Fundamenta Informaticae ? (2014) 1001–1014
DOI 10.3233/FI-2012-0000
IOS Press

Membrane Division, Oracles, and the Counting Hierarchy

Alberto Leporati, Luca Manzoni, Giancarlo Mauri,

Antonio E. Porreca, Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione

Università degli Studi di Milano-Bicocca

Viale Sarca 336/14, 20126 Milano, Italy

{*leporati*, *luca.manzoni*, *mauri*, *porreca*, *zandron*}*@disco.unimib.it*

Abstract. Polynomial-time P systems with active membranes characterise $\mathbf{PSPACE}$ by exploiting membranes nested to a polynomial depth, which may be subject to membrane division rules. When only elementary (leaf) membrane division rules are allowed, the computing power decreases to $\mathbf{P}^{\mathbf{PP}} = \mathbf{P}^{\#\mathbf{P}}$, the class of problems solvable in polynomial time by deterministic Turing machines equipped with oracles for counting (or majority) problems. In this paper we investigate a variant of intermediate power, limiting membrane nesting (hence membrane division) to constant depth, and we prove that the resulting P systems can solve all problems in the counting hierarchy $\mathbf{CH}$, which is located between $\mathbf{P}^{\mathbf{PP}}$ and $\mathbf{PSPACE}$. In particular, for each integer $k \geq 0$ we provide a lower bound to the computing power of P systems of depth k.

Keywords: Membrane computing, counting complexity, oracles

1. Introduction

P systems with active membranes [5] are parallel, nondeterministic devices inspired by the functioning and internal structure of biological cells, in particular by their hierarchical nesting of membranes. This feature, together with the ability to generate exponentially many membranes in polynomial time by *membrane division*, allows them to solve computationally hard problems efficiently, by *trading space for time*. It is known that P systems of polynomial (actually, linear) nesting depth characterise $\mathbf{PSPACE}$ in polynomial time [1, 9]. On the other hand, P systems of membrane nesting depth 1 (which, in particular, use only *elementary division rules*, i.e., division rules limited to membranes not containing further membranes) are known to solve exactly the problems in $\mathbf{P}^{\mathbf{PP}} = \mathbf{P}^{\#\mathbf{P}}$ [7, 3].

In an attempt to establish the importance of nesting depth for the efficiency of P systems, as initiated by Porreca and Murphy [8] for P systems with active membranes without charges, in this paper we analyse P systems using both elementary and *non-elementary* membrane division, but we limit the depth of the P system to a constant, independent of the input size. The results show that constant-depth P systems solve the problems in the *counting hierarchy* $\mathbf{CH}$ in polynomial time and, more precisely, that depth-k P systems solve the problems in $\mathbf{P}^{\mathbf{C}_k\mathbf{P}}$ in polynomial time, where $\mathbf{C}_k\mathbf{P}$ is the k-th level of the hierarchy; this generalises the result already proved for unitary nesting depth [7]. In particular, we show a way of trading membrane nesting depth for oracle power: by increasing the former, Turing machines with oracles for problems higher in the counting hierarchy can be simulated.

After recalling or introducing a few basic notions (Section 2), we describe the techniques we employ: simulating a nonstandard number of charges (Section 3), a new, single-membrane simulation of polynomial-space Turing machines and the use of P systems with oracle membranes (Section 4), and a way to eliminate them by increasing the membrane nesting depth (Section 5, which includes our main results). We conclude the paper with a short summary and future work (Section 6).

2. Basic notions

In this paper we employ the standard definition and notation for P systems with active membranes [5, 7]. Here we just briefly recall that the available rule types are object evolution $[a \to w]^\alpha_h$, send-in communication $a\ [\ ]^\alpha_h \to [b]^\beta_h$, send-out communication $[a]^\alpha_h \to [\ ]^\beta_h\ b$, membrane dissolution $[a]^\alpha_h \to b$ (not used in this paper), elementary membrane division $[a]^\alpha_h \to [b]^\beta_h\ [c]^\gamma_h$, and (strong) non-elementary division

$$\left[\left[\ \right]^+_{h_1} \cdots \left[\ \right]^+_{h_k} \left[\ \right]^-_{h_{k+1}} \cdots \left[\ \right]^-_{h_m}\right]^\alpha_h \to \left[\left[\ \right]^\delta_{h_1} \cdots \left[\ \right]^\delta_{h_k}\right]^\beta_h \left[\left[\ \right]^\zeta_{h_{k+1}} \cdots \left[\ \right]^\zeta_{h_m}\right]^\gamma_h$$

The class of P systems with active membranes using the above types of rules is denoted by $\mathcal{AM}$.

Sometimes another variant of non-elementary division rules, called *weak non-elementary division rules* [11], are employed for P systems with active membranes without charges. In this paper we will employ them for P systems with extended charges (Section 3), in the form $[a]^\alpha_h \to [b]^\beta_h\ [c]^\gamma_h$. These rules are triggered by an object instead of membranes.

Families $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of *recogniser* P systems can be used to decide languages $L \subseteq \Sigma^\star$. Each P system Π_x decides the membership of x in L, sends out from the outermost membrane an object yes or no and halts; we also require *confluence*, i.e., that all nondeterministically generated computations of Π_x agree on the result. In this paper we use *polynomial-time uniform* families of P systems [4, 6], where Π_x is constructed in polynomial time from a P system Π_n, common for all strings of length n, and an input multiset w_x encoding the actual input string.

The class of decision problems solved in polynomial time by uniform families of P systems with active membranes is denoted by $\mathbf{PMC}_{\mathcal{AM}}$; the corresponding class for polynomial space is denoted by $\mathbf{PMCSPACE}_{\mathcal{AM}}$. When the membrane nesting depth is at most k, i.e., the membrane structure is a tree of height at most k, the symbol $\mathcal{AM}$ is replaced by $\mathcal{AM}(k)$. We also define the class of P systems with active membranes of constant depth as $\mathcal{AM}(O(1)) = \bigcup_{k\in\mathbb{N}} \mathcal{AM}(k)$.

For the basic notions of computational complexity and, in particular, counting complexity, we refer the reader to [2]. We recall that a *counting Turing machine* (also known as *probabilistic* or *majority* Turing machine) is a nondeterministic Turing machine M with the following acceptance condition: a string $x \in \Sigma^\star$ is accepted if and only if the majority of the computations of M on input x are accepting.

The class of problems solved in polynomial time by counting Turing machines is denoted by $\mathbf{PP}$. The following lemma shows that we can always assume that counting Turing machines have an odd number of computations on every input x.

Lemma 2.1. If $L \subseteq \Sigma^\star$ is decided by a counting Turing machine M in time $t(n)$, then it is decided by a counting Turing machine *having an odd number of computations* on each input x and working in time $\Theta(t(n))$.

Proof:
Suppose M has $c(x)$ computations on input x, and $a(x)$ of them are accepting. We construct another counting machine M' having $2c(x) + 1$ computations, with $2a(x)$ accepting ones. The machine M' behaves as follows: on input x, nondeterministically choose a bit i; if i is 0, then reject; otherwise, nondeterministically choose another bit, ignore it, and simulate M on input x. Then, we have $x \in L$ if and only if

$$a(x) > \frac{c(x)}{2} \iff 2a(x) > c(x) \iff 2a(x) > c(x) + \frac{1}{2} = \frac{2c(x)+1}{2},$$

where the last implication holds because both $2a(x)$ and $c(x)$ are integers. As a consequence, we have $x \in L(M')$ if and only if $x \in L$. □

We can define increasingly stronger complexity classes by using *oracle Turing machines*. In particular, by $\mathbf{P}^L$ (resp., $\mathbf{PP}^L$) we denote the class of problems solvable in polynomial time by deterministic (resp., counting) Turing machines with an oracle for a language $L \subseteq \Sigma^\star$. If $\mathbf{X}$ is a complexity class, then $\mathbf{P}^\mathbf{X}$ is defined as $\bigcup_{L \in \mathbf{X}} \mathbf{P}^L$; analogously, we have $\mathbf{PP}^\mathbf{X} = \bigcup_{L \in \mathbf{X}} \mathbf{PP}^L$. Finally, the *counting hierarchy* is defined as $\mathbf{CH} = \bigcup_{k \in \mathbb{N}} \mathbf{C}_k\mathbf{P}$, where $\mathbf{C}_0\mathbf{P} = \mathbf{P}$ and $\mathbf{C}_{k+1}\mathbf{P} = \mathbf{PP}^{\mathbf{C}_k\mathbf{P}}$ for all $k \in \mathbb{N}$. We have $\mathbf{P} = \mathbf{C}_0\mathbf{P} \subseteq \cdots \subseteq \mathbf{C}_k\mathbf{P} \subseteq \cdots \subseteq \mathbf{CH} \subseteq \mathbf{PSPACE}$, although no inclusion is known to be proper.

In order to highlight the relationship between membrane nesting depth and oracle power (for Turing machines) we introduce the notion of P systems with oracles.

Definition 2.2. Let L be a multiset language (i.e., a set of multisets). *A P system with active membranes with an oracle for* L is a P system with active membranes with a distinguished label h identifying an *oracle membrane*, which must be an *elementary* membrane. Only send-in rules can be applied to an oracle membrane (it is "write only"), it can only have neutral or positive charge, and it can be *queried* by first setting its charge to positive, sending in further objects without changing the charge, and finally resetting it to neutral. In the next time step, membrane h sends out either yes_L or no_L, depending on whether the multiset contained in h belongs to L. The oracle membrane is simultaneously emptied, thus allowing further queries. Further oracle membranes h can be created by non-elementary division of an ancestor membrane during the computation of Π, allowing multiple parallel queries.

We establish the convention that oracle membranes do not contribute to the depth of the membrane structure. For instance, a single membrane containing an oracle membrane has depth 0 instead of 1. This convention allows us to express the statements of the results of this paper in a simpler way.

In order to employ P systems with oracles for decision problems, which are usually defined in terms of strings rather than multisets, we define an encoding of languages as multiset languages that preserve the ordering of the symbols.

Definition 2.3. Let $L \subseteq \Sigma^\star$ be a (string) language. We define the corresponding *multiset language* over the alphabet $\{\sigma_i : \sigma \in \Sigma, i \in \mathbb{N}\}$ as $\widehat{L} = \{s(x) : x \in L\}$, where $s(x_0 \cdots x_{n-1}) = x_{0,0} \cdots x_{n-1,n-1}$ is the function indexing each symbol of x with its position.

Observe that, even if the alphabet of $\widehat{L}$ is, in general, infinite, each P system working in polynomial time never needs more than a polynomial amount of subscripted symbols, since the query multiset is assembled one object at a time. In the rest of the paper we will write $\widehat{L}$ as L whenever it is clear whether we are referring to a string language or the corresponding multiset language.

We denote the class of problems solved in polynomial time by P systems with active membranes (resp., of nesting depth k) with an oracle for L by $\mathbf{PMC}^L_{\mathcal{AM}}$ (resp., $\mathbf{PMC}^L_{\mathcal{AM}(k)}$).

3. Simulating a polynomial number of charges

In Sections 4 and 5 we will describe several simulations of Turing machines by means of P systems. These simulations involve a large number of auxiliary rules and their corresponding intermediate computation steps; these can be partially simplified if we allow a polynomial number of charges (which we will refer to as "extended charges") instead of the usual three. As a matter of fact, we can prove a technical lemma showing that a membrane using extended charges can be simulated by a standard one, as long as it is not interacting with other membranes, with a multiplicative logarithmic slowdown. Furthermore, we also implement the nonstandard rule type $[a]^\alpha_h \to [b]^\beta_h$, a single-object evolution rule that changes the membrane charge.

This lemma allows us to use P systems with extended charges as a shorthand for (more complicated) standard P systems. To distinguish the two views of the same system, we will call the former an "high-level view", and we will depict their membranes with a darker background. When working with extended charges and rules, we will also say that we are in "high-level mode", whereas working with standard charges and rules will be referred to as "low-level mode" (corresponding to a white background).

Lemma 3.1. Let h be a membrane using $p(n)$ charges (for some polynomial p) and the following types of rules: object evolution $[a \to w]^\alpha_h$, single-object evolution with charge changing $[a]^\alpha_h \to [b]^\beta_h$, and elementary or weak non-elementary division $[a]^\alpha_h \to [b]^\beta_h\ [c]^\gamma_h$. Suppose no object reaches h from its children membranes (i.e., no send-out or dissolution rules are applied to them), the children membranes remain neutrally charged, the rules are applied in a deterministic way, and exactly one rule of types $[a]^\alpha_h \to [b]^\beta_h$ or $[a]^\alpha_h \to [b]^\beta_h\ [c]^\gamma_h$ is applied per step. Then, it is possible to simulate membrane h with standard rules and using only three charges; each step of h is simulated in $\Theta(\log n)$ steps.

Proof:
A configuration of a membrane h using an extended charge α (on the left) is implemented as a configuration of a standard membrane h having neutral charge and having every object subscripted by α (on the right).

This way, each object stores enough information to simulate the extended charge α; we only need to ensure that the subscripts are updated consistently across all objects, according to the unique, deterministically chosen rule which may change the charge of h. Without loss of generality, we can assume that

each extended charge α has the form $(\alpha_0, \ldots, \alpha_{k-1}) \in \{+,-\}^k$ with $k = \lceil \log p(n) \rceil$, i.e., a string over $\{+,-\}$ long enough to represent $p(n)$ different values.

First of all, each extended evolution rule $[a \to w_1 \ \cdots \ w_\ell]_h^\alpha$ is replaced by the rule

$$[a_\alpha \to w'_1 \ \cdots \ w'_\ell]_h^0$$

The objects on the right-hand side will be eventually rewritten as $w_{1,\beta}, \ldots, w_{\ell,\beta}$, where β is the (extended) charge of h in its next configuration, as described below.

An extended rule of the form $[a]_h^\alpha \to [b]_h^\beta$ is implemented by the following standard rules:

$$[a_\alpha \to b'_\beta]_h^0 \tag{1}$$

$$[b'_\beta \to b' \ \beta_{0,0} \ \cdots \ \beta_{k-1,k-1} \ 0_k]_h^0 \qquad \text{where } (\beta_0, \ldots, \beta_{k-1}) = \beta \tag{2}$$

$$[\beta_{i,0}]_h^\delta \to [\ ]_h^{\beta_i} \ \# \qquad \text{for } i \in [0, k-1] \text{ and } \beta_i, \delta \in \{+, 0, -\} \tag{3}$$

$$[\beta_{i,j} \to \beta_{i,j-1}]_h^\delta \qquad \text{for } i \in [0, k-1],\ \beta_i, \delta \in \{+, 0, -\}, \text{ and } j \in [1, k-1] \tag{4}$$

$$[0_0]_h^\delta \to [\ ]_h^0 \ \# \qquad \text{for } \delta \in \{+, 0, -\} \tag{5}$$

$$[0_j \to 0_{j-1}]_h^\delta \qquad \text{for } \delta \in \{+, 0, -\} \text{ and } j \in [1, k] \tag{6}$$

Informally, the object a_α is rewritten into b', and then the new charge $\beta = (\beta_0, \ldots, \beta_{k-1})$ is written in the charge of h sequentially, one symbol $+$ or $-$ at a time, finally followed by 0 (represented by 0_k in rule (2)). This sequence of operations requires $\Theta(k) = \Theta(\log n)$ steps, and allows the remaining objects in h to read and store β as a subscript, using rules of type (7).

An extended elementary or weak non-elementary membrane division rule $[a]_h^\alpha \to [b]_h^\beta \ [c]_h^\gamma$ is implemented similarly, except that, instead of rule (1), we have

$$[a_\alpha]_h^0 \to [b'_\beta]_h^0 \ [c'_\gamma]_h^0$$

and the rules (2)–(6) are also repeated for the object c'_γ.

An extended weak non-elementary division rule $[a]_h^\alpha \to [b]_h^\beta \ [c]_h^\gamma$ can be also implemented in terms of elementary division and strong non-elementary division rules by having an additional, neutrally charged membrane h' placed inside h, and replacing (1) with the following rules:

$$a_\alpha \ [\ ]_{h'}^0 \to [a'_\alpha]_{h'}^0 \qquad [a'_\alpha]_{h'}^0 \to [b_\beta]_{h'}^+ \ [c_\gamma]_{h'}^- \qquad \left[[\ ]_{h'}^+ \ [\ ]_{h'}^-\right]_h^0 \to \left[[\ ]_{h'}^0\right]_h^0 \ \left[[\ ]_{h'}^0\right]_h^0$$

$$[b_\beta]_{h'}^0 \to [\ ]_{h'}^0 \ b'_\beta \qquad [c_\gamma]_{h'}^0 \to [\ ]_{h'}^0 \ c'_\gamma$$

Now rule (2) is triggered, for both b'_β and c'_γ, and the objects contained in the two copies of h acquire the subscripts β and γ, respectively, as described below.

Finally, if an object a has no rule activated by the extended charge α, we add a (seemingly useless) extended evolution rule $[a \to a]_h^\alpha$, which is needed in order to update (in low-level mode) the extended charge stored in the subscript of a.

By hypothesis, exactly one rule of the form $[a]_h^\alpha \to [b]_h^\beta$ or $[a]_h^\alpha \to [b]_h^\beta \ [c]_h^\gamma$ is applied per computation step, and writes the extended charge on the right-hand side one bit at a time. The "primed" objects σ' contained in h simultaneously store these symbols as a string in their subscript:

$$[\sigma'_{(\beta_0,\ldots,\beta_{i-1})} \to \sigma'_{(\beta_0,\ldots,\beta_{i-1},\beta_i)}]_h^{\beta_i} \qquad \text{for } \sigma \in \Gamma,\ i \in [0, k-1],\ j \in [0, i],\ \beta_j \in \{+, -\} \tag{7}$$

When the final object 0_k is sent out, restoring the charge of h to neutral, all objects in h have stored the new charge $\beta = (\beta_0, \ldots, \beta_{k-1})$ in their subscript, and lose the prime:

$$[\sigma'_\beta \to \sigma_\beta]^0_h \qquad \text{for } \sigma \in \Gamma,\ \beta \in \{+,-\}^k$$

The simulation of the next step may now begin. □

4. Single-membrane P systems with oracles

We now describe a new simulation of deterministic polynomial-time Turing machines, using a single membrane and with a polynomial slowdown. This result proves that polynomial-time P systems of depth-0 characterise $\mathbf{P}$.

Lemma 4.1. $\mathbf{PMC}_{\mathcal{AM}(0)} = \mathbf{P}$.

Proof:
Since the outermost membrane can never divide, depth-0 P systems do not use membrane division rules. Hence, by the Milano Theorem [10], we have $\mathbf{PMC}_{\mathcal{AM}(0)} \subseteq \mathbf{P}$.

Let $L \in \mathbf{P}$, and let M be a single-tape deterministic Turing machine working in polynomial time (hence, space) $p(n)$. We define a uniform family $\mathbf{\Pi}_M = \{\Pi_{M,x} : x \in \Sigma^\star\}$ of depth-0 (i.e., single-membrane) P systems. In order to do so, we exploit Lemma 3.1, which allows us to use a polynomial number of charges and extended rules.

Suppose that, in the current configuration, the machine M on input x is in state q, the tape head is located on cell $i \in [0, p(n)]$, and the tape contains the string $w = \sigma_0 \cdots \sigma_{p(n)}$, including trailing blanks. Assuming, for instance, that $w = abba\sqcup\sqcup$, the corresponding configuration of $\Pi_{M,x}$ is

$$\boxed{a_0 \;\; b_1 \;\; b_2 \;\; a_3 \;\; \sqcup_4 \;\; \sqcup_5}^{(q,i)}_M$$

where the charge stores state and position of M, and the symbols in w correspond to objects indexed by their position in the string.

For each non-final state q and each tape symbol a, if $\delta(q, a) = (r, b, d)$, with $d \in \{-1, 0, +1\}$, the P system has the following rule schema:

$$[a_i]^{(q,i)}_M \to [b_i]^{(r,i+d)}_M \qquad \text{for } i \in [0, p(n)] \tag{8}$$

which updates the relevant portion of the configuration of $\Pi_{M,x}$ according to the transition of M. Notice that only one rule of type (8) can be applied at a time, since the index i uniquely identifies an object.

If the machine reaches an accepting state q, then the extended object σ_0 (for σ ranging over the tape alphabet), which corresponds to the first symbol of the tape of M and, in low-level view, is actually $\sigma_{0,(q,i)}$ for some i, is rewritten into yes_M:

$$[\sigma_{0,(q,i)} \to \mathsf{yes}_M]^0_M \qquad \text{for } i \in [0, p(n)] \tag{9}$$

If M is instead in a rejecting state, one of the following rules is applied:

$$[\sigma_{0,(q,i)} \to \mathsf{no}_M]^0_M \qquad \text{for } i \in [0, p(n)] \tag{10}$$

At the same time, the remaining objects are deleted by rewriting them into the empty multiset:

$$[\sigma_{j,(q,i)} \to \epsilon]^0_M \qquad \text{for } i \in [0, p(n)],\ j \in [1, p(n)]$$

In the next step, the result object yes_M (resp., no_M) is sent out; this also changes the charge of the membrane to positive (resp., negative), as this will be useful in a later section.

$$[\mathsf{yes}_M]^0_M \to [\ ]^+_M\ \mathsf{yes}_M \qquad\qquad [\mathsf{no}_M]^0_M \to [\ ]^-_M\ \mathsf{no}_M \tag{11}$$

This halts the computation of $\Pi_{M,x}$, while producing the same output as M.

Hence, each $\Pi_{M,x}$ simulates M on input x with a polynomial slowdown. Furthermore, the family $\mathbf{\Pi}_M$ is polynomial-time uniform: the rule schemata are all generated from the transition table of M (which has constant size) by iterating indices over polynomial-size ranges of integers, there is only a single membrane, and the input multiset is computed by simply subscripting the symbols of x, padded with a polynomial number of blank symbols, with their position. This proves that $\mathbf{P} \subseteq \mathbf{PMC}_{\mathcal{AM}(0)}$. □

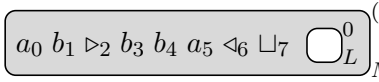
In fact, a stronger statement holds: Lemma 4.1 relativises to arbitrary oracles.

Theorem 4.2. $\mathbf{PMC}^L_{\mathcal{AM}(0)} = \mathbf{P}^L$ for all $L \subseteq \Sigma^\star$.

Proof:
We have $\mathbf{PMC}^L_{\mathcal{AM}(0)} \subseteq \mathbf{P}^L$ because we can simulate the external membrane (which cannot divide) as in the Milano Theorem [10], and simulate the oracle queries performed by the P systems with analogous queries of the Turing machines, after having removed the subscripts from the query multiset.

For the reverse inclusion, we only need to show how the simulation of Lemma 4.1 can be augmented in order to simulate the queries performed by an oracle machine M, working in polynomial time (hence, space) $p(n)$. Without loss of generality, we assume that M does not have a separate query tape, but instead delimits the query string y with the tape symbols $\triangleright$ and $\triangleleft$ before entering its query state $q_?$. In the next step, the machine will find itself in state q_L, if $y \in L$, or $\bar{q}_L$, if $y \notin L$, and the tape head will be re-positioned on cell 0. We also assume, in order to simplify the proof, that M only interrogates the oracle with query strings of length at least 2 (the results for the missing strings can be stored in a constant-size lookup table as part of the finite control of M).

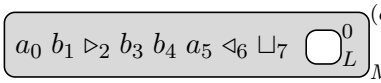
Suppose $\Pi_{M,x}$ simulates M on input x, and suppose M has just entered its query state $q_?$. Then, the P system has the following configuration, where as an example we assume the tape contains the string $ab\triangleright bba\triangleleft\sqcup$:



By means of the left delimiter object $\triangleright_{\ell-1}$, the index of the first symbol of the query string (namely, ℓ) is stored in the charge of M, using the rule

$$[\triangleright_{\ell-1}]^{(q_?,i)}_M \to [\triangleright_{\ell-1}]^{(q_?,i,\ell)}_M \qquad \text{for } i \in [0, p(n)],\ \ell \in [1, p(n)]$$

In our example, this leads to the following configuration:



Analogously, in the next time step the right delimiter object $\triangleleft_{r+1}$ stores the index of the last symbol of the query string (namely, r) in the charge of M:

$$[\triangleleft_{r+1}]_M^{(q_?,i,\ell)} \to [\triangleleft_{r+1}]_M^{(q_?,i,\ell,r)} \qquad \text{for } i \in [0,p(n)],\ \ell \in [1,p(n)],\ r \in [0,p(n)-1]$$

We thus obtain the configuration

$$\boxed{a_0\ b_1\ \triangleright_2\ b_3\ b_4\ a_5\ \triangleleft_6\ \sqcup_7\ \bigcirc_L^0}_M^{(q_?,i,3,5)}$$

The objects σ_j corresponding to the symbols of the query string (that is, those with $j \in [\ell, r]$) rewrite themselves; in the rest of the proof the symbol σ is implicitly quantified across the tape alphabet of the Turing machine.

$$[\sigma_j \to \sigma'_j\ \sigma_{j-\ell,j-\ell}]_M^{(q_?,i,\ell,r)} \qquad \text{for } i \in [0,p(n)],\ \ell \in [1,p(n)],\ r \in [0,p(n)-1],\ j \in [\ell, r-1]$$
$$[\sigma_r \to \sigma'_r\ \widetilde{\sigma}_{r-\ell,r-\ell}]_M^{(q_?,i,\ell,r)} \qquad \text{for } i \in [0,p(n)],\ \ell \in [1,p(n)],\ r \in [0,p(n)-1]$$

where σ'_j is "primed" in order to disable further applications of the rule, and $\sigma_{k,t}$ represents the k-th symbol of the query string, counting from 0; t is the number of steps to be waited before sending $\sigma_{k,t}$ into membrane L. The last symbol of the query string $\widetilde{\sigma}_{r-\ell,r-\ell}$ is also marked with a tilde.

Simultaneously, the remaining objects in membrane M are also primed; furthermore, the first object σ_0 (which is never part of the query string) also produces the object z, whose role will be described later. The corresponding rules are

$$[\sigma_j \to \sigma'_j]_M^{(q_?,i,\ell,r)} \qquad \text{for } i \in [0,p(n)],\ \ell \in [1,p(n)],\ r \in [0,p(n)-1],\ j \in [1,\ell-1] \cup [r+1,p(n)]$$
$$[\sigma_0 \to \sigma'_0\ \mathsf{z}]_M^{(q_?,i,\ell,r)} \qquad \text{for } i \in [0,p(n)],\ \ell \in [1,p(n)],\ r \in [0,p(n)-1]$$

which produce the configuration below, which is depicted both in high-level view and low-level view, with $\alpha = (q_?, i, 3, 5)$.

$$\boxed{\begin{array}{l} b_{0,0}\ b_{1,1}\ \widetilde{a}_{2,2}\ \mathsf{z} \qquad \bigcirc_L^0 \\ a'_0\ b'_1\ \triangleright'_2\ b'_3\ b'_4\ a'_5\ \triangleleft'_6\ \sqcup'_7 \end{array}}_M^{(q_?,i,3,5)} \qquad \boxed{\begin{array}{l} b_{0,0,\alpha}\ b_{1,1,\alpha}\ \widetilde{a}_{2,2,\alpha}\ \mathsf{z}_\alpha \qquad \bigcirc_L^0 \\ a'_{0,\alpha}\ b'_{1,\alpha}\ \triangleright'_{2,\alpha}\ b'_{3,\alpha}\ b'_{4,\alpha}\ a'_{5,\alpha}\ \triangleleft'_{6,\alpha}\ \sqcup'_{7,\alpha} \end{array}}_M^{0}$$

All the objects in M now rewrite themselves by deleting the subscript $\alpha = (q_?, i, \ell, r)$, and we proceed in low-level mode:

$$[\sigma'_{i,\alpha} \to \sigma'_i]_M^0 \qquad \text{for } i \in [0,p(n)],\ \alpha \in \{q_?\} \times [0,p(n)]^3$$
$$[\sigma_{k,t,\alpha} \to \sigma_{k,t}]_M^0 \qquad \text{for } k \in [0,p(n)-2],\ t = k,\ \alpha \in \{q_?\} \times [0,p(n)]^3$$
$$[\widetilde{\sigma}_{k,t,\alpha} \to \widetilde{\sigma}_{k,t}]_M^0 \qquad \text{for } k \in [0,p(n)-2],\ t = k,\ \alpha \in \{q_?\} \times [0,p(n)]^3$$
$$[\mathsf{z}_\alpha \to \mathsf{z}]_M^0 \qquad \text{for } \alpha \in \{q_?\} \times [0,p(n)]^3$$

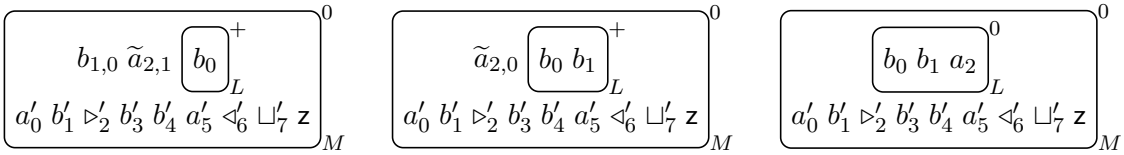
This leads to the configuration

$$\boxed{\begin{array}{l} b_{0,0}\ b_{1,1}\ \widetilde{a}_{2,2}\ \mathsf{z} \qquad \bigcirc_L^0 \\ a'_0\ b'_1\ \triangleright'_2\ b'_3\ b'_4\ a'_5\ \triangleleft'_6\ \sqcup'_7 \end{array}}_M^{0}$$

While the charge of M is neutral, only the objects $\sigma_{k,t}$ and $\widetilde{\sigma}_{k,t}$ have applicable rules. These objects are sent into L as σ_k, in the order given by the subscript k and using t as a timer; the first object to enter also sets the charge of L to positive, and the last one, the only one marked with the tilde, resets it to neutral, thus triggering the oracle query. The corresponding rules are

$$\begin{aligned}
&\sigma_{0,0}\ [\ ]^0_L \to [\sigma_0]^+_L && \\
&\sigma_{k,0}\ [\ ]^+_L \to [\sigma_k]^+_L && \text{for } k \in [1, p(n)-2] \\
&\widetilde{\sigma}_{k,0}\ [\ ]^+_L \to [\sigma_k]^0_L && \text{for } k \in [1, p(n)-2] \\
&[\sigma_{k,t} \to \sigma_{k,t-1}]^0_M && \text{for } k \in [1, p(n)-2],\ t \in [1, k] \\
&[\widetilde{\sigma}_{k,t} \to \widetilde{\sigma}_{k,t-1}]^0_M && \text{for } k \in [1, p(n)-2],\ t \in [1, k]
\end{aligned} \tag{12}$$

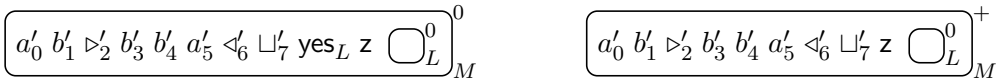
These rules produce the following sequence of configurations:



Resetting the charge of L to zero activates the oracle, which produces the result object yes_L (resp., no_L) inside membrane M, indicating that $y \in L$ (resp., $y \notin L$). Membrane L is also emptied in the process. The result object is then sent out from M as a "junk" object $\#$, while setting the charge to positive (resp., negative):

$$[\mathsf{yes}_L]^0_M \to [\ ]^+_M\ \# \qquad\qquad [\mathsf{no}_L]^0_M \to [\ ]^-_M\ \# \tag{13}$$

The corresponding configurations (assuming $y \in L$) are



When the charge of M becomes positive (resp., negative), the objects in M lose the prime and gain the subscript $(q_L, 0)$ (resp., $(\bar{q}_L, 0)$), denoting the transition of M to state q_L (resp., $\bar{q}_L$) and the resetting of the position of the tape head to the first cell. Simultaneously, the object z is sent out as $\#$, resetting the charge of M to neutral. The corresponding rules are

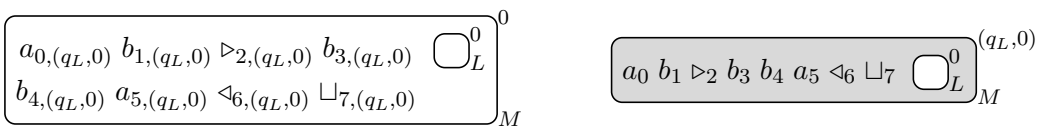
$$[\sigma'_j \to \sigma_{j,(q_L,0)}]^+_M \qquad \text{for } j \in [0, p(n)] \tag{14}$$

$$[\sigma'_j \to \sigma_{j,(\bar{q}_L,0)}]^-_M \qquad \text{for } j \in [0, p(n)] \tag{15}$$

$$[\mathsf{z}]^+_M \to [\ ]^0_M\ \# \tag{16}$$

$$[\mathsf{z}]^-_M \to [\ ]^0_M\ \# \tag{17}$$

The resulting configuration can also be seen as a high-level configuration (as shown below), which corresponds to the configuration of the Turing machine M after the oracle query.



Since performing the oracle query on $\Pi_{M,x}$ only incurs in a polynomial slowdown (due to the sequential input into membrane L) with respect to the same process on M, the statement of the lemma follows. □

Remark 4.3. The simulations of Lemma 4.1 and Theorem 4.2 do not only work for polynomial-time Turing machines, but more generally for polynomial-*space* ones. Therefore, for all $L \subseteq \Sigma^\star$ we have the inclusion $\mathbf{PSPACE}^L \subseteq \mathbf{PMCSPACE}^L_{\mathcal{AM}(0)}$, and in particular $\mathbf{PSPACE} \subseteq \mathbf{PMCSPACE}_{\mathcal{AM}(0)}$.

5. Trading membrane nesting depth for oracle power

We can now show how an oracle for a language in $\mathbf{PP}^L$ can be simulated by P systems with an oracle for L at the cost of a unitary increase in membrane nesting depth.

Lemma 5.1. $\mathbf{PMC}^{\mathbf{PP}^L}_{\mathcal{AM}(0)} \subseteq \mathbf{PMC}^L_{\mathcal{AM}(1)}$ for each $L \subseteq \Sigma^\star$.

Proof:
The simulation of deterministic Turing machines M with oracles of Theorem 4.2 can be easily adapted to simulate counting Turing machines with oracles, as long as membrane M is not the outermost one of the system, thus enabling membrane division rules.

First of all, binary nondeterministic choices can be simulated by complementing the rule schema (8) with the (elementary or non-elementary) division rule schema

$$[a_i]^{(q,i)}_M \to [b_i]^{(r,i+d_1)}_M \; [c_i]^{(s,i+d_2)}_M \qquad \text{for } i \in [0, p(n)]$$

for each nondeterministic choice $\delta(q,a) = \{(r,b,d_1),(s,c,d_2)\}$ of the Turing machine. This produces two copies of membrane M, which simulate in parallel two distinct computations of the Turing machine.

When all computations of M have ended, which we may assume without loss of generality to be all of the same length, the P system will have generated as many copies of membrane M as the number of computations, each of them simultaneously sending out an instance of yes or no, according to the result of the corresponding computation. Simulating a counting machine requires us to check whether the number of yes objects is greater than the number of no objects.

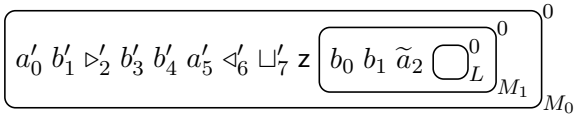
By Theorem 4.2 we have $\mathbf{PMC}^{\mathbf{PP}^L}_{\mathcal{AM}(0)} = \mathbf{P}^{\mathbf{PP}^L}$, hence every problem in $\mathbf{PMC}^{\mathbf{PP}^L}_{\mathcal{AM}(0)}$ can be solved by a family of depth-0 P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ having an external membrane M_0, where a deterministic Turing machine is simulated, containing an oracle membrane for $L_1 \in \mathbf{PP}^L$. Let M_1 be a polynomial-time counting Turing machine deciding L_1; we assume (Lemma 2.1) that M_1 always has an odd number of computations. We replace the oracle membrane L_1 of each Π_x with a membrane, also called M_1, simulating the machine M_1, which in turn contains an oracle membrane for L, a potentially less complex language than L_1. This increases the nesting depth of Π_x by one, as shown below.

We now show how to define membrane M_1 in such a way that it behaves as an oracle for membrane M_0, except for the time needed to respond to its queries (which increases from one step to a polynomial number of steps).

First of all, we change the last step of the oracle querying procedure from (12) to

$$\widetilde{\sigma}_{k,0}\ [\ ]^{+}_{M_1} \to [\widetilde{\sigma}_k]^{0}_{M_1} \qquad \text{for } k \in [1, p(n)-2]$$

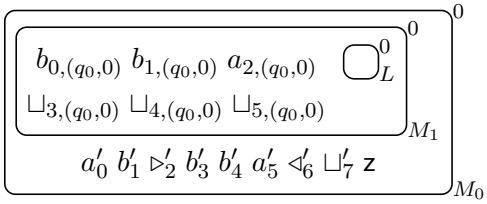
This leads to the following configuration:

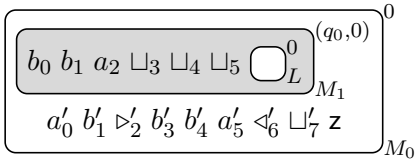


In the next computation step, the query string objects in M_1 acquire the subscript $(q_0, 0)$, where q_0 is the initial state of the Turing machine M_1; the last object $\widetilde{\sigma}_k$ also pads the input multiset to length $q(k+1)$ (the space required by M_1 on inputs of length $k+1$) with as many blank objects as needed in order to perform the simulation:

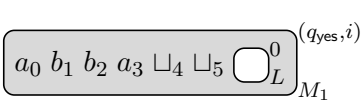
$$[\sigma_j \to \sigma_{j,(q_0,0)}]^{0}_{M_1} \qquad \text{for } j \in [0, p(n)]$$
$$[\widetilde{\sigma}_k \to \sigma_{k,(q_0,0)} \sqcup_{k+1,(q_0,0)} \cdots \sqcup_{q(k+1)-1,(q_0,0)}]^{0}_{M_1} \qquad \text{for } k \in [0, p(n)]$$

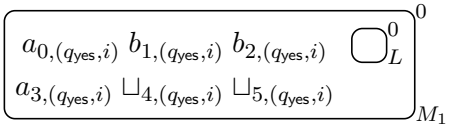
For instance, if the Turing machine M_1 needs three extra blank symbols ($q(3) = 6$), this produces the following configuration, also shown in high-level view:





Now there are no applicable rules for membrane M_0. Membrane M_1 simulates the computation of the machine M_1 (possibly including queries to the oracle for L). At the end of the simulation of this oracle query, all the copies of membrane M_1 that have been generated have, as in the proof of Lemma 4.1, a configuration of the form

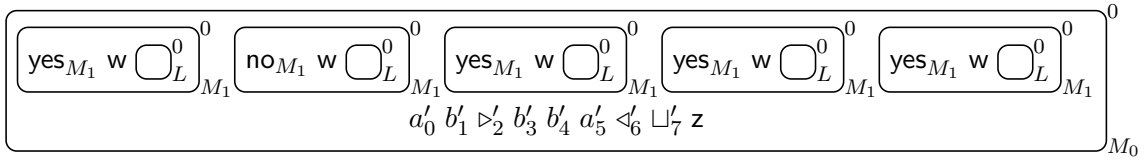




Instead of just rewriting $\sigma_{0,(q_{\mathsf{yes}},i)}$ (resp., $\sigma_{0,(q_{\mathsf{no}},i)}$) into yes_{M_1} (resp., no_{M_1}), we replace rules (9) and (10) with

$$[\sigma_{0,(q_{\mathsf{yes}},i)} \to \mathsf{yes}_M\ \mathsf{w}]^{0}_{M} \qquad \text{for } i \in [0, p(n)]$$
$$[\sigma_{0,(q_{\mathsf{no}},i)} \to \mathsf{no}_M\ \mathsf{w}]^{0}_{M} \qquad \text{for } i \in [0, p(n)]$$

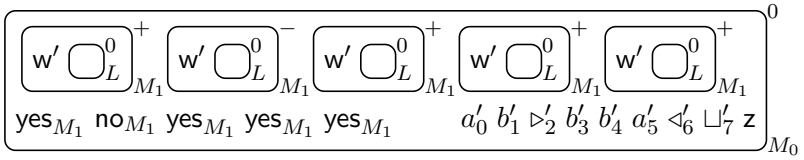
which also produce a timer object w. The configuration of the whole system, assuming the Turing machine M_1 has produced five computations (four of them accepting), is then



The rules in (11) send out the result and change the charge of the membranes M_1, while the object w is primed by means of the rule

$$[\mathsf{w} \to \mathsf{w}']^0_{M_1}$$

leading to



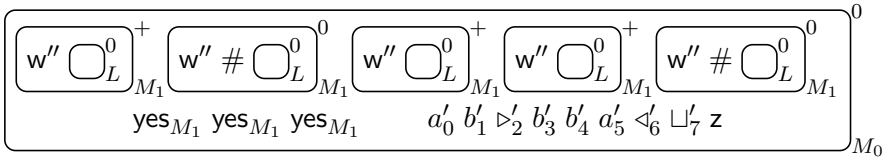
Now the P system compares the number of accepting and rejecting simulated computations of M_1 by pairing positive membranes M_1 with objects no_{M_1}, and negative membranes M_1 with objects yes_{M_1}, by using send-in rules that reset the charges to neutral:

$$\mathsf{no}_{M_1}\ [\ ]^+_{M_1} \to [\#]^0_{M_1} \qquad\qquad \mathsf{yes}_{M_1}\ [\ ]^-_{M_1} \to [\#]^0_{M_1}$$

In the mean time, object w' is primed again:

$$[\mathsf{w}' \to \mathsf{w}'']^+_{M_1} \qquad\qquad [\mathsf{w}' \to \mathsf{w}'']^-_{M_1}$$

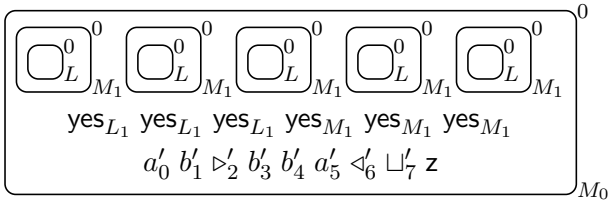
After this step, all the non-neutral membranes M_1 (at least one will exist, since the number of computations of M_1 is odd) will have the same charge, positive if accepting computations are the majority, and negative otherwise:



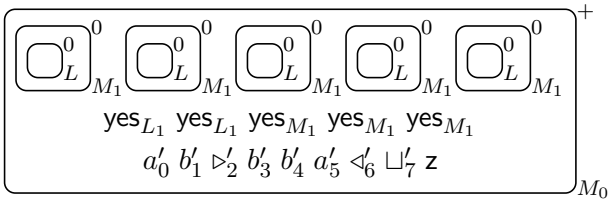
The objects w'' and $\#$ inside neutral membranes M_1 are now deleted, while the objects w'' in positive (resp., negative) membranes M_1 are sent out as yes_{L_1} (resp., no_{L_1}) while resetting the charge to neutral:

$$[\mathsf{w}'' \to \epsilon]^0_{M_1} \qquad\qquad [\# \to \epsilon]^0_{M_1}$$
$$[\mathsf{w}'']^+_{M_1} \to [\ ]^0_{M_1}\ \mathsf{yes}_{L_1} \qquad\qquad [\mathsf{w}'']^-_{M_1} \to [\ ]^0_{M_1}\ \mathsf{no}_{L_1}$$

We obtain the following configuration:



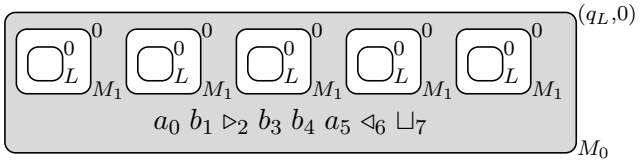
The objects yes_{L_1} (resp., no_{L_1}) are the query results needed by M_0, although possibly in multiple copies. Hence, a single copy of yes_{L_1} (resp., no_{L_1}) is sent out by rules (13). This leads to the configuration



When the charge of M_0 changes, the remaining copies of yes_{M_1} and yes_{L_1} (resp., no_{M_1} and no_{L_1}) are deleted

$$[\mathsf{yes}_{M_1} \to \epsilon]^+_{M_0} \qquad [\mathsf{yes}_{L_1} \to \epsilon]^+_{M_0} \qquad [\mathsf{no}_{M_1} \to \epsilon]^-_{M_0} \qquad [\mathsf{no}_{L_1} \to \epsilon]^-_{M_0}$$

At the same time, rules (14)–(17) are applied, and we reach the configuration



which corresponds to the configuration reached by M_0 when an actual oracle for L_1 replaces M_1.

Notice that several copies of M_1 (empty and with neutral charge) now occur in the configuration of the P system. However, when performing another query, only one of them will be selected, by the first object entering it and setting its charge to positive. □

By iterating Lemma 5.1 from $\mathbf{PMC}^{\mathbf{C}_k\mathbf{P}}_{\mathcal{AM}(0)}$ we obtain, for all $k \in \mathbb{N}$, the chain of inclusions

$$\mathbf{PMC}^{\mathbf{C}_k\mathbf{P}}_{\mathcal{AM}(0)} \subseteq \mathbf{PMC}^{\mathbf{C}_{k-1}\mathbf{P}}_{\mathcal{AM}(1)} \subseteq \cdots \subseteq \mathbf{PMC}^{\mathbf{PP}^{\mathbf{P}}}_{\mathcal{AM}(k-1)} \subseteq \mathbf{PMC}^{\mathbf{P}}_{\mathcal{AM}(k)}.$$

Furthermore, notice that a $\mathbf{P}$ oracle can always be ignored by a polynomial-time, depth-0 P system (i.e., it can be simulated without the need for queries), since $\mathbf{P} \subseteq \mathbf{PMC}_{\mathcal{AM}(0)}$, and the corresponding oracle membrane can be removed from the P system; hence, $\mathbf{PMC}^{\mathbf{P}}_{\mathcal{AM}(k)} \subseteq \mathbf{PMC}_{\mathcal{AM}(k)}$. This proves the following lemma.

Lemma 5.2. $\mathbf{PMC}^{\mathbf{C}_k\mathbf{P}}_{\mathcal{AM}(0)} \subseteq \mathbf{PMC}_{\mathcal{AM}(k)}$ for each $k \in \mathbb{N}$.

By combining Theorem 4.2 and Lemma 5.2 we obtain our main result.

Theorem 5.3. $\mathbf{P}^{\mathbf{C}_k\mathbf{P}} \subseteq \mathbf{PMC}_{\mathcal{AM}(k)}$ for each $k \in \mathbb{N}$.

By taking the union for all $k \in \mathbb{N}$ on both sides of the inclusion and observing that $\mathbf{P}^{\mathbf{C}_k\mathbf{P}}$ is always included in $\mathbf{PP}^{\mathbf{C}_k\mathbf{P}} = \mathbf{C}_{k+1}\mathbf{P}$, this implies that constant-depth P systems with active membranes working in polynomial time are at least as powerful as the counting hierarchy.

Corollary 5.4. $\mathbf{CH} \subseteq \mathbf{PMC}_{\mathcal{AM}(O(1))}$.

6. Conclusions

We have proved that P systems with active membranes of constant depth working in polynomial time solve all problems in the counting hierarchy $\mathbf{CH}$. In particular, membrane nesting depth k is at least as powerful as $\mathbf{P}^{\mathbf{C}_k\mathbf{P}}$. Is this an actual characterisation of $\mathbf{CH}$, that is, does $\mathbf{CH}$ include $\mathbf{PMC}_{\mathcal{AM}(O(1))}$? And, if so, does the reverse inclusion $\mathbf{PMC}_{\mathcal{AM}(k)} \subseteq \mathbf{P}^{\mathbf{C}_k\mathbf{P}}$, currently known to hold for the first two levels (Lemma 4.1 and [3]), also hold for all $k \geq 2$? We conjecture that this is indeed the case, even in the presence of dissolution rules, and plan to pursue this line of investigation in the near future.

Acknowledgements This work was partially supported by Fondo d'Ateneo (FA) 2013 of Università degli Studi di Milano-Bicocca: "Complessità computazionale in modelli di calcolo bioispirati: Sistemi a membrane e sistemi a reazioni".

References

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

[2] Hemaspaandra, L. A., Ogihara, M.: *The Complexity Theory Companion*, Texts in Theoretical Computer Science, Springer, 2002.

[3] Leporati, A., Manzoni, L., Mauri, G., Porreca, A. E., Zandron, C.: Simulating elementary active membranes, with an application to the P conjecture, in: *15th International Conference on Membrane Computing, Proceedings* (M. Gheorghe, P. Sosík, Š. Vavrečková, Eds.), 2014, 251–266.

[4] Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions, *Natural Computing*, **10**(1), 2011, 613–632.

[5] Păun, Gh.: P systems with active membranes: Attacking NP-complete problems, *Journal of Automata, Languages and Combinatorics*, **6**(1), 2001, 75–90.

[6] Pérez-Jiménez, M. J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes, *Natural Computing*, **2**(3), 2003, 265–284.

[7] Porreca, A. E., Leporati, A., Mauri, G., Zandron, C.: P systems simulating oracle computations, in: *Membrane Computing, 12th International Conference, CMC 2011* (M. Gheorghe, Gh. Păun, A. Salomaa, G. Rozenberg, S. Verlan, Eds.), vol. 7184 of *Lecture Notes in Computer Science*, Springer, 2012, 346–358.

[8] Porreca, A. E., Murphy, N.: First steps towards linking membrane depth and the Polynomial Hierarchy, in: *Eight Brainstorming Week on Membrane Computing* (M. A. Martínez-del-Amor, Gh. Păun, I. Pérez-Hurtado, A. Riscos-Núñez, Eds.), number 1/2010 in RGNC Reports, Fénix Editora, 2010, 255–266.

[9] Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE, *Journal of Computer and System Sciences*, **73**(1), 2007, 137–152.

[10] Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes, in: *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference* (I. Antoniou, C. S. Calude, M. J. Dinneen, Eds.), Springer, 2001, 289–301.

[11] Zandron, C., Leporati, A., Ferretti, C., Mauri, G., Pérez-Jiménez, M. J.: On the computational efficiency of polarizationless recognizer P systems with strong division and dissolution, *Fundamenta Informaticae*, **87**, 2008, 79–91.