

Non-confluence in divisionless P systems with active membranes

Antonio E. Porreca^{*,a}, Giancarlo Mauri^a, Claudio Zandron^a

^a*Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
viale Sarca 336, 20126 Milano, Italy*

Abstract

We describe a solution to the SAT problem via non-confluent P systems with active membranes, without using membrane division rules. Furthermore, we provide an algorithm for simulating such devices on a nondeterministic Turing machine with a polynomial slowdown. Together, these results prove that the complexity class of problems solvable non-confluently and in polynomial time by this kind of P systems is exactly the class **NP**.

Key words: membrane computing, complexity theory

1. Introduction

Membrane systems (also called *P systems*) have been introduced in [6] as a new class of distributed and parallel computing devices, inspired by the structure and functioning of living cells. The basic model consists of a hierarchical structure composed of several membranes, embedded into a main membrane called the *skin*. Membranes divide the space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *developmental rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. Moreover, the membrane structure can be modified, by dissolving membranes. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be applied. Usually, the result of a computation is the multiset of objects contained in an *output membrane* or emitted from the skin of the system.

In [7] a variant of such systems was proposed, where the membrane structure can be modified also by adding new membranes, which can be created by dividing existing membranes, through a processes inspired from cellular *mitosis*. This variant allows the production of an exponential number of membranes in polynomial time. For this reason, in the cited paper it was shown that such

*Corresponding author

Email addresses: porreca@disco.unimib.it (Antonio E. Porreca),
mauri@disco.unimib.it (Giancarlo Mauri), zandron@disco.unimib.it (Claudio Zandron)

systems are able to solve **NP**-complete problems in a time-efficient way (by trading space for time).

Starting from this result, various complexity classes for P systems were defined [9]. Such classes were then compared with usual complexity classes such as **P** and **NP** (see, e.g., [10] and [2]); in [13, 1] it was shown that complexity classes defined in terms of two variants of P systems with active membranes contain all problems in **PSPACE**. In [12] it was shown that there exists a complexity class for P systems with active membranes which includes the class **PSPACE** and which is included in the class **EXP**; in [14] this result was strengthened by proving equality to **PSPACE**.

The previous results were given considering either deterministic systems or *confluent* systems, i.e. systems which can operate in a non deterministic way, but in such a way that the result (acceptance or rejection) of every computation is the same (thus, the simulation of a single computation is enough to determine the result of all computations).

In [16] it was shown that the class of problems solved by confluent P systems in polynomial time without using membrane division, usually denoted by $\mathbf{PMC}_{\mathcal{NAM}}$, is equal to the standard complexity class **P**. In this paper, we continue in this direction, considering non-confluent P systems without membrane division and operating in polynomial time. We first propose a solution for the decision problem SAT (satisfiability for Boolean formulas) with a polynomially semi-uniform family of non-confluent P systems without membrane division, thus proving that $\mathbf{NP} \subseteq \mathbf{NPMC}_{\mathcal{NAM}}$. Then we prove the reverse inclusion, providing a non-deterministic variant of the theorem proved in [16], showing that non-confluent P systems without membrane division can be simulated in polynomial time by non-deterministic Turing machines. As a consequence, we have that $\mathbf{NPMC}_{\mathcal{NAM}}$ equals the complexity class **NP**.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For a systematic introduction, we refer the reader to [8]. A survey and an up-to-date bibliography concerning P systems can be found at the web address <http://ppage.psystems.eu>.

The rest of the paper is organized as follows. In section 2 we give basic definitions for membrane systems which will be used throughout the rest of the paper. In section 3 we show that all problems in **NP** can be solved in polynomial time by non-confluent P systems without membrane division. In section 4 we prove that non-confluent P systems without membrane division can be simulated by non-deterministic Turing machines in polynomial time, thus proving the converse inclusion. Section 5 concludes the paper and presents open problems and directions for future research.

2. Definitions

We begin by recalling the formal definition of P system with active membranes and the usual process by which they are used to solve decision problems.

Definition 1. A *P system with active membranes* is a structure

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

where

- Γ is a finite alphabet of symbols or objects;
- Λ is a finite set of labels;
- μ is a membrane structure (i.e. a rooted, unordered tree) of m membranes, labeled with elements of Λ ; different membranes may be given the same label;
- w_1, \dots, w_m are multisets over Γ describing the initial content of the m membranes in μ ;
- R is a finite set of developmental rules.

The *polarization* of a membrane is one of $+$ (positive), $-$ (negative) or 0 (neutral); each membrane is assumed to be initially neutral.

Developmental rules are of the following six kinds:

- *Object evolution rule* of the form $[a \rightarrow w]_h^\alpha$
It can be applied to a membrane labeled by h , having polarization α and containing an occurrence of the object a ; the object a is rewritten to the multiset w (i.e. a is removed from the multiset in h and replaced by every object in w).
- *Communication rule* of the form $a []_h^\alpha \rightarrow [b]_h^\beta$
It can be applied to a membrane labeled by h , having polarization α and such that the external region contains an occurrence of the object a ; the object a is sent into h becoming b and, simultaneously, the polarization of h is changed to β .
- *Communication rule* of the form $[a]_h^\alpha \rightarrow []_h^\beta b$
It can be applied to a membrane labeled by h , having polarization α and containing an occurrence of the object a ; the object a is sent out from membrane h to the outside region becoming b and, simultaneously, the polarization of h is changed to β .
- *Dissolution rule* of the form $[a]_h^\alpha \rightarrow b$
It can be applied to a membrane labeled by h , having polarization α and containing an occurrence of the object a ; the membrane h is dissolved and its content is left in the surrounding region unaltered, except that an occurrence of a becomes b .

The two following kinds of rule are only described here for completeness, but they are not used in the rest of this paper:

- *Elementary division rule* of the form $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$

It can be applied to an elementary membrane labeled by h , having polarization α and containing an occurrence of the object a ; the membrane is divided into two membranes having label h and polarizations β and γ ; the object a is replaced, respectively, by b and c while the other objects in the initial multiset are copied to both membranes.

- *Non-elementary division rule* of the form

$$[[]_{h_1}^+ \cdots []_{h_k}^+ []_{h_{k+1}}^- \cdots []_{h_n}^-]_h^\alpha \rightarrow [[]_{h_1}^\delta \cdots []_{h_k}^\delta]_h^\beta [[]_{h_{k+1}}^\epsilon \cdots []_{h_n}^\epsilon]_h^\gamma$$

It can be applied to a non-elementary membrane labeled by h , having polarization α , containing the positively charged membranes h_1, \dots, h_k and the negatively charged membranes h_{k+1}, \dots, h_n ; no other non-neutral membrane may be contained in h . The membrane h is divided into two copies with polarization β and γ ; the positive children are placed inside the former, their polarization changed to δ , while the negative ones are placed inside the latter, their polarization changed to ϵ . Any neutral membrane inside h is duplicated and placed inside both copies.

A *configuration* of a P system with active membranes Π is given by a membrane structure and the multisets contained in its regions. In particular, the *initial configuration* is given by the membrane structure μ and the initial contents of its membranes w_1, \dots, w_m . A computation step leads from a configuration to the next one according to the following principles:

- the developmental rules are applied in a *maximally parallel* way: when one or more rules can be applied to an object and/or membrane, then one of them *must* be applied. The only elements left untouched are those which can not be subject to any rule;
- each object can be subject to only one rule during that step. Also membranes can be subject to only one rule, except that *any* number of object evolution rules can be applied inside them;
- when more than one rule can be applied to an object or membrane, then the one actually applied is chosen nondeterministically. Thus multiple, distinct configurations may be reachable by means of a computation step from a single configuration;
- when a dissolution or division rule is applied to a membrane, the multiset of objects to be released outside or copied is the one *after* any application of object evolution rules inside such membrane;
- the skin membrane can not be divided or dissolved, nor any object can be sent in from the environment surrounding it (i.e. an object which leaves the skin membrane can not be brought in again).

A sequence of configurations, each one reachable from the previous one by means of developmental rules, is called a *computation*. Due to nondeterminism, there may be multiple computations starting from the initial configuration, thus giving birth to a computation tree. A computation halts when no further configuration can be reached, i.e. when no rule can be applied in a given configuration.

Families of *recognizer* P systems can be used to decide languages (or, equivalently, to solve decision problems) as follows.

Definition 2. Let Π be a P system whose alphabet contains two distinct objects *yes* and *no*, such that every computation of Π is halting and during each computation exactly one of the objects *yes*, *no* is sent out from the skin to signal acceptance or rejection. If all the computations of Π agree on the result, then Π is said to be *confluent*; if this is not necessarily the case, then it is said to be *non-confluent* and the global result is acceptance iff there exists an accepting computation.

Definition 3. Let $L \subseteq \Sigma^*$ be a language, \mathcal{D} a class of P systems and let $\Pi = \{\Pi_x \mid x \in \Sigma^*\} \subseteq \mathcal{D}$ be a family of P systems, either confluent or non-confluent. We say that Π *decides* L when, for each $x \in \Sigma^*$, $x \in L$ iff Π_x accepts.

Complexity classes for P systems are defined by imposing a uniformity condition on Π and restricting the amount of time available for deciding a language.

Definition 4. A language $L \subseteq \Sigma^*$ belongs to the complexity class $\mathbf{PMC}_{\mathcal{D}}$ iff there exists a family of confluent P systems $\Pi = \{\Pi_x \mid x \in \Sigma^*\} \subseteq \mathcal{D}$ deciding L such that

- Π is *polynomially semi-uniform*, i.e. there exists a deterministic Turing machine which, for each input $x \in \Sigma^*$, constructs the P system Π_x in polynomial time;
- Π itself operates in polynomial time, i.e. there exists a polynomial p such that, for each $x \in \Sigma^*$, every computation of Π_x halts within $p(|x|)$ steps.

The analogous complexity class for *non-confluent* P systems is denoted by $\mathbf{NPMC}_{\mathcal{D}}$.

From now on, the only class \mathcal{D} of P systems we are going to consider is \mathcal{NAM} , the class of P systems with active membranes without any membrane division rules; furthermore, we restrict our attention to non-confluent P systems.

3. Solving SAT via non-confluence

In this section we show how to solve the SAT problem by using a semi-uniform family of non-confluent P systems with active membranes, without any form of membrane division rule. A non-confluent solution to this problem, using transition P systems with cooperation and priorities between rules (which

are both very powerful features), can be found in [9]; on the other hand, we only use context-free rules and no priorities, as in the standard definition of P systems with active membranes. The idea of our solution is representing the parse tree of a formula as a membrane structure, then evaluating the formula, by using communication rules, according to a nondeterministically generated truth assignment (as we do when solving SAT via a nondeterministic Turing machine). We begin by describing the evaluation phase, then we turn to the generation of truth assignments.

3.1. Evaluating Boolean formulas without variables

The set Φ_0 of Boolean formulas without variables is recursively constructed from the Boolean constants 0 and 1 and the usual connectives (conjunction, disjunction and negation) according to the following context-free grammar:

$$\Phi_0 \rightarrow 0 \mid 1 \mid (\Phi_0 \wedge \Phi_0) \mid (\Phi_0 \vee \Phi_0) \mid \neg\Phi_0.$$

Such formulas can be evaluated recursively by using the truth table associated with each connective.

An alternative characterization of Boolean formulas is given by their parse trees, and this is where P systems come into play. Indeed, membrane structures are nothing but trees: this suggests a representation where Boolean constants correspond to elementary membranes, and a complex formula ϕ is built by enclosing one or two membrane structures (representing the subformulas of ϕ) by a membrane corresponding to the principal connective. The only potential complication is given by the fact that parse trees are *ordered* trees, while membrane structures are not (i.e. there is no distinguished “first” child membrane, “second” child, etc.). In the context of the evaluation of formulas, however, this poses no real problem, as both binary connectives are commutative.

In order to give a formal transformation from a Boolean formula $\phi \in \Phi_0$ to a membrane structure $\mu(\phi)$, we also need to provide a labeling scheme for the membranes. We begin by indexing each constant and connective in ϕ with an increasing positive integer, starting from the left. For instance, the Boolean formula $1 \wedge (0 \vee \neg 1)$ is indexed as $1_1 \wedge_2 (0_3 \vee_4 \neg_5 1_6)$. The membrane structure corresponding to ϕ can then be recursively defined by

$$\mu(\phi) = \begin{cases} []_i & \text{if } \phi = 0_i \text{ or } \phi = 1_i \\ [\mu(\psi) \mu(\chi)]_i & \text{if } \phi = (\psi \wedge_i \chi) \text{ or } \phi = (\psi \vee_i \chi) \\ [\mu(\psi)]_i & \text{if } \phi = \neg_i \psi \end{cases} \quad (1)$$

that is, the indices are used as labels for the membranes. We give different labels to membranes corresponding to different occurrences of the same connective or constant; this is not strictly necessary with this kind of formulas, as each occurrence behaves in the same way with respect to evaluation; however, we will need such unique labeling scheme when dealing with variables.

The next step is the evaluation proper of our formula. This is performed by using communication rules only; thus, at most one rule per step can be applied

for each membrane. The only possible values for a Boolean formula are 0 and 1; we represent these values as the two symbol-objects 0 and 1 in the alphabet of the P system. The evaluation of a subformula ϕ is performed by sending out from the membrane corresponding to the principal connective of ϕ either 0 or 1, according to the value of ϕ . We assume that, in the initial configuration, each elementary membrane contains exactly one copy of the object corresponding to the constant it represents; that is, if 0_i (resp. 1_i) occurs in ϕ , then membrane i in $\mu(\phi)$ contains exactly the object 0 (resp. 1). As for the base case, when ϕ is simply 0_i , its evaluation is then performed simply by sending out the object 0 according to the communication rule

$$[0]_i^0 \rightarrow []_i^- 0. \quad (2)$$

We set the polarization of membrane i to negative in order to signal that no further result is to be produced from the membrane. When ϕ is 1_i we proceed analogously:

$$[1]_i^0 \rightarrow []_i^- 1. \quad (3)$$

Thus the evaluation of formulas consisting only of a Boolean constant is performed correctly, and it requires a single computation step.

Let us now consider complex formulas. When ϕ is $(\psi \wedge_i \chi)$ and we assume that the evaluation of ψ and χ is correct and performed in, respectively, m and n steps, then we know that *two* value-objects will be sent to membrane i . In order to compute the conjunction of these values, we begin by observing the first object to arrive¹: if it is a 0 we can already output the final result 0 by using the following rule:

$$[0]_i^0 \rightarrow []_i^- 0 \quad (4)$$

as conjunction is false when one of the two conjuncts is. However, if the first value is 1, we need to remember that this value has arrived and wait for the other one. We do so by changing the polarization of membrane i to positive, while simultaneously getting rid of the 1 by sending it out transformed to #: this is a “junk” object which will not have any further role during the computation (i.e. it does not appear on the left-hand side of any rule). The corresponding rule is

$$[1]_i^0 \rightarrow []_i^+ \#. \quad (5)$$

When the second value-object is observed, we can just send it out from the membrane, as $1 \wedge x$ evaluates to x ; the following two rules handle this case:

$$[0]_i^+ \rightarrow []_i^- 0 \quad (6)$$

$$[1]_i^+ \rightarrow []_i^- 1 \quad (7)$$

Thus we can evaluate conjunction correctly in at most $\max\{m, n\} + 2$ steps.

¹The two objects may also arrive simultaneously, when $m = n$; in that case the first object to be observed is then chosen nondeterministically as usual.

The case $\phi = (\psi \vee_i \chi)$ is dual; the corresponding rules are chosen to exploit the fact that $0 \vee x$ evaluates to x and $1 \vee x$ evaluates to 1:

$$[1]_i^0 \rightarrow []_i^- 1 \quad (8)$$

$$[0]_i^0 \rightarrow []_i^+ \# \quad (9)$$

$$[0]_i^+ \rightarrow []_i^- 0 \quad (10)$$

$$[1]_i^+ \rightarrow []_i^- 1. \quad (11)$$

Evaluating disjunction also requires at most $\max\{m, n\} + 2$ steps.

The final case is $\phi = \neg_i \psi$; evaluating negation simply means exchanging zeros and ones:

$$[0]_i^0 \rightarrow []_i^- 1 \quad (12)$$

$$[1]_i^0 \rightarrow []_i^- 0. \quad (13)$$

This requires $n + 1$ steps, where n is the time required to evaluate ψ .

Lemma 1. *The Boolean formula problem (that is, deciding whether $\phi \in \Phi_0$ evaluates to 1) can be solved in linear time via a polynomially semi-uniform family $\mathbf{\Pi}$ of confluent P systems with active membranes by using only communication rules.*

Proof. Let $\phi \in \Phi_0$ be a formula of length n . The P system $\Pi_\phi \in \mathbf{\Pi}$ associated with it has the following features:

- the alphabet is $\Gamma = \{0, 1, \#, yes, no\}$;
- the set of labels is $\Lambda = \{0, \dots, n\}$;
- the membrane structure is $\mu = [\mu(\phi)]_0$, that is, the membrane structure defined in (1), further enclosed by a skin membrane having label 0;
- each elementary membrane contains exactly one copy of the value-object corresponding to the Boolean constant it represents; every other membrane is initially empty;
- the set of rules R contains, for each i :
 - a copy of rule (2) if ϕ contains 0_i ;
 - a copy of rule (3) if ϕ contains 1_i ;
 - a copy of rules (4) to (7) if ϕ contains \wedge_i ;
 - a copy of rules (8) to (11) if ϕ contains \vee_i ;
 - a copy of rules (12) and (13) if ϕ contains \neg_i ;
 - a pair of rules to convert the result of the evaluation into a result of acceptance or rejection:

$$[1]_0^0 \rightarrow []_0^- yes \quad [0]_0^0 \rightarrow []_0^- no.$$

Then Π_ϕ can be constructed in polynomial time by a Turing machine: the membrane structure is isomorphic to the parse tree of ϕ (when viewed as an unordered tree), which can be computed in polynomial time by using any parsing algorithm such as CYK [3]; then, for each membrane $O(1)$ time is needed to output its initial contents and the set of rules associated to it. Thus Π is polynomially semi-uniform.

Furthermore, by induction on n we see that the result of the evaluation of ϕ is sent to the skin membrane (and then converted to *yes* or *no*) in $O(n)$ steps; thus Π_ϕ operates in linear time with respect to the size of ϕ . \square

3.2. Finding a satisfying truth assignment

We now turn our attention to the set Φ of Boolean formulas with variables. Such set is defined recursively by

$$\Phi \rightarrow V \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi) \mid \neg\Phi$$

where V is a countably infinite set of variables. The problem is deciding whether $\phi \in \Phi$ has a satisfying truth assignment.

Solving this problem by using non-confluent P systems is similar to the usual nondeterministic solution: first guess a truth assignment for the variables occurring in ϕ , then evaluate ϕ according to such an assignment and accept if and only if the result is 1, rejecting otherwise. Since nondeterminism ensures that only a satisfying assignment is guessed (if it exists at all), the correctness of this approach is easily established.

Noticing that we are already able to perform evaluation once each variable has been replaced by its truth value, as we showed in the previous section, and that the non-confluent behaviour of P systems is the same as that of nondeterministic Turing machines, we choose to implement the same method. First we take into account the different structure of formulas (which are indexed by integers as above) in the definition of the membrane structure:

$$\mu(\phi) = \begin{cases} []_i & \text{if } \phi = x_i \text{ for some } x \in V \\ [\mu(\psi) \mu(\chi)]_i & \text{if } \phi = (\psi \wedge_i \chi) \text{ or } \phi = (\psi \vee_i \chi) \\ [\mu(\psi)]_i & \text{if } \phi = \neg_i \psi \end{cases} \quad (14)$$

thus elementary membranes now correspond to variables.

In order to evaluate the formula consistently, we need to ensure that each occurrence of the same variable evaluates to the same Boolean value; thus, it is not possible to generate the truth assignment directly inside the elementary membranes. A solution is generating the object-values corresponding to each variable (with the necessary multiplicity) in a single place, e.g. inside the skin membrane, before moving them to the elementary membranes, again by using communication rules, and then proceeding with the evaluation as described in the previous section.

Let $\phi \in \Phi$ and let x_1, \dots, x_k be all the variables occurring in ϕ ; finally, let $\mu = [\mu(\phi)]_0$ be our membrane structure. Initially, we put one copy of each

object x_1, \dots, x_k inside the skin membrane (labeled by 0) and, for $i = 1, \dots, k$, we define a pair of conflicting object evolution rules

$$[x_i \rightarrow f_i]_0^0 \quad (15)$$

$$[x_i \rightarrow t_i]_0^0 \quad (16)$$

representing a nondeterministic choice for the value of x_i , where f_i indicates a false value for x_i and t_i a true value. Then, each of these temporary objects generates a number of zeroes and ones according to the number of occurrences of x_i in ϕ by using object evolution rules; furthermore, each of these value-objects is subscripted by using the label of a membrane representing an occurrence of x_i . Thus, if j_1, \dots, j_ℓ are the indices of the occurrences of x_i in ϕ , the corresponding evolution rules are

$$[f_i \rightarrow 0_{j_1} \cdots 0_{j_\ell}]_0^0 \quad (17)$$

$$[t_i \rightarrow 1_{j_1} \cdots 1_{j_\ell}]_0^0. \quad (18)$$

After two computation steps the skin membrane contains the required number of value-objects, one for each occurrence of a variable, each one subscripted by the label of the elementary membrane it belongs to. By using communication rules, we move these objects along the unique path from the skin membrane to its target membrane. If j is the label of a membrane on the path from the skin to membrane i , including the skin but *excluding* membrane i itself, then we define the following communication rules

$$0_i []_j^\alpha \rightarrow [0_i]_j^\alpha \quad \forall \alpha \in \{+, 0, -\} \quad (19)$$

$$1_i []_j^\alpha \rightarrow [1_i]_j^\alpha \quad \forall \alpha \in \{+, 0, -\} \quad (20)$$

to move the appropriate value-object one step closer to the target (i.e. one level down towards the leaves, when looking at the membrane structure as a tree). As for the target membrane i itself, it absorbs the value-object while removing its subscript:

$$0_i []_i^0 \rightarrow [0]_i^0 \quad (21)$$

$$1_i []_i^0 \rightarrow [1]_i^0. \quad (22)$$

Now the evaluation may proceed as described in section 3.1. Observe how the new communication rules, those bringing the evaluated occurrences of variables to the elementary membranes, do not interfere with the old rules from the point of view of correctness, since values moving towards their target membrane are subscripted instead of “plain” and such movements do not alter any polarization.

This completes the description of our algorithm for solving the satisfiability problem; we now need to show that it can be executed in polynomial time.

Lemma 2. *The satisfiability problem for Boolean formulas can be solved in polynomial time via a polynomially semi-uniform family Π of non-confluent P systems with active membranes by using only object evolution and communication rules.*

Proof. Given $\phi \in \Phi$ of length n , let x_1, \dots, x_k be the variables occurring in ϕ ; we construct the P system $\Pi_\phi \in \mathbf{\Pi}$ having the following features:

- the alphabet is $\Gamma = \{x_i, f_i, t_i, 0_i, 1_i \mid 1 \leq i \leq n\} \cup \{0, 1, \#, yes, no\}$;
- the set of labels is $\Lambda = \{0, \dots, n\}$;
- the membrane structure is $\mu = [\mu(\phi)]_0$;
- the initial skin multiset is $x_1 \cdots x_k$, while every other membrane is initially empty;
- the set of rules R is given by the same set of rules as in the proof of lemma 1, augmented by a copy of the following rules for each variable x_i :
 - a copy of rules (15) to (18);
 - a copy of rules (19) and (20) for each membrane on the path from the skin to membrane i excluded;
 - a copy of rules (21) and (22).

The P system Π_ϕ can be constructed in $O(n^2)$ time given the parse tree of ϕ , as rules (15), (18), (21) and (22) are of constant length while rules (17) to (20) are of length $O(n)$ and can be constructed by e.g. performing a depth-first search of the parse tree of ϕ , requiring linear time.

As for the computation time of Π_ϕ , we have already seen that generating all the copies of the object-values we need requires just two computation steps. Each of these value-objects then needs to move to its target membrane. This requires moving through a number of membranes equal to the depth of the target membrane, which is at most n . Since there are $O(n)$ value-objects, one for each occurrence of a variable, and assuming they move only one at a time (this is not necessarily true, since often there exist non-overlapping sub-paths in the membrane structure; however, this provides an upper bound) after $O(n^2)$ steps each value will have reached its target membrane. After that, further $O(n)$ steps are required to complete the evaluation as in lemma 1 (again, due to parallelism, evaluation may have already begun in some regions of the membrane structure, even if some values are still moving to their destination). Hence, the total number of steps is $O(n^2)$.

The nondeterministic choices between rules (15) and (16) made during the first computation step give rise to every possible truth assignment to the variables; thus, if any satisfying assignment exists, it leads to an accepting computation; when this is not the case, every computation is rejecting. Hence, according to the acceptance policy for non-confluent P systems, the formula ϕ is accepted if and only if satisfiable. \square

Since the satisfiability problem is **NP**-complete and $\mathbf{NPMC}_{\mathcal{N}AM}$ is closed under reductions, every problem in **NP** can be solved in non-confluent polynomial time without using membrane division:

Theorem 1. $\mathbf{NP} \subseteq \mathbf{NPMC}_{\mathcal{N}AM}$. \square

4. Simulating non-confluent P Systems

A well-known result in membrane computing, called the Milano theorem [16], states that a confluent P system with active membranes without division rules can be simulated by a deterministic Turing machine with a polynomial slowdown. In this section we prove a nondeterministic version of this theorem by providing a simulation algorithm; the algorithm is assumed to run on a nondeterministic random access machine with constant time addition and subtraction, but linear time multiplication; such device can be simulated efficiently by a nondeterministic Turing machine [5].

We begin by describing the representation of the P system to be given as input. The membrane structure is represented as a rooted tree, each node having a list of children and fields representing its label, polarization and multiset of objects. We also define an environment multiset to collect the output of the skin membrane. Multisets over the alphabet Γ are represented by vectors of $|\Gamma|$ binary integers (the i -th component measuring the multiplicity of the i -th symbol according to an arbitrary total ordering of the alphabet); this representation is particularly useful from an algorithmic point of view, as multiset operations are reduced to arithmetic operations. Finally, each developmental rule is described as a record holding each component involved in its application:

- records representing object evolution rules require fields representing the label and polarization of the membrane, the object on the left-hand side and the multiset on the right-hand side of the arrow;
- both kinds of evolution rules require the label of the membrane as well as the objects and polarizations on the left and right-hand sides;
- dissolution rules require the label and polarization of the membrane, as well as the objects on the left and right-hand side.

A straightforward simulation algorithm is given by the usual “semantics” of P systems: for each object or membrane, nondeterministically chosen, we pick a rule applicable to said items (again in a nondeterministic way) and change the configuration accordingly, taking care of applying only one communication or dissolution rule to each membrane, of leaving object introduced by the right-hand side of rules untouched until the next step, and of applying a dissolution rule to a membrane only after all possible objects contained inside it have evolved. Unfortunately, such an approach yields an inefficient algorithm, even when using a nondeterministic random access machine, as the number of objects can increase exponentially during the computation, thus requiring exponential time (with respect to the length of the description of the P system) in the worst-case scenario just to simulate object-evolution rules.

4.1. A polynomial-time simulation algorithm

Our solution is based on the diametrically opposed approach: instead of choosing a rule for each object or membrane, we iterate in nondeterministic order

over the developmental rules, discarding them after application (or whenever they are not applicable); when an object evolution rule $[a \rightarrow w]_h^\alpha$ is chosen, we decide (again in a nondeterministic way) how many occurrences of a inside the membrane labeled by h are to be rewritten into w . As multiple evolution rules having a on the left-hand side may occur, this number is not necessarily the maximum (or one less than the maximum, assuming an occurrence of a is used up by a communication or dissolution rule). The only drawback of our approach is the need to ensure that *all* objects which can evolve actually do so during each step; but, as we are using a nondeterministic machine to carry out the simulation, we can simply check that this requirement is met just before the end of the simulated step, aborting the computation by rejecting when this is not the case.

The maximum number of occurrences of objects inside the membrane structure is given by the following lemma.

Lemma 3. *Let Π be a non-confluent P system with active membranes without division rules having a description of length n . Then each possible configuration after t steps contains at most $2^{O(t \log n)}$ objects.*

Proof. Since Π has a description of length n , in particular

- there are at most n objects in the initial configuration;
- each object evolution rule has length at most n , thus any object can be rewritten into a multiset of size at most n .

The only rules increasing the number of objects are object evolution rules: in the worst case, each object is rewritten into at most n objects. This gives an upper bound of $n^{t+1} = 2^{O(t \log n)}$ objects after t steps. \square

This is, of course, also a bound on the multiplicity of an object in a single membrane; representing such integer in binary requires $O(t \log n)$ bits, which can be guessed by a nondeterministic machine in order to decide how many occurrences of an object will be rewritten by the evolution rule.

In order to keep track of which objects and membranes have already been used during the current simulation step, we first make a temporary copy \mathcal{C}' of the current configuration \mathcal{C} . We refer to \mathcal{C}' in order to find out whether a rule is applicable; such temporary configuration is updated by removing the objects on the left-hand side of rules without adding those on the right-hand side, since they can not be subject to further rules until the next simulated step; furthermore, each membrane subject to communication or dissolution rules is marked as used (we do so by removing all communication and dissolution rules involving such membrane from the set of eligible rules for the current simulated step). Finally, dissolution is not actually performed in \mathcal{C}' , as further evolution rules may be applied to any untouched object left inside the dissolving membrane; we just mark the membrane as used and remove the object on the left-hand side. The developmental rules are instead applied as by the definition, polarization changes included, in the actual configuration \mathcal{C} , again with the exception of dissolution

rules: the membranes to be dissolved are marked as such and they are concretely removed only after the set of rules to apply has been exhausted.

After having checked that maximal parallelism has been simulated correctly and having removed any dissolved membrane in \mathcal{C} , thus ensuring that the new value of \mathcal{C} is indeed a valid configuration reachable from the previous one, the temporary configuration \mathcal{C}' can be discarded and we may proceed with the simulation of the following step, until the result of the computation is found: this last condition can be checked by inspecting the environment multiset.

Algorithm 1. Let Π be a non-confluent P system with active membranes without division rules; let \mathcal{C} be the initial configuration, augmented with the environment multiset, R the set of rules and Γ the alphabet of Π .

- A1** If either the object *yes* or *no* is found inside the environment multiset, halt by accepting or rejecting accordingly.
- A2** Remove from R any rule involving membranes not existing in \mathcal{C} . Set $\mathcal{C}' \leftarrow \mathcal{C}$ and $R' \leftarrow R$.
- A3** Until R' is empty, nondeterministically choose $r \in R'$.
 - A4** If $r = [a \rightarrow w]_h^\alpha$ and membrane h has polarization α , nondeterministically choose a nonnegative integer $k \leq m$, where m is the multiplicity of a inside h in \mathcal{C}' . This is accomplished by guessing $O(\log m)$ bits (see lemma 3 for an upper bound). Remove k occurrences of a from h in both \mathcal{C} and \mathcal{C}' ; then add k times w to h in \mathcal{C} . Remove r from R' .
 - A5** If $r = [a]_h^\alpha \rightarrow []_h^\beta b$, membrane h has polarization α and contains an occurrence of a in \mathcal{C}' , then remove such occurrence both from \mathcal{C} and \mathcal{C}' , add an occurrence of b to the parent membrane of h in \mathcal{C} and change the polarization of h to β in \mathcal{C} . Remove r from R' ; furthermore, if r was applied, remove all communication and dissolution rules involving h from R' .
 - A6** If $r = a []_h^\alpha \rightarrow [b]_h^\beta$, membrane h has polarization α and its parent membrane contains an occurrence of a in \mathcal{C}' , then remove such occurrence both from \mathcal{C} and \mathcal{C}' , add an occurrence of b to h in \mathcal{C} and change the polarization of h to β in \mathcal{C} . Remove r from R' ; furthermore, if r was applied, remove all communication and dissolution rules involving h from R' .
 - A7** If $r = [a]_h^\alpha \rightarrow b$, membrane h has polarization α and contains an occurrence of a in \mathcal{C}' , then remove such occurrence both from \mathcal{C} and \mathcal{C}' , add an occurrence of b to the parent membrane of h in \mathcal{C} and mark h for deletion in \mathcal{C} . Remove r from R' ; furthermore, if r was applied, remove all communication and dissolution rules involving h from R' .
- A8** For each membrane h in \mathcal{C} and for each symbol $a \in \Gamma$, if h contains one or more occurrences of a in \mathcal{C}' , there exists an object evolution rule

$[a \rightarrow w]_h^\alpha \in R$ and h has polarization α in \mathcal{C}' , then abort the computation by rejecting.

A9 For each membrane h marked for deletion in \mathcal{C} , in depth-first order, add the multiset of h to the multiset in its parent membrane in \mathcal{C} , then remove h from the list of children of its parent membrane while adding to it all the children of h .

A10 Return to step **A1**.

The algorithm itself is fairly straightforward; we wish to highlight that the rules involving a dissolved membrane are permanently deleted in step **A2** only in order to avoid the need for checking whether such membrane still exists in steps **A4–A7**.

As for the correctness of the algorithm, we just need to check that the rules are indeed applied in a maximally parallel way. Suppose otherwise; then a rule which could be applied was missed, and either of the following must be the case:

- a membrane h remains which could either send out or bring in an object, or dissolve;
- an object remains which could be subject to evolution.

The first case can never happen: when an applicable communication or dissolution rule exists, sooner or later it is chosen nondeterministically; then, according to the algorithm, it must be actually applied to h in step **A5**, **A6** or **A7**. Thus, the second case is the only possible violation of the maximal parallelism requirement (an insufficient number of objects has been subject to evolution while applying evolution rules in step **A4**); however, this violation is detected in step **A8** whenever it occurs and the whole computation is discarded.

4.2. Analysis of the algorithm

We can now prove that our algorithm is able to simulate a non-confluent P system without division with a polynomial slowdown.

Theorem 2. *Let Π be a non-confluent P system with active membranes without division rules, having a description of length n and running in t steps. Then Π can be simulated by algorithm 1 in $O(n^2t^2 \log n + n^3t)$ time.*

Proof. Suppose we have already simulated the k -th computation step of Π ; we now analyze the cost of each step of algorithm 1 during the $(k+1)$ -th simulated step. Given the length of the description of Π , we know that its membrane structure, its alphabet, its set of rules and the rules themselves have size $O(n)$.

A1 Checking whether there are more than zero occurrences of an object in a multiset requires $O(1)$ time on a random access machine, assuming the vector representation.

A2 Removing the “useless” rules requires checking whether each membrane mentioned in the set of rules actually exists, by traversing the membrane structure, in $O(n^2)$ time. The size of the whole configuration \mathcal{C} is given by the product of the number of membranes, the size of the alphabet and the size of the integer representing the multiplicity of each object. By lemma 3, the total size is thus

$$O(n) \times O(n) \times O(k \log n) = O(n^2 k \log n)$$

which is also the time required to make a copy of \mathcal{C} , and the total for step **A2**.

A3 Nondeterministically choosing a rule in R' requires linear time with respect to the size of this set, thus $O(n)$ time.

A4 Checking whether the evolution rule is applicable requires finding membrane h in \mathcal{C}' ($O(n)$ time) plus constant time to check polarization and multiplicity of a . When the rule is applicable, $O(k \log n)$ bits (lemma 3) are chosen to determine how many occurrences are rewritten and a subtraction (requiring $O(1)$ time on a random access machine) is performed to remove them. Then $O(n)$ multiplications and $O(n)$ additions are required to update the multiset inside h . The operands of such multiplications have size $O(k \log n)$ and $O(n)$ respectively, thus requiring $O(k \log n + n)$ time per multiplication on a random access machine. Finally, removing the rule from R' requires $O(n)$ time. Hence, step **A4** requires $O(nk \log n + n^2)$ time in total.

A5 Checking whether the communication rule is applicable requires $O(n)$ time as in step **A4**. Applying the rules itself is constant-time work, and removing all rules involving h from R' requires $O(n)$ time. Thus, step **A5** requires $O(n)$ time.

A6 This step requires $O(n)$ time, the analysis being the same as that of step **A5**.

A7 This step also requires $O(n)$ time, as in steps **A5** and **A6**.

A8 A traversal of the membrane structure ($O(n)$ time) is performed, while checking whether there are unused occurrences of each of the $O(n)$ kinds of objects. Finding a rule applicable to them requires scanning each of the $O(n)$ rules. The total time is thus $O(n^3)$.

A9 Here a traversal of the membrane structure is also performed. Moving a multiset requires $O(n)$ additions, while adjusting the membrane structure to reflect the deletion of a membrane requires $O(n)$ time. This step can thus be executed in $O(n^2)$ time.

A10 This step is simply a jump statement, requiring $O(1)$ time.

Keeping in mind that steps **A4–A7** are performed once for each of the $O(n)$ remaining rules, the total time required to simulate the k -th computation step of Π is then $O(n^2k \log n + n^3)$. Simulating all t steps then requires

$$\sum_{k=0}^t O(n^2k \log n + n^3) = O(n^2t^2 \log n + n^3t)$$

time. □

By using the complexity analysis of algorithm 1 we are now able to prove the converse of theorem 2.

Theorem 3. $\text{NPMC}_{\mathcal{N}\mathcal{A}\mathcal{M}} \subseteq \text{NP}$.

Proof. Let $L \in \text{NPMC}_{\mathcal{N}\mathcal{A}\mathcal{M}}$ and let $\Pi = \{\Pi_x \mid x \in \Sigma^*\}$ be a semi-uniform family of non-confluent P systems with active membranes (without division rules) deciding L in polynomial time; finally, let M be the deterministic Turing machine constructing Π in polynomial time.

We define a nondeterministic random access machine N deciding L in polynomial time. On input $x \in \Sigma^*$, with $|x| = n$, first N simulates M (in polynomial time) in order to obtain a description of Π_x , the P system deciding whether $x \in L$. Such description is, by hypothesis, of length $p(n)$ for some polynomial p . This representation is then translated via a parsing algorithm to the representation of P systems we fixed in this section; for instance, by using the CYK algorithm [3], we can perform this task in $O(p(n)^3)$ time. Now N runs algorithm 1 on Π_x , thus accepting when $x \in L$ and rejecting otherwise.

By hypothesis P system Π_x runs in time $O(q(n))$ for some polynomial q ; then, by theorem 2, the total running time of N is

$$O(p(n)^2q(n)^2 \log p(n) + p(n)^3q(n))$$

which is polynomial with respect to n . But then L is decided by a nondeterministic Turing machine in polynomial time, hence $L \in \text{NP}$. □

Thus, the two complexity classes are really the same:

Theorem 4. $\text{NPMC}_{\mathcal{N}\mathcal{A}\mathcal{M}} = \text{NP}$. □

5. Conclusions

We showed that the decision problem SAT can be solved in polynomial time using a semi-uniform family of non-confluent P systems without membrane division, proving that $\text{NP} \subseteq \text{NPMC}_{\mathcal{N}\mathcal{A}\mathcal{M}}$. Moreover, we proved that the reverse inclusion is also valid, by showing that non-confluent P systems without membrane division can be simulated in polynomial time by non-deterministic Turing machines; hence, we have that $\text{NPMC}_{\mathcal{N}\mathcal{A}\mathcal{M}}$ equals the complexity class **NP**.

Many research topics still remain to be investigated. The results of this paper might be improved by solving **NP**-complete problems with *uniform* families of non-confluent P systems without division. A characterization of **NP** in terms of confluent P systems is still missing, and a precise characterization of the class of problems solvable in polynomial time by using both kinds of membrane division rules (denoted by $\mathbf{NPMC}_{\mathcal{AM}}$) has not yet been found. It is known that this class contains **PSPACE** (as a consequence of the results in [14]) and is conjectured to be contained in **NEXP** (this could be proved by adding the simulation of division rules to algorithm 1, as the authors have already done for confluent P systems [12]).

Space complexity classes for P systems, first considered in [4] and then generalized to recognizer P systems with mutable membrane structure [11, 15], have not yet been investigated thoroughly in the non-confluent setting; this might be an area in which such devices differ from traditional computing models, where using nondeterminism usually does not reduce space requirements.

We also think that it might be useful to examine exponential-time complexity classes: how much power do we gain, and is the inability of increasing the depth of membrane structures a stumbling block in solving harder and harder problems?

Our hope is that the complexity-theoretic approach to membrane computing, besides providing insight on unconventional and parallel computing models, can also help us in clarifying the relations between classic complexity classes.

6. Acknowledgements

We would like to thank Mario de J. Pérez Jiménez for commenting on our results and pointing out an error in a preliminary version of algorithm 1.

References

- [1] A. Alhazov, C. Martín-Vide, L. Pan, Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes, *Fundamenta Informaticae* 58 (2) (2003) 67–77.
- [2] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A. Riscos-Núñez, F. J. Romero-Campero, P systems with active membranes, without polarizations and without dissolution: A characterization of P, in: C. Calude, M. J. Dinneen, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg (Eds.), UC, Vol. 3699 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 105–116.
- [3] J. E. Hopcroft, R. Motwany, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation* (2nd edition), Addison-Wesley, 2000.
- [4] O. H. Ibarra, On the computational complexity of membrane systems, *Theoretical Computer Science* 320 (1) (2004) 89–109.
- [5] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1993.

- [6] G. Păun, Computing with membranes, *Journal of Computer and System Sciences* 61 (1) (2000) 108–143.
- [7] G. Păun, Computing with membranes: Attacking NP-complete problems, in: I. Antoniou, C. Calude, M. J. Dinneen (Eds.), *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, Brussel, Belgium, 13–16 December 2000*, Springer, 2001, pp. 94–115.
- [8] G. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, 2002.
- [9] M. J. Pérez-Jiménez, Á. Romero-Jiménez, F. Sancho-Caparrini, Complexity classes in models of cellular computing with membranes, *Natural Computing* 2 (3) (2003) 265–285.
- [10] M. J. Pérez-Jiménez, Á. Romero-Jiménez, F. Sancho-Caparrini, The P versus NP problem through cellular computing with membranes, in: N. Jonoska, G. Păun, G. Rozenberg (Eds.), *Aspects of Molecular Computing*, Vol. 2950 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 338–352.
- [11] A. E. Porreca, A. Leporati, G. Mauri, C. Zandron, Introducing a space complexity measure for P systems, *International Journal of Computers, Communications & Control* 4 (3) (2009) 301–310.
- [12] A. E. Porreca, G. Mauri, C. Zandron, Complexity classes for membrane systems, *RAIRO Theoretical Informatics and Applications* 40 (2) (2006) 141–162.
- [13] P. Sosík, The computational power of cell division in P systems: Beating down parallel computers?, *Natural Computing* 2 (3) (2003) 287–298.
- [14] P. Sosík, A. Rodríguez-Patón, Membrane computing and complexity theory: A characterization of PSPACE, *Journal of Computer and System Sciences* 73 (1) (2007) 137–152.
- [15] A. Valsecchi, A. E. Porreca, A. Leporati, G. Mauri, C. Zandron, An efficient simulation of polynomial-space Turing machines by P systems with active membranes, in: *Pre-proceedings of the Tenth Workshop on Membrane Computing*, to appear.
- [16] C. Zandron, C. Ferretti, G. Mauri, Solving NP-complete problems using P systems with active membranes, in: I. Antoniou, C. Calude, M. J. Dinneen (Eds.), *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, Brussel, Belgium, 13–16 December 2000*, Springer, 2001, pp. 289–301.