq n, 0 \leq k \leq 15n+21$$
$$[\,\omega_{i,j} \to \omega_{i,j+1}\,]_1\ \text{, for } 0 \leq i \leq n, 0 \leq j \leq 17n+25$$

5.2 Generation Stage

$$\left.\begin{array}{l}[\,\alpha_{1,j,j}\,]_2 \to [\,t_{1,j,j}\,]_2\,[\,\bar{t}_{1,j,j}\,]_2\\ [\,\alpha_{2,j,n+1+j}\,]_2 \to [\,t_{2,j,n+1+j}\,]_2\,[\,\bar{t}_{2,j,n+1+j}\,]_2\end{array}\right\} \text{for } 0 \leq j \leq n$$
$$\left.\begin{array}{l}[\,t_{1,j,v} \to t_{1,j,v+1}\,]_2\\ [\,\bar{t}_{1,j,v} \to \bar{t}_{1,j,v+1}\,]_2\end{array}\right\} \text{for } 0 \leq j \leq n \text{ and } j \leq v \leq 2n$$
$$\left.\begin{array}{l}[\,t_{2,j,v} \to t_{2,j,v+1}\,]_2\\ [\,\bar{t}_{2,j,v} \to \bar{t}_{2,j,v+1}\,]_2\end{array}\right\} \text{for } 0 \leq j \leq n-1 \text{ and } n+1+j \leq v \leq 2n$$
$$\left.\begin{array}{l}[\,t_{h,j,2n+s}\,\overline{T}_{h,j} \to t_{h,j,2n+s+1}\,]_2\\ [\,\bar{t}_{h,j,2n+s}\,T_{h,j} \to \bar{t}_{h,j,2n+s+1}\,]_2\end{array}\right\} \text{for } \begin{array}{c}1 \leq h \leq 2\\ 0 \leq j \leq n\\ 1 \leq s \leq 3n+4\end{array}$$
$$\left.\begin{array}{l}[\,t_{h,j,3n+5}\,\overline{T}_{h,j} \to \#\,]_2\\ [\,\bar{t}_{h,j,3n+5}\,T_{h,j} \to \#\,]_2\\ [\,\beta_{1,j,3n+6}\,T_{1,j} \to T^*_{1,j}\,]_2\\ [\,\beta_{1,j,3n+6}\,\overline{T}_{1,j} \to \overline{T}^*_{1,j}\,]_2\\ [\,\beta_{2,j,3n+6}\,T_{2,j} \to T^*_{2,j}\,]_2\\ [\,\beta_{2,j,3n+6}\,\overline{T}_{2,j} \to \overline{T}^*_{2,j}\,]_2\end{array}\right\} \text{for } 0 \leq j \leq n, 0 \leq k \leq 3n+5$$

5.3 Multiplication Stage

$$\left.\begin{array}{l}[\,T^*_{1,j}\,T^*_{2,j'} \to P_{j+j'}\,]_2\\ [\,T^*_{1,j}\,\overline{T}^*_{2,j'} \to P_j\,]_2\\ [\,\overline{T}^*_{1,j}\,T^*_{2,j'} \to P_{j'}\,]_2\\ [\,\overline{T}^*_{1,j}\,\overline{T}^*_{2,j'} \to \#\,]_2\end{array}\right\} \text{for } 0 \leq j, j' \leq n$$

$$\left.\begin{array}{l}[\,\overline{P}_j\,P_j \to P_{j+1}\,]_2\\ [\,\overline{P}_{j,5n+8} \to \overline{P}_j\,]_2\\ [\,P_j\overline{P}_j \to P_j\,]_2\\ [\,p_{j,5n+10}\,P_j \to P_j^*\,]_2\\ [\,p_{j,5n+10}\,\overline{P}_j \to \overline{P}_j^*\,]_2\end{array}\right\} \text{for } 0 \le j \le 2n+1$$

5.4 Equality Checking Stage

$$\left.\begin{array}{l}[\,P_j^*\,X_j \to e_j\,]_2\\ [\,\overline{P}_j^*\,\overline{X}_j \to e_j\,]_2\\ [\,\overline{P}_j^*\,X_j \to e_j^*\,]_2\\ [\,P_j^*\,\overline{X}_j \to e_j^*\,]_2\end{array}\right\} \text{for } 0 \le j \le n$$

$$[\,e_0\,e_1 \to d_1]_2$$

$$[\,d_j\,e_{j+1} \to d_{j+1}\,]_2 \text{ , for } 0 \le j \le n-1$$

$$[\,e_0^*\,e_1 \to G_1\,]_2$$

$$[\,e_0\,e_1^* \to G_1\,]_2$$

$$[\,e_0^*\,e_1^* \to G_1\,]_2$$

$$\left.\begin{array}{l}[\,d_j\,e_{j+1}^* \to G_{j+1}\,]_2\\ [\,G_j\,e_{j+1} \to G_{j+1}\,]_2\\ [\,G_j\,e_{j+1}^* \to G_{j+1}\,]_2\end{array}\right\} \text{for } 0 \le j \le n$$

$$\left.\begin{array}{l}[\,d_j\,\overline{P}_{j+1} \to d_{j+1}]_2\\ [\,d_j\,P_{j+1} \to G_{j+1}\,]_2\\ [\,G_j\,P_{j+1} \to G_{j+1}\,]_2\\ [\,G_j\,\overline{P}_{j+1} \to G_{j+1}\,]_2\end{array}\right\} \text{for } n \le j \le 2n$$

$$\left.\begin{array}{l}[\,G_{2n+1+j}\,T_{1,j} \to G_{2n+2+j}\,]_2\\ [\,G_{2n+1+j}\,\overline{T}_{1,j} \to G_{2n+2+j}\,]_2\\ [\,G_{3n+2+j}\,T_{2,j} \to G_{3n+3+j}\,]_2\\ [\,G_{3n+2+j}\,\overline{T}_{2,j} \to G_{3n+3+j}\,]_2\end{array}\right\} \text{for } 0 \le j \le n$$

5.5 Trivial Solution Check Stage

$$[\,d_{2n+1}\,T_{1,0} \to d_{2n+2}\,]_2$$

$$[\,d_{2n+1}\,\overline{T}_{1,0} \to d_{2n+2}^*\,]_2$$

$$\left.\begin{array}{l}[\,d_{2n+2+j}\,\overline{T}_{1,j+1} \to d_{2n+3+j}\,]_2\\ [\,d_{2n+2+j}\,T_{1,j+1} \to d_{2n+3+j}^*\,]_2\\ [\,d_{2n+2+j}^*\,T_{1,j+1} \to d_{2n+3+j}^*\,]_2\\ [\,d_{2n+2+j}^*\,\overline{T}_{1,j+1} \to d_{2n+3+j}^*\,]_2\end{array}\right\} \text{for } 0 \le j \le n-2$$

$$[\,d_{3n+1}\,\overline{T}_{1,n} \to T_{3n-1}\,]_2$$

$$[\,d_{3n+1}\,T_{1,n} \to d_{3n-1}\,]_2$$

$$[\,d_{3n+1}^*\,T_{1,n} \to d_{3n+2}\,]_2$$

$$[\,d_{3n+1}^*\,\overline{T}_{1,n} \to d_{3n+2}\,]_2$$

$$[\,d_{3n+2}\,T_{2,0} \to d_{3n}\,]_2$$

$$[\,d_{3n+2}\,\overline{T}_{2,0} \to d_{3n}^*\,]_2$$

$$\left.\begin{array}{l}[\,d_{3n+3+j}\,\overline{T}_{2,j+1} \to d_{3n+4+j}\,]_2\\ [\,d_{3n+3+j}\,T_{2,j+1} \to d_{3n+4+j}^*\,]_2\\ [\,d_{3n+3+j}^*\,T_{2,j+1} \to d_{3n+4+j}^*\,]_2\\ [\,d_{3n+3+j}^*\,\overline{T}_{2,j+1} \to d_{3n+4+j}^*\,]_2\end{array}\right\} \text{for } 0 \le j \le n-2$$

$$[\, d_{4n+2}\, \overline{T}_{2,n} \to T_{4n+3}\,]_2$$

$$[\, d_{4n+2}\, T_{2,n} \to d_{4n+3}\,]_2$$

$$[\, d^*_{4n+2}\, T_{2,n} \to d_{4n+3}\,]_2$$
$$[\, d^*_{4n+2}\, \overline{T}_{2,n} \to d_{4n+3}\,]_2$$

5.6 First Delete Stage

$$\left.\begin{array}{l} [\, G_{4n+3+j}\, T_{1,j} \to G_{4n+4+j}\,]_2 \\ [\, G_{4n+3+j}\, \overline{T}_{1,j} \to G_{4n+4+j}\,]_2 \\ [\, G_{5n+4+j}\, T_{2,j} \to G_{5n+5+j}\,]_2 \\ [\, G_{5n+4+j}\, \overline{T}_{2,j} \to G_{5n+5+j}\,]_2 \\ [\, G_{6n+5+j}\, T_{1,j} \to G_{6n+6+j}\,]_2 \\ [\, G_{6n+5+j}\, \overline{T}_{1,j} \to G_{6n+6+j}\,]_2 \end{array}\right\} \text{for } 0 \leq j \leq n$$

$$\left.\begin{array}{l} [\, G_{7n+6+j}\, T_{2,j} \to G_{7n+7+j}\,]_2 \\ [\, G_{7n+6+j}\, \overline{T}_{2,j} \to G_{7n+7+j}\,]_2 \end{array}\right\} \text{for } 0 \leq j \leq n-1$$

$$[\, G_{8n+6}\, T_{2,n} \to \#\,]_2$$
$$[\, G_{8n+6}\, \overline{T}_{2,n} \to \#\,]_2$$

5.7 Second Delete Stage

$$\left.\begin{array}{l} [\, \tau_{j,14n+12}\, T_{1,j} \to T'_{1,j}\,]_2 \\ [\, \tau_{j,14n+12}\, \overline{T}_{1,j} \to \overline{T}'_{1,j}\,]_2 \end{array}\right\} \text{for } 0 \leq j \leq n$$

$$\left.\begin{array}{l} [\, \overline{T}'_{1,j}\, T_{2,j} \to C_{2,j,n-j}\,]_2 \\ [\, \overline{T}'_{1,i}\, \overline{T}_{2,j} \to C_{1,j,n-j}\,]_2 \\ [\, T'_{1,j}\, T_{2,j} \to C_{1,j,n-j}\,]_2 \\ [\, T'_{1,j}\, \overline{T}_{2,j} \to C_{0,j,n-j}\,]_2 \end{array}\right\} \text{for } 0 \leq j \leq n$$

$$\left.\begin{array}{l} [\, C_{1,j,0}\, C_{2,j-1,1} \to C_{2,j-1,0}\,]_2 \\ [\, C_{0,j,0}\, C_{2,j-1,1} \to C_{0,j-1,0}\,]_2 \\ [\, C_{1,j,0}\, C_{1,j-1,1} \to C_{1,j-1,0}\,]_2 \end{array}\right\} \text{for } 2 \leq j \leq n$$

$$\left.\begin{array}{l} [\, C_{2,j,0}\, C_{i,j-1,1} \to C_{2,j-1,0}\,]_2 \\ [\, C_{0,j,0}\, C_{i,j-1,1} \to C_{0,j-1,0}\,]_2 \end{array}\right\} \text{for } 0 \leq i \leq 2, 2 \leq j \leq n$$

$$[\, C_{1,1,0}\, C_{2,0,1} \to C_{2,0}\,]_2$$

$$[\, C_{1,1,0}\, C_{0,0,1} \to C_{0,0}\,]_2$$

$$[\, C_{1,1,0}\, C_{1,0,1} \to C_{1,0}\,]_2$$

$$\left.\begin{array}{l} [\, C_{2,1,0}\, C_{j,0,1} \to C_{2,0}\,]_2 \\ [\, C_{0,1,0}\, C_{j,0,1} \to C_{0,0}\,]_2 \end{array}\right\} \text{for } 0 \leq j \leq 2$$

$$[\, C_{i,j,k} \to C_{i,j,k-1}\,]_2 \text{ , for } 0 \leq i \leq 2, 2 \leq j \leq n, 0 \leq k \leq n$$

5.8 Output 1 Phase

$$\left.\begin{array}{l}
[\, C_{2,j}\, T_{1,j} \to C_{2,j+1}\,]_2 \\
[\, C_{2,j}\, \overline{T}_{1,j} \to C_{2,j+1}\,]_2 \\
[\, C_{2,n+1+j}\, T_{2,j} \to C_{2,n+2+j}\,]_2 \\
[\, C_{2,n+1+j}\, \overline{T}_{2,j} \to C_{2,n+2+j}\,]_2 \\
[\, \omega_{j,15n+22}\, T_{1,j} \to T_{1,j,j}\,]_2 \\
[\, \omega_{j,15n+22}\, \overline{T}_{1,j} \to \overline{T}_{1,j,j}\,]_2 \\
[\, \omega_{j,15n+22}\, T_{2,j} \to T_{2,j,n+1+j}\,]_2 \\
[\, \omega_{j,15n+22}\, \overline{T}_{2,j} \to \overline{T}_{2,j,n+1+j}\,]_2 \\
[\, T_{1,j,0}\,]_2 \to y_j\, [\quad]_2 \\
[\, \overline{T}_{1,j,0}\,]_2 \to \overline{y}_j\, [\quad]_2 \\
[\, T_{2,j,0}\,]_2 \to z_j\, [\quad]_2 \\
[\, \overline{T}_{2,j,0}\,]_2 \to \overline{z}_j\, [\quad]_2
\end{array}\right\} \text{for } 0 \leq j \leq n$$

$$\left.\begin{array}{l}
[\, T_{1,j,k} \to T_{1,j,k-1}\,]_2 \\
[\, \overline{T}_{1,j,k} \to \overline{T}_{1,j,k-1}\,]_2
\end{array}\right\} \text{for } 0 \leq j \leq n, 1 \leq k \leq n$$

$$\left.\begin{array}{l}
[\, T_{2,j,k} \to T_{2,j,k-1}\,]_2 \\
[\, \overline{T}_{2,j,k} \to \overline{T}_{2,j,k-1}\,]_2
\end{array}\right\} \text{for } 0 \leq j \leq n, 1 \leq k \leq 2n+1$$

5.10 Output 2 Phase

$$\left.\begin{array}{l}
[\, y_j\, y_j \to y_j\,]_1 \\
[\, \overline{y}_j\, \overline{y}_j \to \overline{y}_j\,]_1 \\
[\, z_j\, z_j \to z_j\,]_1 \\
[\, \overline{z}_j\, \overline{z}_j \to \overline{z}_j\,]_1 \\
[\, \omega_{i,17n+26}\, y_j \to y_j^*\,]_1 \\
[\, \omega_{i,17n+26}\, z_j \to z_j^*\,]_1 \\
[\, y_j^*\,]_1 \to b_{1,j}\, [\quad]_1 \\
[\, z_j^*\,]_1 \to b_{2,j}\, [\quad]_1
\end{array}\right\} \text{for } 0 \leq j \leq n$$

(6) The input membrane is the membrane labelled by 1 ($i_{in} = 2$) and the output region is the environment ($i_{out} = env$).

6 An Overview of the Computations

Let $x \in \mathbb{N}$ an instance of the `FACTORIZATION` problem, that is, x is a natural number whose binary representation is given by $(x_0, \ldots, x_n)$, and such that $x = y \cdot z$ being y and z prime numbers with $y \geq z$. Then, x will be processed by the membrane system $\Pi(k_x) + cod(x)$, where $cod(x) = \{a_0^{x_0}, \ldots, a_n^{x_n}\}$.

The family $\{\Pi(n) \mid n \in \mathbb{N}\}$ designed to solve the `FACTORIZATION` problem captures the behaviour of a brute force algorithm: (a) all possible pairs of natural numbers y, z, with $y, z \leq x$ are produced; (b) the product $y \cdot z$ is computed; and (c) the output is the pair $\{y, z\}$ if and only if $x = y \cdot z$. Next, we briefly describe the stages in which the computations of membrane system $\Pi(n)$ are structured, where $n = k_x$, being x an instance of the `FACTORIZATION` problem.

6.1 Generation Stage

At this stage, 2^{2n+2} membranes labelled by 2 are generated in such manner that each of them contains $n + 4$ copies of possible candidate pairs of natural

numbers y, z, whose binary representation have at most $n+1$ digits, which will be represented by symbols $T^*_{h,j}$ and $\overline{T}^*_{h,j}$. For that, first of all, the code $cod(x)$ of the instance $x = (x_0, \ldots, x_n)$ changes to $\{\rho_i \mid 0 \leq i \leq n\}$, where $\rho_j = X_j$ if $x_j = 1$, $\rho_j = \overline{X}_j$ if $x_j = 0$. From the beginning, division rules to objects $\alpha_{1,j,j}$ and $\alpha_{2,j,n+1+j}$ are applied. Second, objects $t_{h,j,v}$ and $\bar{t}_{h,j,v}$ are used in order to remove undesired objects $T_{h,j}$ and $\overline{T}_{h,j}$. Finally, objects $\beta_{h,j,3n+6}$ will produce objects $T^*_{h,j}$ and $\overline{T}^*_{h,j}$ encoding all possible different candidates y, z of natural numbers in each membrane labelled by 2. This stage takes $3n+7$ steps.

6.2 Multiplication Stage

At this stage, the pair of natural numbers encoded in each membrane labelled by 2, is multiplied. For that, first all bits represented by objects $T^*_{1,j}$ or $\overline{T}^*_{1,j}$ are multiplied with all bits represented by objects $T^*_{2,j'}$ or $\overline{T}^*_{2,j'}$, and objects $P_{j+j'}$ are produced. Second, objects $P_{j+j'}$ are handled in order to be sure that there is, at most, one bit for each position. In order to have a complete binary representation of these numbers, that is, the representation of each bit of the product, we use object P_j to represent that the bit j equals 1, and object $\overline{P}_j$ if bit j equals 0. This stage takes $2n+4$ steps.

6.3 Equality Checking Stage

Here, in each membrane labelled by 2, the instance x encoded by objects X_j and $\overline{X}_j$ is compared to the product $y \cdot z$, represented by objects P_j and $\overline{P}_j$. If they are equal, that is, $y \cdot z = x$, then objects encoding y, z remain in that membrane, and they are removed otherwise. First, objects X_j and $\overline{X}_j$ are compared with objects P_j and $\overline{P}_j$. Next, these partial comparisons represented by objects e_j and e^*_j are used to compare the entire number. If some object $T_{h,j}$, with $j \geq n+1$, appears, that is, the binary representation of the product has more "useful" bits than the original number, then all the objects are erased. This stage takes $4n+4$ steps.

6.4 Trivial Solution Check Stage

Next, solutions y, z with either $y = 1$ or $z = 1$ (*trivial* solutions) are removed. For that, bits are checked to be sure that the two numbers are different from 1, and remove them otherwise. This stage takes $2n+2$ steps.

6.5 First Delete Stage

In order to remove remaining objects from a membrane, a garbage recollection strategy is used, so if an object G_{4n+3} appears in a membrane, then all objects in such a membrane are removed. This stage takes $4n+4$ steps.

6.6 Second Delete Stage

If $y \cdot z = x$ and $y \neq z$ then we have two membranes labelled by 2 such that objects $T_{1,j}$ $\overline{T}_{1,j}$ encode y and objects $T_{2,j}$ $\overline{T}_{2,j}$ encode z, but one of them represents that $y > z$ and the other one represents that $y < z$. In this situation, membrane containing objects encoding $y > z$ is distinguished and the corresponding objects of the other membrane are removed. In the case $y = z$, objects encoding these natural numbers will be kept in both membranes. For that, objects $C_{r,j,k}$ will be produced. If the j-th bit of number y will be smaller than the j-th bit of z, then $r = 2$, on the contrary $r = 0$. If j-th bit of both y and z are the same one then $r = 1$. Later, these objects are used to compare the entire numbers. This stage takes $n + 2$ steps.

6.7 Output 1 Stage

In this stage, objects representing numbers y and z are going to be sent out to membrane 1. To make this stage deterministic, first objects $T_{1,j}$ and $\overline{T}_{1,j}$ and second objects $T_{2,j}$ and $\overline{T}_{2,j}$ are released to membrane labelled by 1. At the end of the stage, objects y_j and $\overline{y}_j$ represent $T_{1,j}$ and $\overline{T}_{1,j}$ in membrane 1. Similarly, objects z_j and $\overline{z}_j$ represent $T_{2,j}$ and $\overline{T}_{2,j}$ in membrane 1. This stage takes $4n+5$ steps.

6.8 Output 2 Stage

Finally, binary representations of the numbers y and z are going to be sent out to the environment by using objects of the final alphabet. First, the perfect square case (two copies of objects y_j or $\overline{y}_j$ and two copies of objects z_j or $\overline{z}_j$ appear) has to be taken into account. For that, two objects y_j (or $\overline{y}_j$) will produce only one object y_j (or $\overline{y}_j$), and similarly for objects z_j and $\overline{z}_j$. Next, each object y_j (resp., z_j) will produce an object y_j^* (resp., z_j^*) cooperating with object ω_{17n+26}. Finally, each object y_j^* produce an object $b_{1,j}$ at the environment, and each object z_j^* produce an object $b_{2,j}$ at the environment. This stages takes at most $2n + 3$ steps.

At Table 1, the steps used by each stage, besides the initial and final configuration of each one are indicated.

7 Conclusions and Future Work

The `FACTORIZATION` problem (*given a natural number n which is product of two large primes, find its decomposition*) can be characterized by a *partial function* `FACT` from $\mathbb{N}$ to $\mathbb{N}^2$ defined as follows: for each natural number x which is the product of two prime numbers y, z, with $y \geq z$, we have $\texttt{FACT}(x) = (y, z)$. This problem belongs to the class **FNP** and it is conjectured that it is an intractable problem, assuming that $\mathbf{P} \neq \mathbf{NP}$. Besides, it is the basis for some cryptographic systems as important as RSA, a *de facto* standard for digital signatures. In order

Stage	Steps	Initial configuration	Final configuration
Generation	$3n+7$	0	$3n+7$
Multiplication	$2n+4$	$3n+7$	$5n+11$
Equality checking stage	$4n+4$	$5n+11$	$9n+15$
Trivial solution check	$2n+2$	$7n+13$	$9n+15$
First delete	$4n+4$	$9n+15$	$13n+19$
Second delete	$n+2$	$12n+18$	$13n+20$
Output 1	$4n+5$	$13n+20$	$17n+25$
Output 2	$\leq 2n+3$	$17n+25$	$\leq 19n+28$

Table 1. Number of steps by each stage

to provide solutions in the framework of Membrane Computing, new membrane systems computing partial functions between natural numbers are introduced.

In this work, a linear time solution to the `FACTORIZATION` problem is presented by means of a family of polarizationless P system with active membranes without dissolution rules which use minimal cooperation and minimal production in object evolution rules. This solution improves the previous ones given in the membrane computing framework, in the sense that the use of syntactical ingredients is significantly lower.

P-Lingua [25] and MeCoSim [24] have been very useful as assistant tools for the process of verifying this design. An interesting future work is to use this model in a GPU-based simulator, since it can accelerate the processing of the computation. Several simulators of P systems have been implemented using the NVIDIA CUDA framework. In fact, in the PMCGPU project [26] we can see some simulators for different types of P systems. Some stages could be optimized in order to have faster communications between the multiple cores of the graphic card, like the encoding of objects into integers or the omission of some objects that only act to synchronize the system. Another way to speed up the algorithm would be to omit all the membranes containing an element $c_{1,i}$, because we know that these bits equal zero in our initial number x.

8 Acknowledgements

This work was supported by Project TIN2017-89842-P of the Ministerio de Economía y Competitividad of Spain and by Grant No 61320106005 of the National Natural Science Foundation of China.

References

1. T.H. Cormen, C.E. Leiserson, R.L. Rivest: An Introduction to Algorithms. *The MIT Press*, Cambridge, Massachussets, 1994.
2. D. Díaz-Pernil, H.A. Christinal, M.Á. Gutiérrez-Naranjo: Solving the 3-COL Problem by Using Tissue P Systems without Environment and Proteins on Cells. In: *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, 1-5 February, 2016, Sevilla, Spain, Fénix Editora, 163–172.

3. W. Diffie, M. Hellman: New directions in cryptography. *IEEE Transactions on Information Theory* **22**, 6, 644–654.
4. 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.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking Neural P Systems. *Fundamenta Informaticae* **71**, 2,3 (February 2006), 279–308.
6. A. Leporati, C. Zandron, G. Mauri: Solving the factorization problem with P systems. *Progress in Natural Science* **17**, 4 (2007), 471–478.
7. C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón: Tissue P systems. *Theoretical Computer Science* **296**, 2 (2003), 295–326.
8. R. Merkle, M. Hellman: Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory* **24**, 5, 525–530.
9. T. Murakawa, A. Fujiwara: Operations and Factorization using Asynchronous P systems. *International Journal of Networking and Computing* **2**, 2 (2012), 217–233.
10. Gh. Păun. Computing with membranes. *Journal of Computer and Systems Science* **61**, 1 (2000), 108–143.
11. Gh. Păun: P systems with active membranes: Attacking **NP**–complete problems. *Journal of Automata, Languages and Combinatorics* **6** (2001), 75–90. A preliminary version appeared in *Centre for Discrete Mathematics and Theoretical Computer Science Research Reports Series*, CDMTCS-102, May 1999.
12. M.J. Pérez-Jiménez, A. Riscos-Núñez, M. Rius-Font, L. Valencia-Cabrera: The relevance of the environment on the efficiency of tissue P systems. *Lecture Notes in Computer Science* **8340** (2014), 308–321.
13. M.J. Pérez-Jiménez, Á. Romero-Jiménez, F. Sancho-Caparrini: Complexity classes in models of cellular computing with membranes. *Natural Computing* **2**, 3 (2003), 265–285.
14. M. Sipser: Introduction to the Theory of Computation. *International Thomson Publishing* (1996).
15. A. Riscos-Núñez: Programación celular: Resolución eficiente de problemas numéricos NP-completos. PhD. Thesis, University of Seville, Spain, 2003.
16. R.L. Rivest, A. Shamir, L. Adleman: A method for obtaining digital signatures and public-key cryptosystems. *CAMC* **21**, 2 (1978), 120–126.
17. Á. Romero-Jiménez, M.J. Pérez-Jiménez: Generation of Diophantine Sets by Computing P Systems with External Output. *Lecture Notes in Computer Science* **2509** (2002), 176–190.
18. L. Valencia-Cabrera, D. Orellana-Martín, A. Riscos-Núñez, M.J. Pérez-Jiménez: Minimal cooperation in polarizationless P systems with active membranes. In C. Graciani, Gh. Păun, D. Orellana-Martn, A. Riscos-Nez, L. Valencia-Cabrera (eds.): *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, 1-5 February, 2016, Sevilla, Spain, Fénix Editora, 327–356.
19. L. Valencia-Cabrera, D. Orellana-Martín, M.Á. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez: Polarizationless P systems with active membranes: Computational complexity aspects. *Journal of Automata, Languages and Combinatorics* **21**, 1-2 (2016), 101–117.
20. L. Valencia-Cabrera, D. Orellana, M.A. Martínez-del-Amor, A. Riscos, M.J. Pérez-Jiménez: Reaching efficiency through collaboration in membrane systems: Dissolution, polarization and cooperation. *Theoretical Computer Science* **701** (2017), 226–234.
21. L. Valencia-Cabrera, D. Orellana, A. Riscos, M.J. Pérez-Jiménez: Counting membrane systems. *Lecture Notes in Computer Science* **10725** (2017), 74–87.

22. X. Zang, Y. Niu, L. Pan, M.J. Pérez-Jiménez: Linear Time Solution to Prime Factorization by Tissue P Systems with Cell Division. *Proceedings of the Ninth Brainstorming Week on Membrane Computing*, 2011, 355–372.
23. L.G. Valiant: The complexity of computing the permanent. *Theoretical Computer Science* **8**, 2 (1979), 189–201.
24. MeCoSim Website: `http://www.p-lingua.org/mecosim/`
25. P-Lingua Website: `http://www.p-lingua.org/wiki/index.php/Main_Page`
26. PMCGPU Website: `https://sourceforge.net/projects/pmcgpu/`

A Note on Polymorphic P Systems

Sergiu Ivanov

IBISC, Université Évry, Université Paris-Saclay
23 Boulevard de France, 91025, Évry, France
sergiu.ivanov@univ-evry.fr

Abstract. Polymorphic P systems are a variant of P systems – a multiset-rewriting-based model of computing inspired by the structure and the functioning of the living cell. In polymorphic P systems, rules are not statically defined, but instead are dynamically inferred from the contents of specially designated pairs of membranes. Besides enabling dynamic modifications of the form of the available rules, polymorphism allows for embedding rules into the left-hand and right-hand sides of other rules. The present extended abstract recalls the definition of the model and reiterates the open questions from the survey paper [1].

1 Introduction

Membrane computing is a research field originally founded by Gheorghe Păun in 1998 [6]. Membrane computing focuses on membrane systems (also known as P systems) which is a model of computing based on the abstract notion of a membrane. Formally, a membrane is treated as a container delimiting a region; a region may contain objects which are acted upon by the rewriting rules associated with the membrane. A comprehensive overview of different flavours of membrane systems and their expressive power is given in the 2010 handbook [7]. For a state of the art snapshot of the domain, we refer the reader to the P systems website [9], as well as to the bulletin of the International Membrane Computing Society [8].

As indicated by its name, membrane computing draws inspiration from the structure and functioning of the living cell [5]. Indeed, one can see the cell as a hierarchical arrangement of containers (rooted at the cellular wall) performing biochemical processing. Although computer science and cell biology are arguably distinct domains, they do have one feature in common: the description of the "program" can be modified by the organism itself. In cells, this paradigm (sometimes referred to as "program is data") is represented by mechanisms such as reverse transcription [3], while in computer science this is embodied by the fact that the program is stored in the memory alongside the data it manipulates.

Polymorphic P systems, originally introduced in [2], are another implementation of the "program is data" paradigm for membrane systems which does not limit the set of available rules by a finite cardinality and which allows direct tampering with the form of the rules. In polymorphic P systems, rules are not

statically defined, but are instead dynamically inferred from the contents of specially designated pairs of membranes. One member of such a pair defines the multiset representing the left-hand side of the rule; the other member defines the right-hand side.

The present note is an extended abstract of the survey paper [1], which gives an in-depth overview of the results known about polymorphic P systems. This note first recalls some basic definitions related to polymorphic P systems (Section 2), then shows the classic example of superexponential growth (Section 3), and ends by listing some potentially interesting open questions (Section 4).

2 Preliminaries

We assume the reader is familiar with the terms and concepts frequently used in membrane computing and, more generally, in the theory of formal languages. For an introduction, we refer the reader to [6, 7], as well as to the survey [1].

A *polymorphic P system* is a construct

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

where O is a finite alphabet of objects, T is the subalphabet of terminal objects, μ is a tree structure consisting of $2n+1$ membranes, w_s is the multiset giving the contents of the skin membrane, $\langle w_{iL}, w_{iR}\rangle$ are pairs of multisets giving the contents of membranes iL and iR ($1 \leq i \leq n$), and h_i and h_o are the labels of the input and the output membranes, respectively, with $h_i \in H$ and $h_o \in H \cup \{0\}$, where 0 denotes the environment. We require that, for every $1 \leq i \leq n$, the membranes iL and iR have the same containing (parent) membrane. The *depth* of (the membrane structure of) Π is defined as for conventional P systems: it is the height of μ seen as a tree.

The rules of Π are not statically given in its description and are instead dynamically inferred for each configuration based on the contents of the pairs of membranes iL and iR. Thus, if in a configuration of the system these membranes contain the multisets u and v, respectively, then, in the next step, their parent membrane h will evolve as if it had the multiset rewriting rule $u \to v$ associated with it. If, however, in some configuration, iL is empty, we consider the rule defined by the pair $\langle iL, iR\rangle$ to be *disabled*, i.e., no rule will be inferred from the contents of iL and iR.

Polymorphic P systems evolve by applying the dynamically inferred rules in a maximally parallel way. A *computation* of a polymorphic P system Π is a finite sequence of configurations Π may successively visit, ending in the halting configuration in which no rules can be applied any more in any membrane. Like for other classes of P systems, the output of Π is the contents of the output membrane h_o projected onto the terminal alphabet T.

3 Superexponential Growth

In this section, we recall the classic example of superexponential growth which can be achieved by polymorphic P systems [2, 4].

Example 1 (superexponential growth [4]). Consider the following polymorphic P system:

$$\Pi_1 = (\{a\}, \{a\}, \mu, a, \langle a, a\rangle, \langle a, aa\rangle, s), \text{ where}$$
$$\mu = [\,[\,]_{1L}[\,[\,]_{2L}[\,]_{2R}]_{1R}]_s.$$

Π_1 has a superexponential growth rate. A graphical illustration of Π_1 is given in Figure 1.

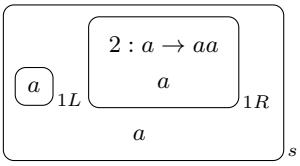

Fig. 1. The polymorphic P system Π_1 with superexponential growth

In the initial configuration, the membranes $1L$ and $1R$ define the rule $a \to a$ in the skin membrane s. Rule 2 in membrane $1R$ is formally represented by the pair of membranes $\langle 2L, 2R\rangle$, but graphically depicted as $a \to aa$, because the contents of $1R$ and $1R$ never change.

In the first derivation step, rule 1 ($a \to a$) is applied in the skin, leaving the contents of the membrane intact, and rule 2 ($a \to aa$) is applied in membrane $1R$, doubling the number of a's; therefore, after the first step, rule 1 will be of the form $a \to aa$. In the second step of the derivation, rule 1 will transform the multiset a in the skin into aa, and rule 2 will double the contents of the right-hand-side membrane $1R$ once again, thus transforming rule 1 into $a \to a^4$. Consequently, in the third derivation step, rule 1 will *quadruple* the number of a's in the skin.

In general, after k derivation steps, the contents of the right-hand-side membrane $1R$ will be 2^k, and rule 1 will have the form $a \to a^{2^k}$. The number of a's in the skin will be given by the product $1 \cdot 2 \cdot 4 \cdot \ldots \cdot 2^{k-1}$ or, equivalently, by the following formula:

$$2^0 \cdot 2^1 \cdot 2^2 \cdot \ldots \cdot 2^{k-1} = 2^{1+2+\ldots+k-1} = 2^{\frac{k(k-1)}{2}}.$$

4 Open Questions

In this section, we will enumerate some of the problems concerning polymorphic P systems which are still open. For further details, we refer to [1, Section 7].

4.1 Expressive Power

Question 1 (right polymorphism). Are non-cooperative polymorphic P systems, in which left-hand-side membranes cannot evolve, less powerful than general polymorphic P systems?

Question 2 (upper bounds). What are the upper bounds on the expressive power of non-cooperative polymorphic P systems?

Question 3 (target indications). What is the expressive power of non-cooperative polymorphic P systems *with* target indications?

4.2 Better Target Indications

The original article [2] already considers polymorphic rules with target indications. It turns out that pretty coarse indications, sending the whole right-hand side into a membrane, already allow building interesting behaviour. However, in a typical P system, target indications are assigned to *individual* symbols, not to entire right-hand sides. The following question seems therefore very natural.

Question 4 (finer targets). What is the most natural way to generalise target indications attached to *individual* symbols?

The dynamic nature of polymorphic P systems allows for stating yet further questions concerning target indications.

Question 5 (dynamic targets). What is the most natural way to define *dynamic* targets (i.e. target indications that the systems can modify dynamically)?

Question 6 (polymorphic tissue). What is the most natural way to define polymorphic *tissue* P systems?

4.3 Dissolution and Division

Question 7 (dissolution). What is the most natural way of introducing *membrane dissolution* for polymorphic P systems?

Question 8 (division). What is the most natural way of introducing *membrane division* for polymorphic P systems?

4.4 Applications

Question 9 (optimising simulators). Is polymorphism always easy to be simulated on conventional computers?

Question 10 (polymorphism vs. complexity). Can polymorphism be used for solving some complex problems faster?

Question 11 (killer applications). What are the problems that polymorphic P systems can solve *faster* than conventional P systems?

References

1. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. Polymorphic P systems: A survey. *Bulletin of the International Membrane Computing Society (IMCS)*, 2:79–102, 2016.
2. Artiom Alhazov, Sergiu Ivanov, and Yurii Rogozhin. Polymorphic P systems. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 6501 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2011.
3. John M. Coffin, Stephen H. Hughes, and Harold E. Varmus, editors. *Overview of Reverse Transcription – Retroviruses*. Cold Spring Harbor Laboratory Press, 1997.
4. Sergiu Ivanov. Polymorphic P systems with non-cooperative rules and no ingredients. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers*, volume 8961 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2014.
5. Gheorghe Păun. *Membrane Computing: An Introduction*. Natural Computing Series Natural Computing. Springer, 2002.
6. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
7. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
8. Bulletin of the International Membrane Computing Society (IMCS). http://membranecomputing.net/IMCSBulletin/index.php.
9. The P Systems Website. http://ppage.psystems.eu/.

Unfair P Systems

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

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

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

[3] IBISC, Université Évry, Université Paris-Saclay
23 Boulevard de France, 91025, Évry, France
sergiu.ivanov@univ-evry.fr

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

1 Introduction

Membrane computing is a research field originally founded by Gheorghe Păun in 1998, see [6]. Membrane systems (also known as P systems) are a model of computing based on the abstract notion of a membrane and the rules associated to it which control the evolution of the objects inside. In many variants of P systems, the objects are plain symbols from a finite alphabet, but P systems operating on more complex objects (e.g., strings, arrays) have been considered, too, e.g., see [3].

A comprehensive overview of different variants of membrane systems and their expressive power is given in the handbook, see [7]. For a state of the art view of the domain, we refer the reader to the P systems website [10] as well as to the bulletin series of the International Membrane Computing Society [9].

In this paper we reconsider and extend the variants P systems in which the application of rules in each step is controlled by a function on the applicable multisets of rules, possibly also depending on the current configuration; we call this function the *fairness function*. This new model has first been introduced in [2] and then also been published in [1]. In the standard variant investigated there, the fairness function chooses those applicable multisets for which the fairness function yields the minimal value. In this paper, we will mainly focus on the fairness function taking the maximal value.

After recalling some preliminary notions and definitions in the next section, in Section 3 we will define the model of *fair P systems* and give some examples

to exhibit the power of this general concept. In Section 4, for several well-known models of P systems we show how they can be simulated by P systems with a suitable fairness function. Future interesting/challenging research topics finally are touched in Section 5.

2 Preliminaries

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

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

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

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

For further introduction to the theory of formal languages and computability, we refer the reader to [7, 8].

2.1 (Hierarchical) P Systems

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

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

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

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

Quite often the rules associated with membranes are multiset rewriting rules (or special cases of such rules). Multiset rewriting rules have the form $u \to v$, with $u \in O^o \setminus \{\lambda\}$ and $v \in O^o$. If $|u| = 1$, the rule $u \to v$ is called *non-cooperative*; otherwise it is called *cooperative*. Rules may additionally be allowed to send symbols to the neighboring membranes. In this case, for rules in R_i,

$v \in O \times Tar_i$, where Tar_i contains the targets *out* (corresponding to sending the symbol to the parent membrane), *here* (indicating that the symbol should be kept in membrane i), and in_j (indicating that the symbol should be sent into the child membrane j of membrane i).

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

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

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

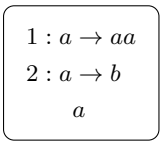

Fig. 1. The example P system Π_1

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

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

While maximal parallelism and halting by inapplicability have been standard ingredients from the beginning, various other derivation modes and halting conditions have been considered for P systems, e.g., see [7].

2.2 Flattening

The folklore flattening construction (see [7] for several examples as well as [4] for a general construction) is quite often directly applicable to many variants of P systems. Hence, also for the systems considered in this paper we will not explicitly mention how results are obtained by flattening.

3 P Systems with a Fairness Function

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

3.1 The General Idea of a Fairness Function in P Systems

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

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

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

where f is the fairness function defined for any configuration C of Π, the corresponding set $Appl_\delta(\Pi, C)$ of multisets of rules from Π applicable to C in the given derivation mode δ, and any multiset of rules $R \in Appl_\delta(\Pi, C)$. We then use the values $f(C, Appl_\delta(\Pi, C), R)$ for all $R \in Appl_\delta(\Pi, C)$ to choose a multiset $R' \in Appl_\delta(\Pi, C)$ of rules to be applied to the underlying configuration C.

A standard option for choosing R' is to require it to yield the minimal value or the maximal value for the fairness function, i.e., we either require $f(C, Appl_\delta(\Pi, C), R') \leq f(C, Appl_\delta(\Pi, C)), R)$ or else $f(C, Appl_\delta(\Pi, C), R') \geq f(C, Appl_\delta(\Pi, C)), R)$ for all $R \in Appl_\delta(\Pi, C)$.

In contrast to [2] and [1], in this paper we choose the variant with choosing R' to yield the maximal value for the fairness function (instead of the minimal value). This is the reason for calling the P system with fairness function an *unfair* P system.

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

As usually the derivation mode δ will be obvious from the context, we often shall omit it.

Fair or Unfair

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

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

Example 2. Consider the P system $\Pi_1 = (\{a, b\}, \{b\}, [\,_1\,]_1, a, R_1, 1)$ with the rule set $R_1 = \{1 : a \to aa, 2 : a \to b\}$ as considered in Example 1 together with the fairness function f_2 defined as follows: if a rule is applied n times then it contributes to the function value of the fairness function f_2 for the multiset of rules with 4^n. The total value for $f_2(R)$ for a multiset of rules R containing k copies of rule $1 : a \to aa$ and m copies of rule $2 : a \to b$ then is the sum $4^k + 4^m$. The resulting unfair P system $\Pi_2 = (\{a, b\}, \{b\}, [\,_1\,]_1, a, R_1, 1, f_2)$ is depicted in Figure 2; we observe that it can also be written as (Π_1, f_2).

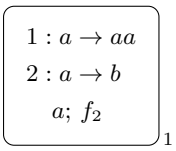

Fig. 2. The P system Π_2

In this unfair P system with one membrane working in the maximally parallel way, we again start with the axiom a and use the two rules $1 : a \to aa$ and $2 : a \to b$. If we apply only one of these rules to all m objects a, then the function value is 4^m and is maximal compared to the function values computed for a mixed multiset of rules using both rules at least once (e.g., $4^{m-1} + 4^1 < 4^m$ for any $m \geq 2$).

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

We observe that the effect is similar to that of controlling the application of rules by the well-known control mechanism called label selection, e.g., see [5], where either the rule with label 1 or the rule with label 2 has to be chosen. We will return to this model in Section 4.2. □

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

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

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

- $f(R) = 1$ if R only contains m copies of rule 3 and $m \in M$,

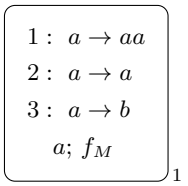

Fig. 3. The P system Π_3

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

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

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

4 Simulation Results

In this section, we show two general results. The first one describes how priorities can be simulated by a suitable fairness function in P systems of any kind working in the sequential mode. The second one exhibits how P systems with rule label control, see [5], can be simulated by suitable unfair P systems for any arbitrary derivation mode.

4.1 Simulating Priorities in the Sequential Derivation Mode

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

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

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

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

If we now define the task of f as choosing only those rules with maximal value, i.e., a rule r can be applied to configuration C if and only if $f(Appl(\Pi, C), \{r\}) = 1$, then we obtain the desired result.

4.2 Simulating Label Selection

In P systems with label selection only rules belonging to one of the predefined subsets of rules can be applied to a given configuration, see [5].

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

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

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

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

Theorem 2. *Let (Π, H, W) be a P system with label selection using any kind of rules in any kind of derivation mode. Then there exists an unfair P system (Π', f) with fairness function f simulating the computations in (Π, H, W) with f selecting the multisets of rules with maximal values.*

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

also fulfill the criteria of the given derivation mode restricted to rules with labels from U.

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

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

According to our standard selection criterion, we choose only those multisets of rules where the fairness function yields the maximal value 1, i.e., those R such that there exists a $U \in W$ such that the labels of all rules in R belong to U and R is applicable according to the underlying derivation mode with rules restricted to those having a label in U, which exactly mimicks the way of choosing R in (Π, H, W). Therefore, in any derivation mode δ, (Π', f_δ) simulates exactly step by step the derivations in (Π, H, W), obviously yielding the same computation results.

5 Conclusions and Future Research

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

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

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

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

– show similar simulation results with suitable fairness functions as in Section 4 for other control mechanisms used in the area of P systems;
– ...

References

1. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. P systems and the concept of fairness. In *Proceedings of the Conference on Mathematical Foundations of Informatics MFOI2017, November 9-11, 2017, Chisinau, Republic of Moldova*, 2017.
2. Artiom Alhazov, Rudolf Freund, and Sergiu Ivanov. Unfair P systems. In *Proceedings BWMC 2017*, 2017.
3. Rudolf Freund. P systems working in the sequential mode on arrays and strings. In Cristian Calude, Elena Calude, and Michael J. Dinneen, editors, *Developments in Language Theory, 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004, Proceedings*, volume 3340 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2004.
4. Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and Zandron. Flattening in (tissue) P systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 8340 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2014.
5. Rudolf Freund, Marion Oswald, and Gheorghe Păun. Catalytic and purely catalytic P systems and P automata: Control mechanisms for obtaining computational completeness. *Fundamenta Informaticae*, 136(1-2):59–84, 2015.
6. Gheorghe Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61:108–143, 1998.
7. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.
8. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, 3 volumes*. Springer, New York, NY, USA, 1997.
9. Bulletin of the International Membrane Computing Society (IMCS). `http://membranecomputing.net/IMCSBulletin/index.php`.
10. The P Systems Website. `http://ppage.psystems.eu/`.