Theoretical Informatics and Applications Will be set by the publisher
Informatique Théorique et Applications

COMPLEXITY CLASSES FOR MEMBRANE SYSTEMS

Antonio E. Porreca[1], Giancarlo Mauri[2] and Claudio Zandron[3]

Abstract. We compare various computational complexity classes defined within the framework of membrane systems, a distributed parallel computing device which is inspired from the functioning of the cell, with usual computational complexity classes for Turing machines. In particular, we focus our attention on the comparison among complexity classes for membrane systems with active membranes (where new membranes can be created by division of existing membranes) and the classes **PSPACE**, **EXP**, and **EXPSPACE**.

1991 Mathematics Subject Classification. 68Q05, 68Q15.

1. Introduction

Membrane systems (or P–systems) were introduced in [6] as a class of distributed parallel computing devices of a biochemical type. The basic model consists of a membrane structure composed of several cell-membranes, hierarchically embedded in a main membrane called skin membrane. The membranes delimit regions and can contain objects, which evolve according to given evolution rules associated with the regions. Such rules are applied in a nondeterministic and maximally parallel manner: at each step, all the objects which can evolve should evolve. A computation device is obtained: we start from an initial configuration and we let the system evolve. A computation halts when no further rule can be applied. The objects expelled through the skin membrane (or collected inside a specified output membrane) are the result of the computation. A survey and an up-to-date bibliography concerning P–systems can be found at the web address `http://psystems.disco.unimib.it`.

Keywords and phrases: Membrane systems, computational complexity, molecular computing

[1] DISCo - Università di Milano-Bicocca - Italy; e-mail: `porreca@disco.unimib.it`

[2] DISCo - Università di Milano-Bicocca - Italy; e-mail: `mauri@disco.unimib.it`

[3] DISCo - Università di Milano-Bicocca - Italy; e-mail: `zandron@disco.unimib.it`

Many variants have been defined up to now (see, e.g., [8]). In particular, in [7] one considers membrane systems where membranes play an active role in the computation: the evolution rules are associated both with objects and the membrane; the communication of the objects through the membranes is performed with the direct participation of the membranes; moreover, the membranes can not only be dissolved, but they can multiply by division. There are two different types of division rules: division rules for elementary membranes (i.e. membranes not containing other membranes) and division rules for non-elementary membranes.

In [7] and [4] it is shown how to solve two well known **NP** complete problems (the Satisfiability problem and the Hamiltonian Path problem, respectively) in linear time (and exponential space) with respect to the input length, using P–systems with active membranes. In [13] the two problems were solved in linear time by using P–systems with active membranes which make use of division for elementary membranes only. Moreover, in the same paper it was shown that, unless $\mathbf{P} = \mathbf{NP}$, a deterministic P–system with active membranes but without membrane division cannot solve an **NP** complete problem in polynomial time.

Starting from these results, various complexity classes for P–systems with active membranes were defined (see [9]). Such classes were then compared with usual complexity classes such as **P** and **NP** (see, e.g., [10] and [2]); in [12], [1] it was shown that two variants of membrane systems with active membranes contains all problems in the class **PSPACE**.

In this paper we focus our attention on the comparison among complexity classes for P–systems with active membranes and the complexity classes **PSPACE**, **EXP**, and **EXPSPACE**. We prove various inclusion properties which confirm the possibility to use membrane systems to attack open questions in computational complexity theory. In particular, the main result of the paper shows that there exists a complexity class for P–systems which includes the class **PSPACE** and which is included in the class **EXP**. As a consequence, this class could be used to attack the open question regarding whether the inclusion $\mathbf{PSPACE} \subseteq \mathbf{EXP}$ is proper or not.

The rest of the paper is organized as follows. In section 2 we give basic definitions for membrane systems with active membranes. In section 3 we recall the definitions of complexity classes for membrane systems and we present known results concerning relations among these classes and usual computational complexity classes. In section 4 we show how to simulate membrane systems by means of deterministic Turing machines and we give various results concerning relations between complexity classes for P–systems with active membranes and the complexity classes **PSPACE**, **EXP**, and **EXPSPACE**. Section 5 concludes the paper and presents open problems and directions for future research.

2. P-systems with Active Membranes

In this section we shortly recall basic notions concerning membrane systems which will be used in the rest of the paper. In particular, we will describe the

variant of P–systems with active membranes. For further details, we refer the reader to [8]. For elements of Formal Language theory, we refer the reader to [11]. For notions and results of computational complexity theory we refer the reader to [5].

A membrane structure is a construct consisting of several membranes placed in a unique membrane, called the skin membrane. We identify a membrane structure with a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane. The membranes can be marked by $+$, $-$ or 0, and this is interpreted as an electrical charge (0 is the neutral charge). A membrane identifies a region, delimited by it, and the membranes immediately inside it. If we place multisets of objects from a specified finite set Γ in the region, we get a super-cell. A super-cell system (or P-system) is a super-cell provided with evolution rules for its objects.

A P-system (of degree m) with active membranes is a construct

$$\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R),$$

where:

- $m \geq 1$;
- Γ is an alphabet;
- μ is a membrane structure, consisting of m membranes, labelled (not necessarily in a one-to-one manner) with numbers h, such that $1 \leq h \leq m$; all membranes in μ are supposed to be neutral;
- $w_1, \ldots, w_m$ are strings over Γ, describing the multisets of objects placed in the m regions of μ;
- R is a finite set of rules, of the following forms:

  (a) $[_h a \to v]_h^\alpha$, for $1 \leq h \leq m,\ a \in \Gamma,\ v \in \Gamma^*, \alpha \in \{+, -, 0\}$ (object evolution rules),

  (b) $a[_h]_h^{\alpha_1} \to [_h b]_h^{\alpha_2}$, where $1 \leq h \leq m, a, b \in \Gamma,\ \alpha_1, \alpha_2 \in \{+, -, 0\}$ (an object from the region immediately outside the membrane h is introduced in membrane h),

  (c) $[_h a]_h^{\alpha_1} \to [_h]_h^{\alpha_2} b$, for $1 \leq h \leq m,\ \alpha_1, \alpha_2 \in \{+, -, 0\},\ a, b \in \Gamma$ (an object is sent out from membrane h to the region immediately outside),

  (d) $[_h a]_h^\alpha \to b$, for $1 \leq h \leq m,\ \alpha \in \{+, -, 0\},\ a, b \in \Gamma$ (membrane h is dissolved),

  (e) $[_h a]_h^{\alpha_1} \to [_h b]_h^{\alpha_2} [_h c]_h^{\alpha_3}$, for $1 \leq h \leq m, \alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}, a, b, c \in \Gamma$ (division rules for elementary membranes)

  (f) $[_{h_0} [_{h_1}]_{h_1}^{\alpha_1} \ldots [_{h_k}]_{h_k}^{\alpha_1} [_{h_{k+1}}]_{h_{k+1}}^{\alpha_2} \ldots [_{h_n}]_{h_n}^{\alpha_2}]_{h_0}^{\alpha_0} \to$
  $\to [_{h_0} [_{h_1}]_{h_1}^{\alpha_3} \ldots [_{h_k}]_{h_k}^{\alpha_3}]_{h_0}^{\alpha_5} [_{h_0} [_{h_{k+1}}]_{h_{k+1}}^{\alpha_4} \ldots [_{h_n}]_{h_n}^{\alpha_4}]_{h_0}^{\alpha_6}$ for $k \geq 1, n \geq 1, 0 \leq i \leq n, 1 \leq h_i \leq m$ and $\alpha_0, ..., \alpha_6 \in \{+, -, 0\}$ with $\{\alpha_1, \alpha_2\} = \{+, -\}$ (division rules for non-elementary membranes)

These rules are applied accordingly to the principles in [7]:

(1) All the rules are applied in parallel: in a step, the rules of type (a) (i.e. the rules that do not modify the membranes) are applied to all objects to

which they can be applied; all other rules are applied to all membranes to which they can be applied; an object can be used by only one rule, non-deterministically chosen (there is no priority relation among rules), but any object which can evolve by a rule of any form, must evolve.

(2) If a membrane is dissolved, then all the objects in its region are left free in the region immediately above it. Because all rules are associated with membranes, the rules of a dissolved membrane are no longer available at the next steps. The skin membrane is never dissolved.

(3) All objects and membranes not specified in a rule and which do not evolve are passed unchanged to the next step.

(4) If at the same time a membrane h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then in the new copies of the membrane we introduce the result of the evolution; that is, we may suppose that first the evolution rules of type (a) are used, changing the objects, and then the division is produced, so that in the two new membranes with label h we introduce copies of the changed objects. Of course, this process takes only one step. The same assertions apply to the division by means of a rule of type (f): we always assume that the rules are applied "from bottom-up", in one step, but first the rules of the innermost region and then level by level until the region of the skin membrane.

(5) The rules associated with a membrane h are used for all copies of this membrane, irrespective whether or not the membrane is an initial one or it is obtained by division. At one step, a membrane h can be the subject of only one rule of types (b) – (f).

(6) The skin membrane can never divide. As with any other membrane, the skin membrane can be "electrically charged".

The membrane structure at a given time, together with all multisets of objects associated with the regions defined by the membrane structure, is the *configuration* of the system at that time. The *initial configuration* is $(\mu, w_1, \ldots, w_m)$. We can pass from a configuration to another one by using the rules in R, according to the principles previously described (we call this a *transition*). A *computation* is a sequence of transitions between configurations. A computation *halts* when there is no rule which can be applied to objects and membranes in the current configuration.

During the computation, objects can leave the skin membrane (with a rule of type *(c)*). The symbols which leave the skin membrane are collected in the order of their expelling from the system, so a string is associated to a complete computation. All symbols which remain in the system at the end of a halting computation are not considered in the generated strings. If a computation never stops, then it provides no output.

In order to clarify the functioning of a P–system with active membranes, we propose the following example. In particular, we present a system which generates all possible truth assignments for two boolean variables x_1 and x_2, and then non-deterministically select one of the truth assignment to be expelled through the skin

membrane. We stress the fact that the example is given for explanation purpose only; the output of the computation could also be obtained with a simpler system. Nonetheless, the example (hopefully) gives a first hint concerning the use and the power of membrane division, and its use to approach NP-complete problems (we refer the reader to [7], [4], and [13] for details on this topic).

Example 2.1. Consider the system with active membranes

$$\Pi = (\Gamma, \mu, w_1, w_2, w_3, R),$$

where $\Gamma = \{c_0, c_1, c_2, c_3, c_4, x_1, x_2, t_1, t_2, f_1, f_2, r, r_1, r_2\}$, $\mu = [_1\,[_2\,[_3\,]_3^0\,]_2^0\,]_1^0$, $w_1 = \{r\}$, $w_2 = \emptyset$, $w_3 = \{c_0, x_1, x_2\}$, and R contains the rules

- $[c_i \to c_{i+1}]_3^0$, $0 \leq i \leq 1$
- $[x_j]_3^0 \to [t_j]_3^+ [f_j]_3^-$, $1 \leq j \leq 2$
- $[_2[_3]_3^+ [_3]_3^-]_2^0 \to [_2[_3]_3^0]_2[_2[_3]_3^0]_2^0$
- $[_3c_2]_3^0 \to c_3$
- $[_2c_3]_2^0 \to c_4[_2]_2^+$
- $r[_2]_2^+ \to [_2r_1]_2^-$
- $[_2r_1]_2^- \to r_2$
- $[_1t_k]_1^0 \to [_1]_1^0 t_k$ and $[_1f_k]_1^0 \to [_1]_1^0 f_k$, $1 \leq k \leq 2$

We show that this system generates all possible truth assignments for two boolean variables x_1 and x_2, and non-deterministically sends in the environment one of this assignment.

The starting configuration of the system is the following: $[_1r[_2[_3c_0\,x_1\,x_2]_3^0]_2^0]_1^0$. The only applicable rules at this moment are the rule $[c_0 \to c_1]_3^0$, which replaces the symbol c_0 with c_1, and one (non–deterministically chosen) rule among rules $[x_j]_3^0 \to [t_j]_3^+ [f_j]_3^-$, $1 \leq j \leq 2$, which divide membrane 3 in two membranes of opposite charge, replacing x_j with t_j and f_j, and duplicating all other symbols (including the symbol c_1). We assume, in the following, that the applied rule at this step is $[x_1]_3^0 \to [t_1]_3^+ [f_1]_3^-$; the reader can easily check that the order of application of these rules is not relevant. After applying these rules, the first computation step ended, and the configuration of the system is $[_1r[_2[_3c_1\,t_1\,x_2]_3^+[_3c_1\,f_1\,x_2]_3^-]_2^0]_1^0$.

We can now apply the rule $[_2[_3]_3^+[_3]_3^-]_2^0 \to [_2[_3]_3^0]_2[_2[_3]_3^0]_2^0$, which divides membrane 2 in two membranes, separating the inner membranes with opposite charge. After the second computation step, the system is in the following configuration: $[_1r[_2[_3c_1\,t_1\,x_2]_3^0]_2^0[_2[_3c_1\,f_1\,x_2]_3^0]_2^0]_1^0$.

It is now possible to apply the rules $[c_1 \to c_2]_3^0$, which replaces the symbol c_1 with c_2 in both membranes with label 3, and $[x_2]_3^0 \to [t_2]_3^+ [f_2]_3^-$, which divides each membrane 3 in two membranes of opposite charge. After the third computation step, we have $[_1r[_2[_3c_2\,t_1\,t_2]_3^+[_3c_2\,t_1\,f_2]_3^-]_2^0[_2[_3c_2\,f_1\,t_2]_3^+[_3c_1\,f_1\,f_2]_3^-]_2^0]_1^0$.

We can now apply again the rule $[_2[_3]_3^+[_3]_3^-]_2^0 \to [_2[_3]_3^0]_2^0[_2[_3]_3^0]_2^0$, which divides each copy of membrane 2 in two membranes, separating the inner membranes according to their charge. After the fourth computation step, the configuration of the system is $[_1r[_2[_3c_2\,t_1\,t_2]_3^0]_2^0[_2[_3c_2\,t_1\,f_2]_3^0]_2^0[_2[_3c_2\,f_1\,t_2]_3^0]_2^0[_2[_3c_2\,f_1\,f_2]_3^0]_2^0]_1^0$.

We dissolve now each copy of membrane 3 by means of the rule $[_3 c_2]^0_3 \to c_3$, thus obtaining $[_1 r [_2 c_3\, t_1\, t_2]^0_2 [_2 c_3\, t_1\, f_2]^0_2 [_2 c_3\, f_1\, t_2]^0_2 [_2 c_3\, f_1\, f_2]^0_2]^0_1$, and then we apply the rule $[_2 c_3]^0_2 \to c_4 [_2]^+_2$, which change the polarity of all copies of membrane 2 to positive.

It is now possible to apply the rule $r[_2]^+_2 \to [_2 r_1]^-_2$ on the (unique) symbol r in membrane 1; this symbol is non-deterministically sent to a single copy of membrane 2, changing its charge from positive to negative. Let us assume, for example, that it reaches the copy of membrane 2 containing the symbols t_1 and f_2. The configuration is now $[_1 c_4 c_4 c_4 c_4 [_2 t_1\, t_2]^+_2 [_2 r_1\, t_1\, f_2]^-_2 [_2 f_1\, t_2]^+_2 [_2 f_1\, f_2]^+_2]^0_1$.

By means of the rule $[_2 r_1]^-_2 \to r_2$ we dissolve this copy of membrane 2; the objects t_1 and f_2 reaches the skin membrane. By means of the rules $[_1 t_1]^0_1 \to [_1]^0_1 t_1$ and $[_1 f_2]^0_1 \to [_1]^0_1 f_2$ (applied in any order) we sent these two symbols in the region outside the skin membrane and then the computation halts, producing in output one possible truth assignment for x_1 and x_2. □

3. Complexity classes for P-systems with Active Membranes

We recall from [9] various notions of complexity classes for membrane systems, and we present known results concerning these classes and their relations with usual computational complexity classes.

We start with the definition of recognizer P–systems and their properties.

Definition 3.1. A *recognizer P–system* is a P–system Π with external output such that:

(1) The working alphabet contains two distinct symbols, *yes* and *no*
(2) If C is a halting computation of the system, then exactly one symbol of *yes* and *no* is expelled through the skin membrane.

Let C be a halting computation of Π. If the object emitted through the skin membrane is *yes*, then we will say that C is an *accepting computation*; if the object emitted through the skin membrane is *no*, then we will say that C is a *rejecting* computation.

Definition 3.2. We say that a recognizer P–system is *confluent*, if it verifies the following conditions:

(1) Every computation of Π is a halting computation
(2) If C_1 and C_2 are two computations of Π with the same initial configuration, then the result for both C_1 and C_2 is the same. In other words, either both C_1 and C_2 are accepting computations or both are rejecting computations.

Definition 3.3. Consider an alphabet Γ and a family $\mathbf{\Pi} = \{\Pi_x \mid x \in \Gamma^*\}$ of P–systems of the same type. We say that $\mathbf{\Pi}$ is *polynomially uniform* if there exist a deterministic Turing machine M and a constant k such that, for every input $x \in \Gamma^*$, M constructs the P–system Π_x in time $O(|x|^k)$.

We are now ready to define complexity classes for membrane systems.

Definition 3.4. Let $\mathcal{D}$ be a class of confluent recognizer P–systems, $f : \mathbb{N} \to \mathbb{N}$ a proper complexity function (see [5]), and $L \subseteq \Gamma^*$ a language over the alphabet Γ. We say that $L \in \mathbf{MC}_{\mathcal{D}}(f)$ if there exists a family, $\mathbf{\Pi} = \{\Pi_x \mid x \in \Gamma^*\}$, such that:

- $\mathbf{\Pi}$ is $\mathcal{D}$–consistent, that is, each $\Pi_x \in \mathbf{\Pi}$ is in $\mathcal{D}$;
- $\mathbf{\Pi}$ is polynomially uniform;
- For each $x \in \Gamma^*$, every computation in Π_x requires at most $f(|x|)$ steps;
- For each $x \in \Gamma^*$, computations of Π_x are accepting if and only if $x \in L$.

Following the definition for computational complexity classes for Turing machines, it is useful to define the class of languages which can be recognized efficiently (that is, in polynomial time with respect to the input length) by recognizer P–systems:

Definition 3.5. The class of languages recognized in polynomial time (w.r.t. the input length) by a uniform family of confluent P–systems of type $\mathcal{D}$ is

$$\mathbf{PMC}_{\mathcal{D}} = \bigcup_{k \in \mathbb{N}} \mathbf{MC}_{\mathcal{D}}(O(n^k)).$$

It is known that the class $\mathbf{PMC}_{\mathcal{D}}$ is closed under complementation and under polynomial time reduction.

We can also consider complexity classes for non–deterministic recognizer P–systems. In this case, we do not require anymore that all halting computations give the same answer. On the contrary, given a non-deterministic recognizer P-system Π_x associated with a string $x \in L$, we just require the existence of at least one accepting computation of Π_x, similarly to non-deterministic Turing machines.

Definition 3.6. Let $\mathcal{D}$ be a class of non–deterministic recognizer P–systems, $f : \mathbb{N} \to \mathbb{N}$ a proper complexity function, and $L \subseteq \Gamma^*$ a language over the alphabet Γ. We say that $L \in \mathbf{NMC}_{\mathcal{D}}(f)$ if there exists a family, $\mathbf{\Pi} = \{\Pi_x \mid x \in \Gamma^*\}$, such that:

- $\mathbf{\Pi}$ is $\mathcal{D}$–consistent, that is, each $\Pi_x \in \mathbf{\Pi}$ is in $\mathcal{D}$;
- $\mathbf{\Pi}$ is polynomially uniform;
- For each $x \in \Gamma^*$, every computation in Π_x requires at most $f(|x|)$ steps;
- For each $x \in \Gamma^*$, there exists an accepting computation in Π_x if and only if $x \in L$.

Definition 3.7. The class of languages recognized in polynomial time (w.r.t. the input length) by a uniform family of non-deterministic P–systems of type $\mathcal{D}$ is

$$\mathbf{NPMC}_{\mathcal{D}} = \bigcup_{k \in \mathbb{N}} \mathbf{NMC}_{\mathcal{D}}(O(n^k)).$$

It is known that the class $\mathbf{NPMC}_{\mathcal{D}}$ is closed under polynomial time reduction.

In [4], [7], [13] various solutions for **NP**-complete problems were given, using P–systems with active membranes working in polynomial time (but exponential

space) with respect to the input length. Following these results, a series of investigations started to define relations among complexity classes for P–systems with active membranes. We recall here the main results.

Let $f : \mathbb{N} \to \mathbb{N}$ be a proper complexity function (see [5]). By $\mathbf{MC}_{\mathcal{AM}}(f(n))$ we denote the class of languages recognized by P–systems with active membranes within time $f(n)$; by $\mathbf{MC}_{\mathcal{EAM}}(f(n))$ we denote the class of languages recognized within time $f(n)$ by P–systems with active membranes which make use of elementary membrane division only; finally, by $\mathbf{MC}_{\mathcal{NAM}}(f(n))$ we denote the classes of languages recognized within time $f(n)$ by P–systems with active membranes which do not make use of membrane division.

In particular, by $\mathbf{PMC}_{\mathcal{AM}}$, $\mathbf{PMC}_{\mathcal{EAM}}$, and $\mathbf{PMC}_{\mathcal{NAM}}$ we denote the class of languages recognized in polynomial time (w.r.t. the input length) by P–systems with active membranes, P–systems with active membranes which make use of elementary membrane division only, and P–systems with active membranes which do not make use of membrane division, respectively.

From the definitions, it follows immediately:

Theorem 3.8. $\mathbf{MC}_{\mathcal{NAM}}(f(n)) \subseteq \mathbf{MC}_{\mathcal{EAM}}(f(n)) \subseteq \mathbf{MC}_{\mathcal{AM}}(f(n))$, *for all proper complexity functions* $f : \mathbb{N} \to \mathbb{N}$ *(thus, in particular,* $\mathbf{PMC}_{\mathcal{NAM}} \subseteq \mathbf{PMC}_{\mathcal{EAM}} \subseteq \mathbf{PMC}_{\mathcal{AM}}$*)*

It is not known which inclusions are proper.

In [13] it is shown how to solve the **NP**-complete problem SAT (Satisfiability for boolean formulas) using P–systems with active membranes and division for elementary membranes only, which work in polynomial time. Given an instance of SAT, the time needed to build the system which solves the instance is polynomial, with respect to the instance length, for a deterministic Turing machine.

Thus we have:

Theorem 3.9. $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{EAM}}$

From this result and from the closure properties for $\mathbf{PMC}_{\mathcal{EAM}}$ it also follows:

Theorem 3.10. $\mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{EAM}}$

If we allow the use of division for non-elementary membranes, then the following can be proved ([1], [12]):

Theorem 3.11. $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{AM}}$

On the contrary, if we do not allow membrane division, then the languages accepted in polynomial time by (deterministic) membrane systems can also be accepted by a deterministic Turing machine in polynomial time, as showed in [13]. In [3] a characterization of **P** was given, in terms of P–systems with active membranes but without membrane division:

Theorem 3.12. $\mathbf{P} = \mathbf{PMC}_{\mathcal{NAM}}$

For the interested reader, we also recall here another important characterization of **P**, given in [2], where it is shown that membrane division alone does not suffice to obtain a speed-up of computations:

Theorem 3.13. $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}}(n\delta, nEC)$, *where* $\mathbf{PMC}_{\mathcal{AM}}(n\delta, nEC)$ *is the class of languages recognized in polynomial time by P–systems with active membranes, which do not make use of neither the dissolving membrane feature, nor of polarization on the membranes.*

4. Simulation of P-systems with active membranes

In this section we will describe an algorithm to simulate a computation of any confluent recognizer P-system with active membranes. An analysis of the resources will show the relation between some complexity classes; in particular, we will prove that each language in the class $\mathbf{PMC}_{\mathcal{AM}}$ can be accepted by a deterministic Turing machine (DTM) in exponential time.

Our description will be in terms of a program running on a random access machine, exploiting the fact that such device can be simulated by a DTM with a polynomial slowdown.

4.1. A coding for P-systems

First of all, we must choose the data structures that model the P-system to be simulated; these will be the description to be presented as input to the simulation algorithm.

The fundamental component of a P-system is its membrane structure; an adequate formalism to describe it is the rooted tree. Each membrane (equivalently, the region it delimits from outside) corresponds to a node in the tree; in particular, the skin is associated with the root. Two nodes are connected by an edge if and only if they represent membranes placed one immediately inside the other. The tree representation will allow us to simulate the application of the developmental rules in the right order (e.g. division of a non-elementary membrane can occur only after its children membranes have evolved).

Each node *mbr* in the structural tree can be implemented with a record containing the following fields:

- a pointer to the *parent* node (which is set to *nil* for the root);
- a list of its *children* nodes;
- the *label* of the associated membrane;
- the *polarization* of the corresponding membrane;
- the *multiset* of objects in its region.

For the sake of simplicity, we will represent multisets as a linked list of objects (with repetitions in the case of multiple occurrences of the same object). This implementation is, essentially, a unary coding of the multiplicities of objects; exponentially less space would be enough for a binary coding, but this would not reduce the overall space requirements of our algorithm, which are mainly due to the number of membranes (degree) of the P-system: it can grow exponentially during the computation in the worst case.

The developmental rules can also be represented with a record, containing fields corresponding to the elements in the left and right hand side of the rule (respectively, the objects and membranes which are present in the configuration before and after the rule application) and the conditions for rule applicability (label and polarization of the membrane where the rule can be applied).

Summarizing, in the algorithm we will make use of the following:

- *rule* is the pointer to the rule which is currently under simulation;
- *rule.h* is the label of the membrane where the rule is applied;
- *rule.α* is the polarization required for the rule to be applied;
- *rule.a* is the object on the left part of the rule;
- *rule.v* is the multiset of objects in the right part of the rule;
- *mbr* is the pointer to the membrane which is currently under simulation;
- *mbr.label* is the label of the membrane *mbr*;
- *mbr.polarization* is the polarization of the membrane *mbr*;
- *mbr.multiset* is the multiset of objects in the membrane *mbr*.
- *mbr.parent* is a pointer to the parent node;
- *mbr.children* is a set of pointers to the children nodes;

Each object x in a multiset, and each membrane in the system have a further attribute *mark* which can be either 0 or 1; the *mark* attribute is used to distinguish which objects and membranes still have to be used for the current simulation step, as it will be explained in the next section.

4.2. Description of the algorithm

The simulation of a computation of the P-system is carried out by applying the developmental rules, starting from the internal membranes (i.e. the leaves in the structural tree) towards the external ones; for each simulated step, a depth-first traversal of the membrane structure is performed. The procedure halts when an object *yes* or *no* is expelled from the skin; then, the algorithm returns the same result as the simulated system.

Each object can be involved in the application of only one rule during each step of computation. Moreover, each membrane can be involved in only one rule of type [b] to [f]. Thus, as the simulation of a single step of computation of the P–system is done in various sub–steps, it is necessary to "mark" the components of the system already used in one of previous sub–steps as currently unavailable. As said before, we do this by associating with each membrane and with each object a "mark" bit. All these bits must be cleared (set to zero) before simulating each computation step of the P–system; each time an object or a membrane is involved in the application of a rule, its mark bit is set to 1.

In each membrane, the algorithm iterates the rule applications until no further rule can be used, simulating in a sequential way the maximal parallelism of the P–system.

We are considering only confluent systems, so the nondeterminism has only a *local* effect (i.e. regarding only the choice between multiple applicable rules during

one step); thus, the rule selection policy is immaterial with regard to the correctness of the computation. Without loss of generality, we shall assume that the first applicable rule is selected, with the exception that evolution rules are applied *before* other types of rules (to ensure the correct interactions between evolution of objects and membrane division or dissolution).

To analyze the time required to simulate the application of the rules, we will denote by $\deg(t)$ the degree (that is, the number of membranes) of the system after t steps of computation. By $\text{obj}(t)$ we denote the maximum cardinality among all multisets in the system to be simulated after t steps. We will find expressions for these parameters in section 4.3.

Finally, we denote by k the size of the initial description of the P–system, i.e. before the computation starts. The description of the system is given as illustrated in section 2. The size k is given by the sum of the size of the initial configuration (membrane structure and multisets), the size of the alphabet and the length of the developmental rules (we will use this parameter as an upper bound to the length of each rule and of the initial configuration).

We point out that, when we simulate a step t of computation, the maximum number of objects to be considered in each membrane is not $\text{obj}(t)$; in fact, as the simulation of a computation step proceeds from the inner membranes towards the external ones, the application of the rules in an external membrane is done with the objects originally present in it plus the objects that are communicated from the inner membranes. Of course, the "mark" bit of these last objects is set to 1, and thus they will not be used in the application of the rules in the membrane; nonetheless, the simple check for the "mark" bit requires a time proportional to the number of objects. The maxiumum number of objects in a membrane c at step t is, at most, $\text{obj}(t) + R$, where $\text{obj}(t)$ is the number of objects initially present in the membrane, and R is the number of objects which can reach membrane c from the membranes inside it. The worst case is given when all evolved objects of all inner membranes reach the most external membrane (the skin). As each initial object can be replaced by at most k objects, and the number of inner membranes is limited by $\deg(t)$, we have $R \leq k \cdot \deg(t) \cdot \text{obj}(t)$; thus, the maximum number of objects in a membrane is $O(k \cdot \deg(t) \cdot \text{obj}(t))$.

We shall now describe the operations to be performed in order to simulate each kind of rule.

4.2.1. *Object evolution rules*

The following algorithm (given using pseudo-code instructions) shows the simulation of a rule of the form $[_h a \to v]^{\alpha}_h$.

The procedure returns $false$ if the rule is not currently applicable in the current membrane (that is, $mbr \neq h$ or $mbr.polarization \neq \alpha$), or $true$ when the application of the rule has been simulated.

```
ApplyEvolutionRule(rule, mbr):
1   if rule.h ≠ mbr.label or rule.α ≠ mbr.polarization then
2       return false
```

```
3   for each x in mbr.multiset do
4       if x = rule.a and x.mark = 0 then
5         remove(x,mbr.multiset)
6         for each y in rule.v do
7             add(y,mbr.multiset)
8             y.mark ← 1
9         return true
10  return false
```

Lines 1–2 test whether the current membrane has the correct label and polarization; these instructions can be executed in $O(1)$ time. The *for* loop in lines 3–9 searches for a copy of a in the current multiset (performing a list scan); if it is found, the object x is removed, by means of the procedure *remove(x,mbr.multiset),* from the multiset of mbr by adjusting the list pointers, and the objects in v are then added to the multiset mbr. This loop is executed at most once for every object in the region. As the loop beginning at line 3 checks each object in the current multiset, it is executed at most $k \cdot \deg(t)\mathrm{obj}(t)$ times. The test at line 4, which requires constant time, is checked at each iteration while all the other instructions between lines 5 to 9 are executed at most one time (when the test is successful). The operations at lines 5 and 9 require constant time, while lines 6–8 require a time proportional to the size of v, which is $O(k)$.

Thus, the total time required to simulate an evolution rule is $O(k \cdot \deg(t)\mathrm{obj}(t))$.

4.2.2. *Communication rules*

These rules, of the form ${}_h[a]_h^{\alpha_1} \to [\,]_h^{\alpha_2}b$ or $a[\,]_h^{\alpha_1} \to {}_h[b]_h^{\alpha_2}$, can be simulated with a procedure analogous to ApplyEvolutionRule, except that there are two different multisets involved and the object to add is unique. Thus, the time required to simulate a communication rule depends only on the length of the list to scan, which is $O(k \cdot \deg(t)\mathrm{obj}(t))$.

4.2.3. *Dissolution rules*

```
ApplyDissolutionRule(rule, mbr):
1   if rule.h ≠ mbr.label or rule.α ≠ mbr.polarization or mbr.mark = 1 then
2     return false
3   for each x in mbr.multiset do
4       if x = rule.a and x.mark = 0 then
5         mbr.mark ← 1
6         remove(x, mbr.multiset)
7         add(rule.b, mbr.multiset)
8         b.mark ← 1
9         for each y in mbr.multiset do
10            add(y, mbr.parent.multiset)
11        for each z in mbr.children do
12            add(z, mbr.parent.children)
13        remove(mbr,mbr.parent.children)
```

```
14          return true
15  return false
```

When a membrane is dissolved, we first need to remove the object used on the left part of the applied rule. Then, all remaining objects in the multiset of the dissolved membrane are added to the multiset of its parent membrane. Finally, the membrane structure is adjusted by removing the dissolved membrane. From the previous algorithm it is easy to see that the total time to perform these operations is $O(k \cdot \deg(t)\text{obj}(t) + \deg(t)) = O(k \cdot \deg(t)\text{obj}(t)$.

4.2.4. *Elementary membrane division rules*

```
ApplyElementaryDivisionRule(rule, mbr):
1   if rule.h ≠ mbr.label or rule.α ≠ mbr.polarization or mbr.mark = 1 then
2     return false
3   for each x in mbr.multiset do
4       if x = rule.a and x.mark = 0 then
5         mbr.mark ← 1
6         remove(x, mbr.multiset)
7         tmp ← duplicate mbr
8         add(tmp, mbr.parent.children)
9         mbr.polarization ← rule.α2
10        tmp.polarization ← rule.α3
11        add(rule.b,mbr.multiset)
12        b.mark ← 1
13        add(rule.c,tmp.multiset)
14        c.mark ← 1
15        return true
16  return false
```

When a membrane is duplicated by division, we first need to remove the object used on the left part of the applied rule. Then, the membrane is duplicated with all its objects and rules, and the polarization of the two membranes is adjusted. Finally, the membrane structure is adjusted. Notice that, as we are considering elementary membranes, the size of the multiset of the membrane is, at most, $k \cdot \text{obj}(t)$), as no object evolved from inner membranes can be added to it. Then, the total time to perform these operations is $O(k \cdot \text{obj}(t))$.

4.2.5. *Non-elementary membrane division rules*

```
ApplyNonElementaryDivisionRule(rule, mbr):
1   if rule.h0 ≠ mbr.label or rule.α0 ≠ mbr.polarization or mbr.mark = 1 then
2     return false
3   for each c in mbr.children do
4       if c.mark = 1 then
5         return false
6   n ← numberofmembranestobesplitted
7   for i ← 1 to n do
```

```
8       found ← false
9       for each c in mbr.children do
10          if c.label = rule.h_i and c.polarization = rule.α_i then
11              found ← true
12      if not found then
13          return false
14  for each c in mbr.children do
15      if c.polarization ≠ 0 then
16          found ← false
17          for i ← 1 to n do
18              if rule.h_i = c.label and rule.α_i = c.polarization then
19                  found ← true
20          if not found then
21              return false
22  mbr.polarization ← rule.α_5
23  tmp ← new membrane
24  tmp.label ← rule.h_0
25  tmp.polarization ← rule.α_6
26  mbr.mark ← 1 and tmp.mark ← 1
27  for each c in mbr.children do
28      c.mark ← 1
29      if c.polarization = rule.α_1 then
30          c.polarization ← rule.α_3
31      else if c.polarization = rule.α_2 then
32          remove(c,mbr.children)
33          add(c,tmp.children)
34          c.polarization ← rule.α_4
35      else
36          d ← DuplicateStructureRootedIn(c)
37          add(d,tmp.children)
38  return true
```

Lines 3 to 21 describe the check for rule applicability, with respect to the children membranes of *mbr* and their polarization. In order to do this, we need to check that:

(1) No membrane inside *mbr* has already been used as a subject of another rule in the same computation step; this is done in lines 3–5.
(2) All membranes to be used in the rule (with positive and negative polarization) are present as children of *mbr*; this is done in lines 6–13.
(3) No membrane with non–neutral polarization is present as a child of *mbr* that is not used in the rule.

In other words, we need to check that the children of *mbr* with non–neutral polarization are exactly the set of membranes used in the rule to be simulated.

As the description of the rule is bounded by k, and the number of children membranes is $O(\deg(t))$, then the time required to perform operations between

lines 3–13 is $O(k{\cdot}\deg(t))$. The same amount of time is needed to perform operations between lines 14–21.

The lines 22–26 require a constant amount of time to be executed.

The procedure at line 36 is the most time–consuming operation among those in the loop in lines 27-37. It duplicates a membrane subsystem (with all its sub–membranes, objects, and rules), having as its input a membrane c, which becomes the root membrane of the subsystem to be duplicated.

The number of objects in the subsystem is less than $k \cdot \deg(t) \cdot \mathrm{obj}(t)$, and the number of membranes in the subsystem is less than $2 \cdot \deg(t)$.

The time required to execute the procedure at line 36 is, therefore, $O(k \cdot \deg(t) \cdot \mathrm{obj}(t))$.

This procedure is executed for each child membrane of c, thus for at most $2 \cdot \deg(t)$ times. Hence, the loop of lines 27–37 requires $O(k \cdot \deg(t)^2 \cdot \mathrm{obj}(t))$.

Summarizing, the total time needed to simulate the application of a non–elementary membrane division rule is $O(2 \cdot k \cdot \deg(t) + k \cdot \deg(t)^2 \cdot \mathrm{obj}(t))$ that is $O(k \cdot \deg(t)^2 \cdot \mathrm{obj}(t))$.

4.3. Space complexity analysis

Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_k, R)$ be a confluent recognizer P-system with active membranes. Our purpose is to calculate how much space we need to simulate a computation of Π on a Turing machine with the algorithm we just described.

Since the developmental rules are fixed (i.e. they do not evolve during a computation), they remain part of the input and do not affect the space complexity of the algorithm. Instead, we need to store the instantaneous configuration of the P-system: it requires linear space with respect to the degree and the number of objects in the system.

Thus, the storage needed for the configuration of Π after t steps is a function of $\deg(t)$ and $\mathrm{obj}(t)$:

$$\mathrm{cfg}(t) = O(\deg(t) + \deg(t) \cdot \mathrm{obj}(t)) = O(\deg(t) \cdot \mathrm{obj}(t)). \tag{1}$$

We also need some auxiliary data structure to perform the depth-first traversal of the structural tree (e.g. a stack). We will intentionally ignore this: the traversal requires $O(\deg(t))$ space, so it does not change the space complexity substantially.

Thus, we will use $\mathrm{cfg}(t)$ as a measure of the space complexity of our algorithm. We need to find an explicit expression for both $\mathrm{obj}(t)$ and $\deg(t)$: we shall prove that the former depends on the latter, which in turn depends on the type of P-system we consider (with or without non-elementary membrane division).

Lemma 4.1. *Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R)$ be a confluent recognizer P-system with active membranes; let k be the length of the description of Π. Then* $\mathrm{obj}(t) = 2^{O(t \log k)} \cdot \deg(t)^t$.

Proof. Since the length of the initial configuration of Π is at most k, this is of course an upper bound on the number of objects in each region before the first

step of computation:

$$\mathrm{obj}(0) \leq k.$$

Each object in the region can be subject to the application of an object evolution rule; the length of such rule is, again, bounded by k (an object can be substituted by at most k objects), thus the occurrences of objects can increase up to k times. Furthermore, each membrane in the subtree rooted in the current membrane can be dissolved; in this case, all its objects reach the current region after having evolved. The number of membranes in this subtree is at most $\deg(t)$ (this is the case when the subtree is rooted in the skin membrane). Thus, the number of objects after step $t+1$ is at most

$$\mathrm{obj}(t+1) \leq k \cdot \mathrm{obj}(t) \cdot \deg(t).$$

By induction on t we get

$$\mathrm{obj}(t) \leq k^{t+1} \cdot \prod_{i=0}^{t-1} \deg(i) \leq 2^{(t+1)\log k} \cdot \prod_{i=0}^{t} \deg(i) \leq 2^{(t+1)\log k} \cdot \deg(t)^t$$

which is $2^{O(t \log k)} \cdot \deg(t)^t$. □

We shall analyze separately P-systems of type $\mathcal{EAM}$ and $\mathcal{AM}$, since the value of $\deg(t)$ varies in the two cases.

4.3.1. *Space analysis for P-systems with elementary active membranes*

Let us consider the parameter $\deg(t)$ in the case of P-systems which do not make use of division rules for non-elementary membranes. The only way to increase the degree of the P-systems is, then, by division of membranes which do not contain other membranes; thus, the increase is maximum when every membrane but the skin is a leaf in the structural tree.

Lemma 4.2. *Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R)$ be a confluent recognizer P-system with elementary active membranes; let k be the length of the description of Π. Then $\deg(t) = 2^{O(t+\log k)}$.*

Proof. Initially, the degree of Π is bounded by k:

$$\deg(0) \leq k.$$

By applying a division rule to every elementary membrane at time t, their number doubles; since there are less than $\deg(t)$ of them, we get

$$\deg(t+1) \leq 2\deg(t).$$

By induction on t, we can conclude that $\deg(t) \leq 2^t \cdot k = 2^{t+\log k}$. □

Lemma 4.3. *Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R)$ be a confluent recognizer P-system with elementary active membranes; let k be the length of the description of Π. Then* $\mathrm{cfg}(t) = 2^{O(t^2 + t \log k)}$.

Proof. It follows immediately from equation 1 and from lemmata 4.1 and 4.2. □

Thus, the space needed by our algorithm to simulate t steps of computation of a P-system, having description of length k, is exponential in both t and k in the worst case. This result allows us to show some relations between complexity classes for P-systems and Turing machines.

Theorem 4.4. *For each proper complexity function $f : \mathbb{N} \to \mathbb{N}$ we have*

$$\mathbf{MC}_{\mathcal{EAM}}(f(n)) \subseteq \mathbf{SPACE}(2^{O(f(n)^2 + f(n) \log p(n))})$$

for some polynomial p.

Proof. Let $L \in \mathbf{MC}_{\mathcal{EAM}}(f(n))$. Then, there exists a DTM M which, on input $x \in \Sigma^*$ of length n, constructs the associated confluent recognizer P-system Π_x of type $\mathcal{EAM}$ in polynomial time (w.r.t. n); the latter, in time $f(n)$, decides whether $x \in L$ by emitting the object *yes* or *no*.

Let M' be the composition of M and our simulation algorithm. That is, M' on input x simulates M in order to obtain the description of Π_x; this requires $p(n)$ time, for some polynomial p. Furthermore, the length of the description is bounded by $p(n)$ itself. Then, M' executes the simulation algorithm on Π_x.

From lemma 4.3, the space required to simulate the $f(n)$ steps of computation is

$$\mathrm{cfg}(f(n)) = 2^{O(f(n)^2 + f(n) \log p(n))},$$

thus $L \in \mathbf{SPACE}(2^{O(f(n)^2 + f(n) \log p(n))})$. □

Corollary 4.5. $\mathbf{PMC}_{\mathcal{EAM}} \subseteq \mathbf{EXPSPACE}$.

Proof. It follows immediately from theorem 4.4, from the fact that the complexity function is now polynomial and from the definition of **EXPSPACE**, which is $\mathbf{SPACE}(2^{p(n)})$ for every polynomial p. □

4.3.2. *Space analysis for general P-systems with active membranes*

Let us consider, now, the simulation of P-systems which admit division rules for non-elementary membranes. Those rules can cause the duplication of each membrane with neutral polarization placed in the involved region: the greater the depth of the membrane structure, the greater the size of the subtrees to duplicate in the worst case.

To prove the following lemma, it is useful to consider a tree independently of the membrane structure it represents. Let μ_0 be a tree of k nodes arranged in a chain. We duplicate the only leaf (the node of level $k-1$), joining the copy to the node of level $k-2$ by a new edge. Now, we duplicate the node of level $k-2$ together with its subtree, joining the new subtree to the node of level $k-3$. We

repeat the process at each level, root level excluded. It is easy to see that, after these operations, we obtain a full binary tree μ_1.

Repeating the entire process from the leaves up to the root, we obtain a full quaternary tree μ_2. In general, after t iterations, we get a full 2^t-ary tree μ_t. The total number of nodes in μ_t is

$$\sum_{i=0}^{k-1}(2^t)^i = \frac{2^{kt}-1}{2^t-1} \leq 2^{kt} \tag{2}$$

for each $t > 0$.

The scenario we just described is only a hypothetical worst case, since it can never actually happen in practical terms: the application of division rules requires some conditions which cannot be satisfied by all membranes simultaneously (e.g. positively charged membranes are required in order to allow non-elementary membrane division, but they are not duplicated). Nonetheless, we obtained an upper bound on the number of nodes in the structural tree.

Lemma 4.6. *Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R)$ be a confluent recognizer P-system with active membranes; let k be the length of the description of Π. Then $\deg(t) = 2^{O(kt)}$.*

Proof. In the worst case, μ is a chain of at most k nodes:

$$\deg(0) \leq k \leq 2^k.$$

By applying non-elementary division rules to each internal and elementary division rules to each leaf in the structural tree at time t, we obtain a structure whose degree is bounded by equation (2):

$$\deg(t) \leq 2^{kt} \leq 2^{k(t+1)}$$

for each $t > 0$. Thus, $\deg(t) \leq 2^{k(t+1)} = 2^{O(kt)}$ for each $t \in \mathbb{N}$. □

Lemma 4.7. *Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R)$ be a confluent recognizer P-system with active membranes; let k be the length of the description of Π. Then $\mathrm{cfg}(t) = 2^{O(t^2k)}$.*

Proof. It follows immediately from equation (1) and from lemmata 4.1 and 4.6. □

The proofs of the two following propositions are analogous to those of theorem 4.4 and corollary 4.5.

Theorem 4.8. *For each proper complexity function $f : \mathbb{N} \to \mathbb{N}$ we have*

$$\mathbf{MC}_{\mathcal{AM}}(f) \subseteq \mathbf{SPACE}(2^{O(f(n)^2 p(n))})$$

for some polynomial p.

Corollary 4.9. $\mathbf{PMC}_{\mathcal{AM}} \subseteq \mathbf{EXPSPACE}$.

Note that, while it may be possible to reduce the storage required for the representation of objects in a region by using a binary coding, the total storage is still affected by the number of membranes: exponential space seems to be unavoidable for our representation of P-systems.

4.4. Time complexity analysis

The time we need to simulate confluent recognizer P-systems with active membranes need not be more than *doubly* exponential (i.e. $O(2^{2^{f(n)}})$) in the number of steps, since $\mathbf{SPACE}(f(n)) \subseteq \mathbf{TIME}(2^{O(f(n))})$. We can improve on this; in fact, our simulation algorithm needs only exponential time.

To prove the previous statement, we need to analyze the most expensive combination of rules to simulate during each step. As usual, we consider a P-system Π with active membranes having description of length k, analyzing separately the cases in which Π does or does not make use of non-elementary membrane division.

4.4.1. *Time analysis for P-systems with elementary active membranes*

We already analyzed the time required to apply a single developmental rule in section 4.2. The most expensive rules to simulate for P-systems with elementary active membranes are object evolution and elementary membrane division (dissolution is expensive too, but it reduces the degree of the system). The former are applicable in each node of the tree, while the latter only in the leaves.

For the sake of worst case analysis, let us consider again the P-system with a membrane structure of degree m and depth 1, having a description of length k. We already showed that, at time t, there are $2^{O(t+\log k)}$ nodes in the structural tree. In each of them, almost all objects can evolve (those which take part in division rules are excluded), i.e. $2^{O(t^2+t\log k)}$. Furthermore, each membrane but the skin can evolve.

Let us recall that the application (or application failure) of an object evolution rule requires $O(k \cdot \mathrm{obj}(t) \cdot \deg(t))$ time. In order to apply the maximum possible number of such rules in a single membrane, the procedure ApplyEvolutionRule is called repeatedly with each of the $O(k)$ rules, until it returns $false$ (this means that the rule is not applicable anymore). Since the objects which can evolve during step t are $O(\mathrm{obj}(t))$, the total time required to apply evolution rules is $O(k^2 \cdot \mathrm{obj}(t)^2 \cdot \deg(t))$.

With regard to the application of division rules, a scan of the list of rules is required too. Taking rule application failures into account, the time needed is $O(k^2 \cdot \mathrm{obj}(t))$.

All these operations must be performed in each of the $\deg(t)$ membranes of the structure; thus, the total time to simulate step t of computation is

$$\begin{aligned} \mathrm{step}(t) &= O(\deg(t) \cdot (k^2 \cdot \mathrm{obj}(t)^2 \cdot \deg(t) + k^2 \cdot \mathrm{obj}(t))) \\ &= O(k^2 \cdot \mathrm{obj}(t)^2 \cdot \deg(t)^2) \\ &= 2^{O(t^2+t\log k)}. \end{aligned} \tag{3}$$

Lemma 4.10. *Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R)$ be a confluent recognizer P-system with elementary active membranes; let k be the length of the description of Π. Suppose that Π operates in time $f(n)$. Then, the algorithm takes $2^{O(f(n)^2 + f(n)\log k)}$ time to simulate a computation of Π.*

Proof. We must sum $\mathrm{step}(t)$ from equation (3) for $t = 0, \ldots, f(n)$:

$$\begin{aligned}\sum_{t=0}^{f(n)} \mathrm{step}(t) &\leq (f(n)+1)\cdot \mathrm{step}(f(n)) \\ &= 2^{\log(f(n)+1)} \cdot 2^{O(f(n)^2 + f(n)\log k)} \\ &= 2^{O(f(n)^2 + f(n)\log k)}.\end{aligned}$$

□

Theorem 4.11. *For each proper complexity function $f : \mathbb{N} \to \mathbb{N}$ we have*

$$\mathbf{MC}_{\mathcal{EAM}}(f(n)) \subseteq \mathbf{TIME}(2^{O(f(n)^2 + f(n)\log p(n))})$$

for some polynomial p.

Proof. It follows from lemma 4.10, keeping in mind that the size of the description of each P-system is polynomial in the length of the input string. □

Corollary 4.12. $\mathbf{PMC}_{\mathcal{EAM}} \subseteq \mathbf{EXP}$.

4.4.2. *Time analysis for general P-systems with active membranes*

With regard to P-systems with active membranes and with non-elementary membrane division, the most expensive rule combination consists of evolution for each object and division of each membrane (but the skin).

The growth of the structural tree is maximal when the initial membrane structure is a chain of nodes: the degree after t steps is $2^{O(kt)}$ (lemma 4.6), and each region contains $2^{O(t^2 k)}$ objects (lemma 4.1).

The time required to apply evolution rules in a single membrane is still $O(k^2 \cdot \mathrm{obj}(t)^2 \cdot \deg(t))$ (see previous section). Non-elementary membrane division rules selection requires a scan of the $O(k)$ rules to identify an applicable rule; since the application (or failure of the application) of such a rule requires $O(k \cdot \deg(t)^2 \cdot \mathrm{obj}(t))$ time, the total is $O(k^2 \cdot \deg(t)^2 \cdot \mathrm{obj}(t))$.

These operations must be performed for each membrane at time t, and this gives

$$\begin{aligned}\mathrm{step}(t) &= O(\deg(t) \cdot (k^2 \cdot \mathrm{obj}(t)^2 \cdot \deg(t) + k^2 \cdot \deg(t)^2 \cdot \mathrm{obj}(t))) \\ &= O(k^2 \cdot \deg(t)^3 \cdot \mathrm{obj}(t)^2) \\ &= 2^{O(t^2 k)}.\end{aligned} \tag{4}$$

Lemma 4.13. *Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R)$ be a confluent recognizer P-system with active membranes; let k be the length of the description of Π. Suppose that*

Π *operates in time* $f(n)$*. Then, the simulation of a computation of* Π *requires* $2^{O(f(n)^2 k)}$ *time.*

Proof. We only need to sum the time required to simulate each step, from equation (4):

$$\begin{aligned}\sum_{t=0}^{f(n)} \text{step}(t) &\leq (f(n)+1)\cdot \text{step}(f(n)) \\ &= 2^{\log(f(n)+1)} \cdot 2^{O(f(n)^2 k)} \\ &= 2^{O(f(n)^2 k)}.\end{aligned}$$

□

Theorem 4.14. *For each proper complexity function* $f : \mathbb{N} \to \mathbb{N}$ *we have*

$$\mathbf{MC}_{\mathcal{AM}}(f(n)) \subseteq \mathbf{TIME}(2^{O(f(n)^2 p(n))})$$

for some polynomial p*.*

Proof. Analogous to the proof of theorem 4.11. □

Corollary 4.15. $\mathbf{PMC}_{\mathcal{AM}} \subseteq \mathbf{EXP}$*.*

From theorem 3.11 and this last corollary we obtain the following result.

Corollary 4.16. $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{AM}} \subseteq \mathbf{EXP}$*.*

5. Conclusions

We presented new results concerning the comparison of complexity classes defined within the framework of membrane computing and standard complexity classes for Turing machines: **PSPACE**, **EXP**, and **EXPSPACE**.

In particular, we have shown that the class $\mathbf{PMC}_{\mathcal{AM}}$, that is, the class of languages accepted by P–systems with active membranes working in polynomial time is included in the class **EXP**. This proves the importance of the class $\mathbf{PMC}_{\mathcal{AM}}$, as it is an intermediate complexity class between **PSPACE** and **EXP**.

An open problem is to determine if $\mathbf{PMC}_{\mathcal{AM}}$ characterizes the class **PSPACE** or **EXP** (or both), or which of the previous inclusions are proper; an answer to such questions could also give an answer to the important open question whether or not $\mathbf{EXP} = \mathbf{PSPACE}$.

We plan to continue our investigations by focusing on the relations of complexity classes for non-deterministic P–systems with active membranes. We conjecture that such systems working in polynomial time can be simulated by non-deterministic Turing machines working in exponential time. This would allow us to prove the following inclusions: $\mathbf{NPMC}_{\mathcal{AM}} \subseteq \mathbf{NEXP} \subseteq \mathbf{EXPSPACE}$.

This work has been supported by the Italian Ministry of University (MIUR), under project PRIN-04 “Systems Biology: modellazione, linguaggi e analisi (SYBILLA)”.

References

[1] A. Alhazov, C. Martin-Vide, L. Pan, Solving a PSPACE–complete problem by P–systems with restricted active membranes, *Fundamenta Informaticae*, 58, 2(2003), 67-77.

[2] M. Gutierrez-Naranjo, M.J. Perez-Jimenez, P–systems with active membranes, without polarizations and without dissolution: a characterization of P, in vol. *Unconventional Computation, 4th International Conference, UC 2005*, C.S. Calude, M.J. Dinneen, Gh. Paun, M.J. Perez-Jimenez, G. Rozenberg, eds., LNCS 3699, Springer-Verlag, Berlin-Heidelberg, 2005, 105-116.

[3] M. Gutierrez-Naranjo, M.J. Perez-Jimenez, A.Riscos-Nunez, F.J. Romero-Campero, Characterizing Standard Tractability by Cell-like Membrane Systems, to appear.

[4] 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 (1999).

[5] C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[6] Gh. Păun, Computing with membranes, *J. of Computer and System Sciences*, 61, 1 (2000), 108-143 (see also TUCS Research Report No 208, November 1998 http://www.tucs.fi).

[7] Gh. Păun, P-systems with active membranes: attacking NP complete problems, in vol. *Unconventional Models of Computation*, I. Antoniou, C.S. Calude, M.J. Dinneen, eds., Springer-Verlag, London, 2000, 94-115 (see also CDMTCS Research report No. 102, 1999, Auckland Univ., New Zeland, www.cs.auckland.ac.nz/CDMTCS).

[8] G. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.

[9] M.J. Perez-Jimenez, A. Romero-Jimenez, F. Sancho-Caparrini, Complexity Classes in Cellular Computing with Membranes, Rovira i Virgili Univ., Tech. Rep. No. 26, M. Cavaliere, C. Martin-Vide, Gh. Păun (Eds), Brainstorming Week on Membrane Computing; Tarragona, Feb 5-11 2003, 270-278, and *Natural Computing*, 2 (3), 265-285, Aug 2003.

[10] M.J. Perez-Jimenez, A. Romero-Jimenez, F. Sancho-Caparrini, The P versus NP problem through cellular computing with membranes, *Aspects of Molecular Computing*, LNCS 2950, Springer, 2004, 338-352.

[11] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.

[12] P. Sosik, The computational power of cell division in P–systems: Beating down parallel computers?, *Natural Computing*, 2, 3(2003), 287-298.

[13] C. Zandron, C. Ferretti, G. Mauri, Solving NP-Complete Problems Using P Systems with Active Membranes, in vol. *Unconventional Models of Computation*, I. Antoniou, C.S. Calude, M.J. Dinneen, eds., Springer-Verlag, London, 2000, 289-301.

Communicated by (The editor will be set by the publisher).