# Recent complexity-theoretic results on P systems with active membranes

Giancarlo Mauri  Alberto Leporati  Antonio E. Porreca
Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{mauri,leporati,porreca,zandron}@disco.unimib.it

**Abstract**

Membrane systems, also called P systems, are an interesting class of parallel and distributed models of computation inspired by cell biology. They have been thoroughly investigated in the literature, both from the theoretical standpoint—analysing their computing power and efficiency—and as tools to model natural phenomena. In this paper we focus on the complexity theory of P systems with active membranes, a variant of P systems where the membranes themselves affect the applicability of rules and change (both in number and structurally) during computations. We summarise the main results on their space complexity, and describe some recent improvements related to time complexity, proved via a few general proof techniques.

## 1 Introduction

*P systems* (also called *membrane systems*) are parallel computing devices inspired by biology and, in particular, by the internal structure of cells and the biochemical processes that occur inside them [6]. The main feature of P systems is a hierarchically organised *membrane structure*, with membranes defining and separating regions of the system. Each region contains a number of *objects* (symbols taken from a finite alphabet) representing molecules. Since the multiplicity of molecules (i.e., the various concentrations of chemicals) matters for the functioning of the cell, we actually use *multisets* of objects (unordered sets with multiplicity) to describe the chemical environments.

P systems evolve by using a set of rules which describe chemical reactions and regulate the movement of objects between adjacent regions. In *P systems with active membranes* [7], the particular variant we investigate in this paper, the membranes themselves evolve during the computation: they can *dissolve*, releasing their contents in the external region, and *divide* as in the biological process of mitosis. The latter feature makes P systems with active membranes interesting from a complexity-theoretic standpoint: indeed, an exponential number of membranes operating in parallel can be generated in linear time, and this can be exploited to solve computationally hard problems (for instance, **NP**-complete problems) in polynomial time, by *trading space for time.*

Extensive investigation on P systems with active membranes has been carried out in the literature, particularly from the point of view of computability and computational complexity theory, as described in Chapters 11 and 12 of the recent *Handbook of Membrane Computing* [8]. The aim of this paper is to survey some of the more recent results in this area, obtained by the Milano research group on membrane computing. These results mainly involve *space complexity*, a subject that has been previously considered only from an informal standpoint, *non-confluence*, the notion corresponding to nondeterminsm for Turing machines, and further improvements on computability and complexity of restricted classes of P systems with active membranes. In particular, we show that

- the power of polynomial space coincides for Turing machines and P systems with active membranes;
- P systems with active membranes using only rules that move objects across membranes and division of nonelementary membranes (i.e., membranes containing other membranes) are not universal, but characterise a surprisingly large class of problems;
- P systems where only elementary membranes (i.e., membranes *not* containing other membranes) can divide are able to solve **PP**-complete problems;
- polynomial-time non-confluent P systems without membrane division characterise **NP**.

The paper is organised as follows: in Section 2 we give a formal definition of P systems with active membranes and how they can be used in order to solve decision problems; Section 3 contains a selection of proof techniques related to these computing devices, that also serve as examples; these techniques are then employed in Section 4 and 5, where the recent results (for space and time complexity, respectively) are described; finally, in Section 6 we make some concluding remarks and we present some directions for future research.

## 2 Definitions

We begin by defining P systems with active membranes; for a more formal definition we refer the reader to Chapter 12 of the *Handbook of Membrane Computing* [8].

**Definition 1.** A *P system with active membranes* of initial degree $d \geq 1$ is a tuple $\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_d, R)$, where:

- $\Gamma$ is a finite alphabet of symbols (the objects);
- $\Lambda$ is a finite set of labels for the membranes;
- $\mu$ is a membrane structure (i.e., a rooted *unordered* tree) consisting of $d$ membranes enumerated by $1, \ldots, d$; furthermore, each membrane is labeled by an element of $\Lambda$, not necessarily in a one-to-one way;

- $w_1, \ldots, w_d$ are strings over $\Gamma$, describing the initial multisets of objects placed in the $d$ regions of $\mu$;
- $R$ is a finite set of rules.

The membrane structure is usually represented symbolically as a string of balanced nested brackets, where each pair of corresponding open/close ones represents an individual membrane. The nesting of brackets corresponds to the ancestor-descendant relation of nodes in the tree; brackets at the same nesting levels can listed in any order.

Each membrane possesses, besides its label and position in $\mu$, another attribute called *electrical charge* (or polarization), which can be either neutral (0), positive (+) or negative (−) and is always neutral before the beginning of the computation.

The rules are of the following kinds:

- *Object evolution rules*, of the form $[a \to w]_h^\alpha$

  They can be applied inside a membrane labeled by $h$, having charge $\alpha$ and containing an occurrence of the object $a$; the object $a$ is rewritten into the multiset $w$ (i.e., $a$ is removed from the multiset in $h$ and replaced by every object in $w$).

- *Send-in communication rules*, of the form $a\,[\;]_h^\alpha \to [b]_h^\beta$

  They can be applied to a membrane labeled by $h$, having charge $\alpha$ and such that the external region contains an occurrence of the object $a$; the object $a$ is sent into $h$ becoming $b$ and, simultaneously, the charge of $h$ is changed to $\beta$.

- *Send-out communication rules*, of the form $[a]_h^\alpha \to [\;]_h^\beta\, b$

  They can be applied to a membrane labeled by $h$, having charge $\alpha$ and containing an occurrence of the object $a$; the object $a$ is sent out from $h$ to the outside region becoming $b$ and, simultaneously, the charge of $h$ is changed to $\beta$.

- *Dissolution rules*, of the form $[a]_h^\alpha \to b$

  They can be applied to a membrane labeled by $h$, having charge $\alpha$ and containing an occurrence of the object $a$; the membrane $h$ is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of $a$ becomes $b$.

- *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$

  They can be applied to a membrane labeled by $h$, having charge $\alpha$, containing an occurrence of the object $a$ but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes having label $h$ and charge $\beta$ and $\gamma$; the object $a$ is replaced, respectively, by $b$ and $c$ while the other objects in the initial multiset are copied to both membranes.

- *nonelementary division rules*, of the form

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

They can be applied to a membrane labeled by $h$, having charge $\alpha$, containing the positively charged membranes $h_1, \ldots, h_k$, the negatively charged membranes $h_{k+1}, \ldots, h_n$, and possibly some neutral membranes. The membrane $h$ is divided into two copies having charge $\beta$ and $\gamma$, respectively; the positive children are placed inside the former, their charge changed to $\delta$, while the negative ones are placed inside the latter, their charges changed to $\epsilon$. Any neutral membrane inside $h$ is duplicated and placed inside both copies.

Each instantaneous configuration of a P system with active membranes is described by the current membrane structure, including the electrical charges, together with the multisets located in the corresponding regions. A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules (inside each membrane any number of evolution rules can be applied simultaneously).
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or elementary division must be subject to exactly one of them (unless the current charge of the membrane prohibits it). The same reasoning applies to each membrane that can be involved to communication, dissolution, elementary or nonelementary division rules. In other words, the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.
- While all the chosen rules are considered to be applied simultaneously during each computation step, they are logically applied in a bottom-up fashion: first, all evolution rules are applied to the elementary membranes, then all communication, dissolution and division rules; then we proceed towards the root of the membrane structure. In other words, each membrane evolves only after its internal configuration has been updated.
- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

A *halting computation* of a P system is a finite sequence of configurations $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable by $\mathcal{C}_i$ via a single computation step, and no rules can be applied anymore in $\mathcal{C}_k$. A *non-halting* computation consists of infinitely many successive configurations $\vec{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$.

P systems can be used as *recognisers* by employing two specified objects YES and NO; exactly one of these must be sent out from the outermost membrane during each computation, in order to signal acceptance or rejection respectively; we also assume that all computations are halting. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. If this is not necessarily the case, then we have a *non-confluent* P system, and the overall result is established as for nondeterministic Turing machines: it is acceptance iff an accepting computation exists.

In order to solve decision problems (i.e., decide languages), we use *families* of recogniser P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ for some finite alphabet $\Sigma$. Each input $x$ is associated with a P system $\Pi_x$ that decides the membership of $x$ in the language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ is restricted, in order to be computable efficiently; usually one of the following *uniformity conditions* is imposed.

**Definition 2.** A family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be *semi-uniform* if the mapping $x \mapsto \Pi_x$ can be computed in polynomial time by a deterministic Turing machine.

The Turing machine can encode its output $\Pi_x$ by describing the membrane structure with brackets, the multisets as strings of symbols (in unary notation) and listing the rules one by one. However, any explicit encoding of $\Pi_x$ is allowed as output, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. We pose this restriction in order to enforce the initial membranes, initial objects and rules to be at most polynomial in number, as they can be super-polynomial if more compact representations (e.g., binary numbers) are used; this mimics a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed one by one, and require actual physical space in proportion to their number (see also Section 4 and [4]).

**Definition 3.** A family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be *uniform* if the mapping $x \mapsto \Pi_x$ can be computed by two deterministic polynomial-time Turing machines $M_1$ and $M_2$ as follows:

- The machine $M_1$, taking as input *the length $n$ of $x$* in unary notation, constructs a P system $\Pi_n$ with a distinguished input membrane (the P system $\Pi_n$ is common for all inputs of length $n$).
- The machine $M_2$, on input $x$, outputs a multiset $w_x$ (an encoding of the specific input $x$).
- Finally, $\Pi_x$ is simply $\Pi_n$ with $w_x$ added to the multiset placed inside its input membrane.

Notice how the uniform construction is just a restricted case of semi-uniform construction. The relations between the two kinds of uniformity have not completely been clarified yet; see [8, 4] for further details on uniformity conditions (including even weaker constructions).

Finally, we describe how time complexity for families of recogniser P systems is measured.

**Definition 4.** A uniform or semi-uniform family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to decide the language $L \subseteq \Sigma^\star$ in time $f \colon \mathbb{N} \to \mathbb{N}$ iff, for each $x \in \Sigma^\star$,

- the system $\Pi_x$ accepts if $x \in L$, and rejects if $x \notin L$;
- each computation of $\Pi_x$ halts within $f(|x|)$ computation steps.

## 3 Proof techniques

In this section we describe some general proof techniques related to P systems with active membranes. These serve both to demonstrate how a membrane system works, and to provide us with some arguments we will use in later sections to prove our main results.

### 3.1 Simulating register machines

In this section we describe how a deterministic register machine can be simulated by P systems; this shows that some classes of P systems are computationally universal, and is also useful to derive space complexity results in Section 4. The simulation we present here is a variant of the version published in [8] and [12]; it requires only object evolution and communication rules, and it is essentially sequential (recall that only one object per step can be communicated through a membrane).

The variant of register machines we consider [12] has a finite number of registers, and a finite number of instructions (labelled by consecutive integers starting from 1) of the following kinds:

- "$i\colon \textsc{inc}(r), j$" increments the value of register $r$ and jumps to instruction $j$;
- "$i\colon \textsc{dec}(r), j, k$" decrements the value of register $r$ and jumps to instruction $j$, if the value of $r$ is positive; otherwise, it jumps to instruction $k$ without changing the value of $r$.

The configuration of the machine is given by the tuple of the current values of its registers, together with the label of the current instruction. The computation starts by executing the instruction labelled by 1, and proceeds by executing the next one, until the execution reaches a label greater than the number of the last instruction; when this happens, the output can be read from a specified register.

Given a register machine $R$ with $k$ registers $r_1, \ldots, r_k$ having initial values $n_1, \ldots, n_k$, we construct a P system $\Pi_R$ having the following initial configuration:

$$\big[p_1 \; g \; [a^{n_1}]_{r_1}^0 \cdots [a^{n_k}]_{r_k}^0 \; [\,]_z^0\big]_s^0$$

In this case, we do not need an infinite uniform or semi-uniform family of P systems in order to simulate $R$; the system $\Pi_R$ is sufficient for all input sizes, and only the initial amounts $n_1, \ldots, n_k$ of objects have to be changed according to the input of $R$.

The membranes having label $r_1, \ldots, r_k$ represent the registers of $R$ (we call them *register-membranes* in the following), while $z$ is a *waiting membrane*. The value of a register $r_i$ of $R$ is represented by the multiplicity of the symbol $a$ inside the corresponding membrane $r_i$. We also use a *program counter object*, denoted by $p$, $p'$, or $p''$, having a subscript representing the label of the currently simulated instruction of $R$ (initially, the subscript is set to 1). Finally, the object $g$, located inside the outermost membrane, is used to generate further instances of object $a$ inside membrane $s$; these instances are used to perform increment operations.

An instruction of the form "$i\colon \textsc{inc}(r), j$" is implemented as follows: the program counter object $p_i$ sets the charge of membrane $r$ to positive, thus enabling a copy of object $a$ to enter it (incrementing the value of the simulated register) and restore the neutral charge. Then the program counter object exits $r$ as $p_j$. The corresponding rules are

$$p_i\ [\,]_r^0 \to [p_i']_r^+ \qquad a\ [\,]_r^+ \to [a]_r^0 \qquad [p_i']_r^0 \to [\,]_r^0\ p_j.$$

A supply of copies of $a$ sufficient to perform the increment operations is provided by object $g$, as long as membrane $s$ is neutrally charged, according to the object evolution rule

$$[g \to ga]_s^0. \tag{1}$$

Decrement instructions such as "$i$: DEC$(r), j, k$" are, in principle, implemented symmetrically with respect to increment instructions: the program counter sets membrane $r$ to negative, enabling a copy of $a$ to exit it. However, the final value of the subscript of the program counter object depends on whether the decrement operation was actually successful (i.e., whether a copy of $a$ appeared inside membrane $r$, and the neutral charge of $r$ was restored); in order to perform this check, we need to "wait" enough steps, in order to leave membrane $r$ available for the communication rule sending out $a$. This is accomplished by temporarily moving the program counter object to the waiting membrane $z$.

$$\begin{array}{ccc} p_i\ [\,]_r^0 \to [p_i']_r^- & [a]_r^- \to [\,]_r^0\ a & [p_i']_r^\alpha \to [\,]_r^\alpha\ p_i' \\ p_i'\ [\,]_z^0 \to [p_i']_z^0 & [p_i']_z^0 \to [\,]_z^0\ p_i'' & p_i''\ [\,]_r^\alpha \to [p_i'']_r^\alpha \\ [p_i'']_r^0 \to [\,]_r^0\ p_j & [p_i'']_r^- \to [\,]_r^0\ p_k. & \end{array}$$

The third and sixth rules have to be duplicated for both neutral and negative values of $\alpha$, since the first object to exit membrane $r$ is either $p_i'$ or $a$ according to a nondeterministic choice (that does not, however, affect the deterministic result of the decrement operation).

When the simulated register machine reaches a halting configuration, identified by a program counter larger than any instruction number, the program counter object changes the charge of membrane $s$ to positive, thus halting the production of further instances of the object $a$. This is accomplished by the communication rule $[p_i]_s^0 \to [\,]_s^+\ b$. If the register machine $R$ is used as an accepting device, the object $b$ sent out during the last step can be chosen among YES and NO (e.g., according to the value of $i$), thus ensuring that the P system simulating $R$ also behaves as a recogniser.

Using the above construction it is not hard to give a proof of the following result.

**Theorem 1.** *P systems with active membranes using only object evolution and communication rules are universal computing devices.* □

## 3.2 Encoding Boolean formulae

Our aim is to provide *uniform* solutions to decision problems related to Boolean formulae, where the membrane structure and the rules of the P systems solving it only depend on the *length* of the input instance, and only a multiset is used to encode the input itself. Hence, it is useful to devise an appropriate representation for Boolean formulae. In this section we describe an encoding for formulae in *ternary conjunctive normal form* (3CNF), i.e., a conjunction of disjunctive clauses of exactly three literals (optionally negated variables) where each variable occurs only once in each clause [12].

If $\varphi$ is a 3CNF formula of $m$ variables, then it consists of at most $8\binom{m}{3}$ clauses: indeed, there are $\binom{m}{3}$ sets of three variables out of $m$, and 8 ways in which they can be negated. After having totally ordered them (e.g., a lexicographic order allows us to compute the $i$-th clause in polynomial time with respect to $i$), the formula $\varphi$ can be represented as a string $\langle\varphi\rangle = c_1 \cdots c_n$ of $n = 8\binom{m}{3}$ bits, where $c_i = 1$ iff the $i$-th clause appears in $\varphi$.

For our purposes, the main advantage of this encoding is that we can recover the number $m$ of variables from the length $n$ of $\langle\varphi\rangle$ without the need to examine $\varphi$ itself: it is sufficient to find the unique positive integer root of the polynomial

$$p(m) = 8\tbinom{m}{3} - n = \tfrac{4}{3}m^3 - 4m^2 + \tfrac{8}{3}m - n.$$

This root can be computed in polynomial time with respect to $n$ (e.g., simply by evaluating $p(m)$ for all integers $m \le n$); if no such root is found, then we may conclude that the input is not well-formed and reject it on that basis.

## 3.3 Exploiting maximal parallelism

One of the main features of P systems with active membranes is their ability to divide membranes: an exponential number of membranes can be generated in linear time, and then evolve in parallel. It is well known that P systems using only evolution, communication and elementary division rules can solve **NP**-complete problems in polynomial time; in this section we recall how this result can be accomplished [19] by describing a family of P systems solving 3SAT. This family is designed in such a way that, with minor modifications to the construction, we can easily improve that result in Sections 5.1 and 5.2.

An abstract overview of the membrane computing algorithm for 3SAT is the following one.

**Algorithm 1.** Solving 3SAT on input $\varphi$, a 3CNF formula of $m$ variables.

1. *Generate* $2^m$ membranes using elementary division rules, each one containing a different truth assignment to the $m$ variables.

2. *Evaluate* $\varphi$ inside all $2^m$ membranes in parallel; send out an object $t$ from each membrane corresponding to a satisfying assignment.

3. *Output* the object YES if at least one instance of $t$ has been generated; otherwise output NO.

To all strings $x \in \{0,1\}^n$ such that $n = 8\binom{m}{3}$ for some $m$, representing well-formed inputs according to Section 3.2, we associate a single P system $\Pi_n$ having the following initial configuration:

$$\big[q_0\ r_0\ [p_0\ x_1 \cdots x_m]_h^0\big]_s^0.$$

The objects $x_1, \ldots, x_m$ represent unassigned variables, while $p_0$ and $q_0$ are used to implement timers, counting from zero.

The P system $\Pi_x$ associated with the specific input $x$ is obtained from $\Pi_n$, by placing inside membrane $h$ the following encoding of the input instance: a multiset consisting of all the objects $c_i$ such that the $i$-th clause in our ordering of clauses over $m$ variables (see Section 3.2) does *not* occur in the formula $\varphi$ represented by $x$. Starting from the initial configuration of $\Pi_x$, the computation proceeds according to the three phases of Algorithm 1.

**Phase 1** (Generate)**.** Each variable object $x_i$ in turn, in a nondeterministic order, causes a copy of membrane $h$ to divide; the object is rewritten as $t_i$ in one of the two new copies, and as $f_i$ inside the other, thus simulating, respectively, a true and a false assignment. The elementary division rules we need are

$$[x_i]_h^0 \to [t_i]_h^0[f_i]_h^0 \qquad\qquad \text{for } 1 \le i \le m. \tag{2}$$

While the membranes labelled by $h$ divide, generating $2^m$ copies in $m$ steps (each one containing a different assignment to the variables of $\varphi$), the timer objects $p_t$ count up to $m$ according to the rules

$$[p_t \to p_{t+1}]_h^0 \qquad\qquad \text{for } 1 \le t < m.$$

After $m$ steps, each copy of $p_m$ is sent out from its membrane, changing the charge to positive, and brought back in as $u_0$ during the next step via the following rules:

$$[p_m]_h^0 \to [\,]_h^+ \; p_m \qquad\qquad p_m \; [\,]_h^+ \to [u_0]_h^+.$$

While the instances of $p_m$ re-enter membrane $h$ (note that each copy of $h$ gets exactly one copy of $u_0$ due to maximal parallelism and the fact that only one communication rule per membrane can be applied), each instance of the objects $t_i$ and $f_i$ is replaced by a set of objects $c_k$ corresponding to the clauses satisfied by setting the variable $x_i$ to true or false respectively.

$$\begin{aligned}
&[t_i \to \{c_k : \text{the } k\text{-th clause contains } x_i\}]_h^+ && \text{for } 1 \le i \le m;\\
&[f_i \to \{c_k : \text{the } k\text{-th clause contains } \neg x_i\}]_h^+ && \text{for } 1 \le i \le m;
\end{aligned}$$

Notice how the set of clauses satisfied by the assignment to a particular variable can be computed from the length of the input $n = 8\binom{m}{3}$, due to our chosen encoding; it is sufficient to simply enumerate and check all clauses in order. The formula itself is not needed; as a consequence, these evolution rules can be computed in a uniform way.

After $m+2$ steps, each copy of membrane $h$ contains at least one occurrence of the object $c_k$ for each clause that is either occurring in $\varphi$ and satisfied by the assignment contained in that particular copy of $h$, or missing altogether from $\varphi$ (recall that, in that case, object $c_k$ is part of the input multiset)[1].

**Phase 2** (Evaluate)**.** According to the argument above, an assignment satisfies $\varphi$ exactly when all the objects $c_1, \ldots, c_n$ appear (with any nonzero multiplicity) inside the associated membrane $h$, including the objects $c_k$ corresponding to clauses not appearing in the formula $\varphi$.

In this phase we use the objects $u_j$ to count the number of clauses satisfied inside each membrane labelled by $h$. We do so by repeatedly checking whether an instance of $c_1$ appears, then decreasing the subscript of all the objects $c_k$;

[1]The missing clauses can be considered as satisfied by default, since the value "true" is the identity of conjunction.

this way, the next clause to be checked is always renamed as $c_1$. The procedure is repeated until we find an unsatisfied clause, or until all of them are found to be satisfied.

If $c_1$ actually occurs, then it is sent out from $h$, changing its charge to negative and allowing the subscript decrementing rules to be applied in the next step, when $u_j$ is also incremented.

$$\begin{aligned}
&[c_1]_h^+ \to [\,]_h^- \; c_1 && \\
&[c_k \to c_{k-1}]_h^- && \text{for } n \geq k > 0; \\
&[u_j \to u_{j+1}]_h^- && \text{for } 0 \leq j < n.
\end{aligned}$$

Notice how any extra copy of $c_1$ remaining inside $h$ is decremented to $c_0$ and ignored during the following steps.

Each object $c_1$ then re-enters $h$ (not necessarily the same instance of $h$ it came from, as any negatively charged one is equivalent) as the useless object $c_0$ and resets the charge to positive, thus restarting the checking loop. The corresponding rule is

$$c_1 \; [\,]_h^- \to [c_0]_h^+.$$

If all clauses are satisfied inside a certain copy of $h$, the counter reaches the value $u_n$ after $2n$ steps. This object can then be sent out as $t$, thus signalling the existence of a satisfying assignment for $\varphi$, using the rule

$$[u_n]_h^+ \to [\,]_h^+ \; t.$$

**Phase 3** (Output)**.** The objects $q_t$ and $r_t$, whose behaviour has not been described yet, implement two timers counting up during the first two phases of computation, i.e., for $\ell = (m+2) + (2n+1)$ steps, using the rules

$$\begin{aligned}
&[q_t \to q_{t+1}]_s^0 && \text{for } 0 \leq t \leq \ell; \\
&[r_t \to r_{t+1}]_s^\alpha && \text{for } 0 \leq t \leq \ell + 2 \text{ and } \alpha \in \{+, 0, -\}.
\end{aligned}$$

When the subscript of $q$ reaches $\ell$, the third phase begins. The object $q_\ell$ is sent out, changing the charge of membrane $s$ to positive and enabling a (possibly occurring) instance of $t$ to exit $s$; if this happens, the charge is changed again in order to record the fact that one of the assignments satisfies $\varphi$. The corresponding rules are

$$[q_\ell]_s^0 \to [\,]_s^+ \; q_\ell \qquad\qquad [t]_s^+ \to [\,]_s^- \; t.$$

Finally, the object $r_{\ell+2}$ is sent out as YES or NO according to the charge of $s$, i.e., depending on whether a satisfying assignment exists.

$$[r_{\ell+2}]_s^+ \to [\,]_s^+ \; \text{NO} \qquad\qquad [r_{\ell+2}]_s^- \to [\,]_s^- \; \text{YES}.$$

From the description of this algorithm, we can prove the following result.

**Theorem 2.** *There exists a uniform family of P systems with active membranes solving the* **NP***-complete problem* 3SAT *in polynomial time, without using nonelementary division or dissolution rules. Using the complexity class notation for P systems [9], this is written as* $3\text{SAT} \in \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$. □

## 3.4 Simulating P systems by Turing machines

Let us now describe how a (not necessarily confluent) recogniser P system with active membranes $\Pi$ can be simulated by a nondeterministic Turing machine, and analyse the time and space requirements [13].

For convenience, the actual simulating device we use is a nondeterministic random access machine (or RAM for short, see [5] for definitions), where the operations of addition and subtraction between registers are assumed to require constant time, while multiplication is performed in logarithmic time with respect to the size of the integers involved (i.e., linear time with respect to their length in bits); it is known that such a RAM can be simulated by a Turing machine with only a polynomial increase in time and space complexity [5].

The membrane structure is represented internally as a rooted tree, where each node corresponds to a membrane (the outermost one corresponds to the root) and possesses a list of children membranes and attributes describing the label and current charge of each membrane. The multisets of objects are represented as $k$-tuples of integers, where $k$ is the size of the alphabet and the $i$-th entry represents the multiplicity of the $i$-th object under any fixed ordering of the alphabet. Finally, the set of rules of $\Pi$ is represented as a list of records [16], each one containing a field for each component of the rule (i.e., the objects, and the labels and charges of the membranes involved in that rule).

The following is a high-level description of the simulation algorithm.

**Algorithm 2.** Simulating a P system with active membranes $\Pi$.

**A** For each rule in $R$, nondeterministically select the membranes and objects to which it has to be applied during the current simulated step.

**B** Check whether the rules have been chosen in a maximally parallel way; if this is not the case, abort the simulation by rejecting.

**C** Apply the rules in the current configuration of $\Pi$, starting from the elementary membranes and moving up towards the outermost one.

**D** If either YES or NO were sent out from the outermost membrane, then halt and accept or reject accordingly; otherwise, simulate the next step by jumping to **A**.

The nondeterministic assignment of rules to objects and membranes in step **A** can be described in details as follows.

**A**$_1$ Let $R'$ be the set of currently unused rules; set $R' \leftarrow R$.

**A**$_2$ While $R' \neq \emptyset$ pick a rule $r \in R'$, otherwise go to step **B**.

> **A**$_3$ If $r = [a \to w]_h^\alpha$ then, for each membrane of the form $[\ ]_h^\alpha$, nondeterministically choose an amount $k$ of copies of object $a$ to be rewritten into $w$; this amount can be anywhere from 0 to the multiplicity of $a$ in that particular membrane. Subtract $k$ from the number of available copies of $a$ in $h$.
>
> **A**$_4$ If $r = a\ [\ ]_h^\alpha \to [b]_h^\beta$ then, for each available membrane of the form $[\ ]_h^\alpha$ having an available instance of $a$ in the region immediately outside, nondeterministically choose whether to apply $r$ and, in that

case, assign those particular instances of $a$ and $h$ to $r$ making them unavailable.

**A**$_5$ If $r = [a]_h^\alpha \to [\ ]_h^\beta\ b$ or $r = [a]_h^\alpha \to b$ or $r = [a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$ then, for each available membrane of the form $[\ ]_h^\alpha$ containing an available instance of $a$, nondeterministically choose whether to apply $r$ and, in that case, assign those particular instances of $a$ and $h$ to $r$ making them unavailable.

**A**$_6$ If $r = \big[[\ ]_{h_1}^+ \cdots [\ ]_{h_k}^+ [\ ]_{h_{k+1}}^- \cdots [\ ]_{h_n}^-\big]_h^\alpha \to \big[[\ ]_{h_1}^\delta \cdots [\ ]_{h_k}^\delta\big]_h^\beta \big[[\ ]_{h_{k+1}}^\varepsilon \cdots [\ ]_{h_n}^\varepsilon\big]_h^\gamma$ then, for each available membrane of the form $[\ ]_h^\alpha$ containing the available membranes $[\ ]_{h_1}^+, \ldots, [\ ]_{h_k}^+, [\ ]_{h_{k+1}}^-, \ldots [\ ]_{h_n}^-$ and possibly some neutral available membranes, nondeterministically choose whether to apply $r$ or not; if so, assign all the involved membranes to $r$ making them unavailable.

**A**$_7$ Set $R' \leftarrow R' - \{r\}$ and go back to step **A**$_2$.

Deciding in step **B** whether the assignment of rules computed in **A** is indeed maximally parallel simply requires us to check, for each membrane and object still available in the current configuration, whether there exists a rule in $R$ that could be applied to them. If this is the case, then the choice of rules is not maximal, and we abort the simulation by rejecting. This does not change the behaviour of the simulating device, since the existence of one accepting computation is not influenced by adding further rejecting ones (in other words, the RAM simulating $\Pi$ is designed to have an accepting computation iff $\Pi$ has one).

The actual application of the rules chosen in **A** is performed in step **C** according to the following sub-steps.

**C**$_1$ For each $r = [a \to w]_h^\alpha$, remove $k$ instances of $a$, where $k$ is the multiplicity chosen for $r$ in step **A**$_3$, and add $k$ times the objects in $w$.

**C**$_2$ For $r = a\ [\ ]_h^\alpha \to [b]_h^\beta$ remove an instance of $a$ from the external region, add an instance of $b$ to the internal one, and change the charge to $\beta$.

**C**$_3$ For $r = [a]_h^\alpha \to [\ ]_h^\beta\ b$ remove an instance of $a$ from the internal region, add an instance of $b$ to the external region, and change the charge to $\beta$.

**C**$_4$ For $r = [a]_h^\alpha \to b$ move all the objects from the internal region to the external one, replacing an instance of $a$ with $b$, then, remove the membrane $h$ from the current configuration; the children of $h$ are adopted by its parent.

**C**$_5$ For $r = [a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$ duplicate membrane $h$ and its contents, replacing an instance of $a$ by one of $b$ on one side, and by an instance of $c$ on the other; set the charges of the new membranes to $\beta$ and $\gamma$ respectively.

**C**$_6$ For $r = \big[[\ ]_{h_1}^+ \cdots [\ ]_{h_k}^+ [\ ]_{h_{k+1}}^- \cdots [\ ]_{h_n}^-\big]_h^\alpha \to \big[[\ ]_{h_1}^\delta \cdots [\ ]_{h_k}^\delta\big]_h^\beta \big[[\ ]_{h_{k+1}}^\varepsilon \cdots [\ ]_{h_n}^\varepsilon\big]_h^\gamma$ create a new instance of $h$, placing inside it all the negative membranes $h_{k+1}, \ldots, h_n$ from the current membrane, and a *deep copy* of all neutral membranes inside $h$ (i.e., including the substructure having them as the root, including its contents); finally, update the charges according to $r$.

Finally, in step **D** we decide whether the computation of $\Pi$ has ended, giving the same result if this is the case, and simulating the next step of $\Pi$ otherwise.

## 4 Space complexity of P systems

We now turn our attention to results related to the notion of space complexity. This topic has been formally introduced in the membrane computing setting [11] in order to analyse the time/space trade-off that is common when solving computationally hard problems via P systems. In the next two sections we show that P systems and Turing machines operating in polynomial space solve exactly the same class of problems, and that P systems using only communication and nonelementary membrane division solve a very large but non-universal class of decision problems. These results are adapted from [12, 13] and [15] respectively.

The size $|\mathcal{C}|$ of a configuration $\mathcal{C}$ of a P system is given by the sum of the number of objects and the number of membranes; this definition assumes that every component of the system requires some fixed amount of physical space, thus approximating (up to a polynomial) the size of a real cell. The space required by a halting computation $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is then given by

$$|\vec{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\},$$

and the space required by a P system $\Pi$ is

$$|\Pi| = \max\{|\vec{\mathcal{C}}| : \vec{\mathcal{C}} \text{ is a computation of } \Pi\}.$$

Finally, we say that a family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ operates in space $f\colon \mathbb{N} \to \mathbb{N}$ if $|\Pi_x| \leq f(|x|)$ for all $x \in \Sigma^\star$.

### 4.1 Polynomial space complexity

The main data structure required for the simulation of Section 3.4 is the tree describing the membrane structure, augmented with a tuple of integers for each node describing the multiset of objects in the corresponding membrane. The auxiliary data structures, required for instance to keep track of what objects and membranes are still available in step **A**, clearly do not require an amount of space larger than that of the augmented tree itself [13]. As a consequence, the following result holds.

**Theorem 3.** *A (confluent or non-confluent) P system with active membranes $\Pi$ operating in space $f(n)$ can be simulated by a deterministic Turing machine in space $O(p(f(n)))$ for some polynomial $p$.* $\square$

We are also able to prove that polynomial-space families of P systems can solve **PSPACE**-complete problems: as a consequence, the classes of languages decided in polynomial space by P systems and by Turing machines coincide. In order to prove this result, we employ the simulation of register machines by P systems shown in Section 3.1.

Consider the **PSPACE**-complete decision problem Q3SAT: given a Boolean formula $\varphi$ of $m$ variables in 3CNF, decide whether the following quantified version of $\varphi$ holds:

$$\forall x_1 \exists x_2 \forall x_3 \cdots Q_m x_m \ \varphi.$$

Here the universal and existential quantifiers are alternated, and $Q_m$ is $\forall$ or $\exists$ depending on whether $m$ is odd or even, respectively. We assume the input formula $\varphi$ is encoded as in Section 3.2, as a string of bits $c_1 \cdots c_n$, with $n = 8\binom{m}{3}$, where $c_j = 1$ iff the $j$-th clause occurs in $\varphi$.

Quantified Boolean formulae can be easily evaluated recursively in polynomial space; however, for every fixed value of $m$, recursion can be replaced by $m$ nested *for* loops: the $i$-th outermost loop sets the truth value $t_i$ of variable $x_i$ from 0 to 1 (i.e., false or true) and evaluates the partial result

$$\begin{aligned} r_i = {} & Q_{i+1}x_{i+1} \cdots Q_m x_m \ \varphi(t_1, \ldots, t_{i-1}, 0, x_{i+1}, \ldots, x_m) \Diamond \\ & Q_{i+1}x_{i+1} \cdots Q_m x_m \ \varphi(t_1, \ldots, t_{i-1}, 1, x_{i+1}, \ldots, x_m) \end{aligned}$$

where $t_1, \ldots, t_{i-1}$ is the partial truth assignment fixed by the $i-1$ outermost loops, and the connective $\Diamond$ is $\wedge$ if $Q_i$ is $\forall$, and $\vee$ if it is $\exists$. Inside the $m$-th loop, all variables have been assigned a truth value, and $\varphi$ is evaluated according to that particular assignment. By construction, the final value $r_1$ is the truth value of the whole quantified formula.

```
$r_1 \leftarrow 1$;
for $t_1 \leftarrow 0$ to 1 do
    $r_2 \leftarrow 0$;
    for $t_2 \leftarrow 0$ to 1 do
        $\ddots$
            $r_m \leftarrow e$
            for $t_m \leftarrow 0$ to 1 do
                $r \leftarrow \varphi(t_1, \ldots, t_m)$;
                $r_m \leftarrow r_m \Diamond r$
            end;
        $\cdot^{\cdot^{\cdot}}$
        $r_2 \leftarrow r_2 \vee r_3$
    end;
    $r_1 \leftarrow r_1 \wedge r_2$
end
```

The instruction "$r \leftarrow \varphi(t_1, \ldots, t_m)$" inside the innermost loop is implemented by computing the value $v_j$ of the $j$-th clause as a disjunction of the corresponding literals, and then computing the conjunction of $v_1, \ldots, v_n$. If the $j$-th clause does not occur in $\varphi$, we set $v_j$ to 1 (the identity of conjunction) in order to avoid affecting the overall result.

```
if $c_1$ then
    $v_1 \leftarrow t_1 \vee t_2 \vee t_3$
else
    $v_1 \leftarrow 1$
end;
$\vdots$
if $c_n$ then
    $v_n \leftarrow \neg t_{m-2} \vee \neg t_{m-1} \vee \neg t_m$
else
    $v_n \leftarrow 1$
end;
$r \leftarrow v_1 \wedge \cdots \wedge v_n$
```

This algorithm requires exponential time, as $\varphi$ is evaluated under all $2^m$ truth assignments to its variables; however, the space needed is linear, as only $O(m+n) = O(n)$ Boolean variables are used.

The language we used to describe the formula evaluation procedure is simple enough to be easily "compiled" [12] into a polynomial-time uniform family of register machines $\mathbf{R} = \{R_n : n \in \mathbb{N}\}$. Each program variable corresponds to a different register; in particular, the input bits $c_1 \cdots c_n$ are placed in different registers. Since all variables assume only Boolean values (0 or 1), the total space required by $R_n$ is only linear with respect to $n$.

The family $\mathbf{R}$ can then be simulated by a uniform family of P system $\mathbf{\Pi}$, consisting of a base structure $\Pi_n$ for each $n = 8\binom{m}{3}$ (a direct simulation of $R_n$) together with an input multiset $a_{i_1}, \ldots, a_{i_k}$, where $i_1, \ldots, i_k$ are the clauses occurring in the input formula. Each object $a_i$ is moved, rewritten as $a$, to the membrane corresponding to the program variable $c_i$ during the first computation step, by using communication rules of the form $a_i\ [\ ]_{c_i}^0 \to [a]_{c_i}^0$. Once these registers have been initialised, the computation proceeds as in Section 3.1. As the maximum number of instances of $a$ is polynomial and fixed before the beginning of the simulation, there is no need to generate a copy of $a$ in the outermost membrane during each step: we can place the right amount of objects directly during the construction phase and avoid the object $g$ and rule (1) altogether. When the simulated register machine halts, we check whether the value of register $r_1$ is nonzero, and send out from the outermost membrane the object YES or NO accordingly. Notice that, although the P systems encode the values stored by the simulated register machine in unary, the number of objects required is polynomial, as each register only contains 0 or 1.

This construction can be used to prove the following results.

**Theorem 4.** *The* **PSPACE**-*complete problem* Q3SAT *can be solved by a uniform family of P systems working in polynomial space; in symbols,* Q3SAT $\in$ $\mathbf{PMCSPACE}_{\mathcal{AM}}$. $\square$

**Corollary 1.** *Turing machines and P systems with active membranes working in polynomial space solve exactly the same class of problems; in symbols,* $\mathbf{PMCSPACE}_{\mathcal{AM}} = \mathbf{PSPACE}$. *The equality to* **PSPACE** *also holds for nonconfluent P systems, and it is independent of the presence or absence of membrane dissolution and division rules.* $\square$

## 4.2 Characterising "tetrational space"

In this section we consider a restricted variant of P systems with active membranes, where only communication rules (in both directions) and nonelementary division rules are allowed, and analyse their computing power when no resource bounds are imposed. By space complexity arguments, we can prove that such P systems are not computationally universal, while powerful enough to decide a very large class of languages [15].

Let us first consider how large the membrane structure of one of these restricted P systems $\Pi$, having an encoding of length $m$ (including the membrane structure in bracket notation, the initial objects in unary and the set of rules), can become during the computation. The initial membrane structure clearly consists of at most $m$ membranes arbitrarily arranged, and is then (ignoring labels and electrical charges) necessarily a subtree of the complete $m$-ary tree

$T_m$ of $m$ levels. In order to simplify the calculations, we analyse how a membrane structure having exactly the same shape as $T_m$ can evolve in the worst case; this will provide us an upper bound on the maximal size of any membrane structure of $m$ nodes. The tree $T_m$ has the distinctive feature that the number of descendants of a node is only a function of the level where it is located (i.e., all nodes on the same level are roots of identical subtrees)[2]; hence, these trees can be described by a $k$-ary function $T(n_1, n_2, \ldots, n_k)$ where $n_i$ denotes the number of nodes on the $i$-th level of the tree (of course we always have $n_1 = 1$). In particular, we have

$$T_m = T(1, m, m^2, \ldots, m^{m-1}).$$

A nonelementary division rule applied to a certain membrane $h$ creates a new sibling membrane, containing all negatively charged children membranes of $h$, together with a copy of all neutral ones; the positive children membranes and another copy of the neutral ones are left inside the original membrane. It is clear, then, that a maximal number of new membranes is created when there is exactly one positively charged and one negatively charged children membranes, and the division rules are of the following form:

$$[[\,]_{h_1}^{+}[\,]_{h_1}^{-}]_h^{\alpha} \to [[\,]_{h_1}^{\delta}]_h^{\beta}[[\,]_{h_1}^{\epsilon}]_h^{\gamma}. \tag{3}$$

If this rule is applied to a membrane $h$ with $k$ children, where one of them is a positive membrane with label $h_1$, another is a negative one with label $h_2$, and all the others are neutral, then the original membrane $h$ is split into two membranes with $k - 1$ children.

From now on, we ignore electrical charges and assume that nonelementary division rules are always applicable; furthermore, we assume that all such rules have the same form as (3), i.e., only two children membranes appear on the left-hand side.

The number of newly created membranes is maximised if the nonelementary division rules are applied in a "bottom-up" order, that is, first to all membranes of level $m-1$, then to all membranes of level $m-2$, and so on. On the other hand, all membranes at the same level can be simultaneously divided without affecting each other; we assume this behaviour in order to simplify the calculations. All other cases either ultimately reduce to this one, or to a situation where a smaller number of membranes is created.

By applying nonelementary division rules to the nodes of

$$T(1, m, m^2, \ldots, m^{m-2}, m^{m-1})$$

on the level above the leaves, they are duplicated but each one of them loses one child, and we obtain

$$T\big(1, m, m^2, \ldots, 2 \cdot m^{m-2}, 2 \cdot m^{m-2} \cdot (m-1)\big)$$

By repeating the process we obtain

$$T\big(1, m, m^2, \ldots, 2^2 \cdot m^{m-2}, 2^2 \cdot m^{m-2} \cdot (m-2)\big)$$
$$T\big(1, m, m^2, \ldots, 2^3 \cdot m^{m-2}, 2^3 \cdot m^{m-2} \cdot (m-3)\big)$$

[2]This feature is sometimes called *uniformity* [3].

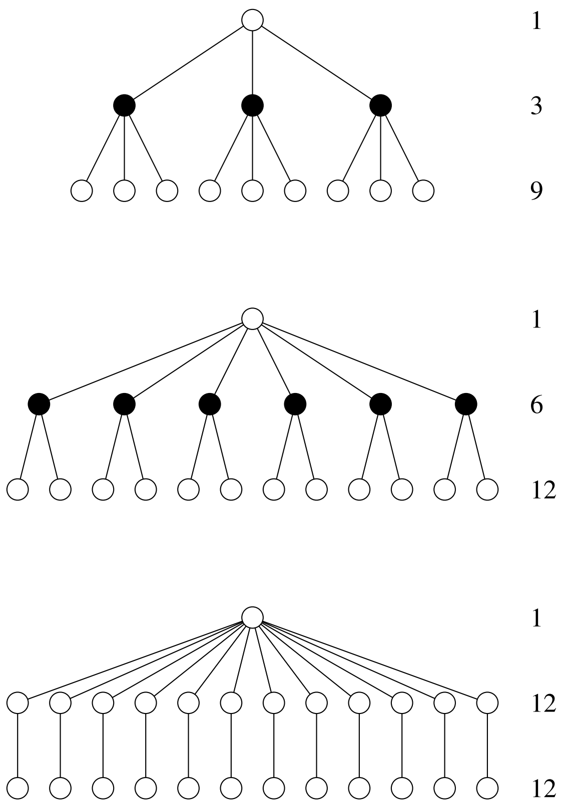

Figure 1: Dividing the membranes of level 2 in a full ternary tree. The membranes to be divided during the next step are highlighted in black, and the numbers on the right represent the number of nodes on the corresponding level.

and so on, until

$$T(1, m, m^2, \ldots, 2^{m-1} \cdot m^{m-2}, 2^{m-1} \cdot m^{m-2}).$$

Now the last two levels of the tree have the same number of nodes, i.e., each node at level $m-1$ has a single child: no further division is possible. The division process is depicted, for $m = 3$, in Figure 1.

Repeating the whole process on level $m-2$ produces the following tree:

$$T\bigl(m, m^2, \ldots, m^{m-4}, 2^{2^{m-1} \cdot m-1} \cdot m^{m-3}, \\ 2^{2^{m-1} \cdot m-1} \cdot m^{m-3}, 2^{2^{m-1} \cdot m-1} \cdot m^{m-3}\bigr).$$

In general, applying all possible nonelementary division rules to the nodes in the lowest $k$ levels of the tree produces $m^{m-k} \cdot f(k)$ nodes on those levels, where $f(k)$ is recursively defined as

$$\begin{aligned} f(1) &= 1 \\ f(k) &= 2^{f(k-1) \cdot m-1} && \text{for } 1 < k \leq m-1. \end{aligned}$$

Thus, the tree obtained by applying all nonelementary division rules on all levels (excluding the root level) has a total of $1 + (m-1) \cdot m \cdot f(m-1)$ nodes: $m-1$ levels with $m \cdot f(m-1)$ nodes, plus the root.

An upper bound for this value can be given by using the operation of *tetration*, defined for all natural numbers $b$, $n$ as

$$ {}^{n}b = \begin{cases} 1 & \text{if } n = 0 \\ b^{({}^{n-1}b)} & \text{if } n > 0 \end{cases}, \qquad \text{that is,} \qquad {}^{n}b = \underbrace{b^{b^{\cdot^{\cdot^{\cdot^{b}}}}}}_{n \text{ times}}. $$

It can be proved by induction that the size of the tree obtained following the above procedure is bounded by ${}^{O(m^2)}2$.

A similar bound on the maximum number of objects in the configuration of $\Pi$ can be given by recalling that the only way to increase the amount of objects is nonelementary membrane division itself, as communication rules simply move them from a region to another. Assume, for the sake of finding an upper bound, that *all* the objects are moved in such a way to be duplicated whenever a nonelementary division rule is applied. The number of actual applications of these rules is ${}^{O(m^2)}2$; assuming an initial number of objects bounded by $m$, the final amount is then $m \cdot 2^{({}^{O(m^2)}2)}$, i.e., ${}^{O(m^2)}2 \cdot m$.

Using the arguments of this section and the simulation algorithm of Section 3.4 it is possible to prove the following result.

**Theorem 5.** *P systems with active membranes using only communication and nonelementary division rules, and having an encoding of length $m$, can be simulated by deterministic Turing machines operating in space ${}^{p(m)}2$ for some polynomial $p$.* □

This result implies that these restricted P systems are not computationally universal. Indeed, denote by **PTETRA** the class of problems decidable in space ${}^{p(m)}2$ for some polynomial $p$ by Turing machines. This class, which according to Theorem 5 is an upper bound to the computational power of this variant of P systems, does not exhaust the whole class of recursive languages: a harder language can be constructed by diagonalisation, as in the space hierarchy theorem [5].

We can prove that the **PTETRA** upper bound can actually be reached. In order to do so, we modify the simulation of register machines given in Section 3.1. The main structure of the simulation is unaltered; however, we have to find an alternative way to produce instances of object $a$ inside the outermost membranes, since object evolution rules like (1) are not allowed. This process can be performed by using only communication and nonelementary division rules, although the maximum number of objects produced in this way will be bounded by a tetrational function, as described above.

To this end, we design a P system "module" $\Pi_{d,w}$ ($d, w \in \mathbb{N}$) sending out tetrationally many copies of $a$ from its outermost membrane. The initial membrane structure of $\Pi_{d,w}$ is the following:

$$ \overbrace{\Big[\Big[\cdots\Big[\Big[}^{d-1 \text{ levels}} \underbrace{[\,]_{h_d} \cdots [\,]_{h_d}}_{w \text{ copies}} [[\,]_\ell]_\ell \Big]_{h_{d-1}} [[\,]_\ell]_\ell [\,]_c \Big]_{h_{d-2}} \cdots [[\,]_\ell]_\ell [\,]_c \Big]_{h_2} [\,]_e \Big]_{h_1} $$

The role of each membrane can be described as follows:

- Membranes $h_1, \ldots, h_{d-1}$ are arranged in a linear chain, whereas all copies of $h_d$ are children of $h_{d-1}$. These membranes constitute the main structure of $\Pi_{d,w}$, and (with the exception of the root $h_1$) they are going to be divided into tetrationally many copies. The remaining membranes perform auxiliary roles.
- The nested membranes $\big[[\,]_\ell\big]_\ell$ occur inside each membrane $h_2, \ldots, h_{d-1}$; these membranes are traversed back and forth by the object $\ell$ when we need to "wait" for the nonelementary membrane division of $h_i$ to occur.
- Membrane $c$ occurs inside each membrane $h_2, \ldots, h_{d-2}$; these membranes host the object $c$, which is released whenever all possible nonelementary divisions of the membrane labeled by $h_{i+1}$ have occurred.
- Membrane $e$, placed inside $h_1$, performs the same role as the membranes labeled by $c$, with the difference that object $e$ is sent out when all possible nonelementary divisions have occurred on *each* level.

This membrane structure initially contains the following objects:

- Each membrane $h_1, \ldots, h_{d-2}$ contains the objects $\ell$ and $r$.
- Membranes $h_2, \ldots, h_{d-2}$ contain the object $c'$, while $h_{d-1}$ contains $c$ and $h_1$ contains $e'$.

During the first computation step, all objects $c'$ enter the membrane labeled by $c$ near them by using the communication rule $c'\ [\,]_c^0 \to [c]_c^+$, thus also changing its charge to positive; the same happens to $e'$, which uses the rule $e'\ [\,]_e^0 \to [e]_e^+$.

We can now discuss how the level-by-level division procedure is performed. Let $i \in \{3, \ldots, d-1\}$, and assume that all divisions of membranes labeled by $h_{i-1}, \ldots, h_{d-1}$ have been carried out and the object $c$ has been released; let $k$ be the number of children of membrane $h_i$ at this moment. The configuration is then the following one (where only the relevant membranes and objects are displayed, and those not currently involved are omitted):

$$\mathcal{C}_0 = \ell r \Big[ c \underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k \text{ copies}} \big[[\,]_\ell^0\big]_\ell^0 \Big]_{h_i}^0 [c]_c^+$$

Object $c$ exits $h_i$ via $[c]_{h_i}^0 \to [\,]_{h_i}^+\ \#$, where # denotes a "junk" object (i.e., it does not appear on the left-hand side of any rule; as such, it is irrelevant and omitted from the following configurations):

$$\mathcal{C}_1 = \ell r \Big[ \underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k \text{ copies}} \big[[\,]_\ell^0\big]_\ell^0 \Big]_{h_i}^+ [c]_c^+$$

When $h_i$ is positive, object $\ell$ enters via $\ell\ [\,]_{h_i}^+ \to [\ell_1]_{h_i}^-$:

$$\mathcal{C}_2 = r \Big[ \ell_1 \underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k \text{ copies}} \big[[\,]_\ell^0\big]_\ell^0 \Big]_{h_i}^- [c]_c^+$$

When $h_i$ is negative, object $r$ can enter via $r\ [\ ]_{h_i}^{-} \to [r_1]_{h_i}^{+}$ and, simultaneously, $\ell_1$ enters one of the copies of $h_{i+1}$ and sets its charge to positive, using the rule $\ell_1\ [\ ]_{h_{i+1}} \to [\ell_2]_{h_{i+1}}^{+}$:

$$\mathcal{C}_3 = \Big[r_1 [\ell_2]_{h_{i+1}}^{+} \underbrace{[\ ]_{h_{i+1}}^{0} \cdots [\ ]_{h_{i+1}}^{0}}_{k-1 \text{ copies}} [[\ ]_\ell^0]_\ell^0\Big]_{h_i}^{+} [c]_c^{+}$$

The evolution of the computation from this point on depends on whether there exists another neutral membrane labeled by $h_{i+1}$ (i.e., on whether $k > 1$). First, assume that this is indeed the case. While $\ell_2$ exists its membrane by using the rule $[\ell_2]_{h_{i+1}}^{+} \to [\ ]_{h_{i+1}}^{+}\ \ell_3$, object $r_1$ enters another one via $r_1\ [\ ]_{h_{i+1}}^{0} \to [r_2]_{h_{i+1}}^{0}$:

$$\mathcal{C}_4 = \Big[\ell_3 [\ ]_{h_{i+1}}^{+} [r_2]_{h_{i+1}}^{0} \underbrace{[\ ]_{h_{i+1}}^{0} \cdots [\ ]_{h_{i+1}}^{0}}_{k-2 \text{ copies}} [[\ ]_\ell^0]_\ell^0\Big]_{h_i}^{+} [c]_c^{+}$$

Now $r_2$ exits $h_{i+1}$, via $[r_2]_{h_{i+1}}^{0} \to [\ ]_{h_{i+1}}^{+}\ r_3$, setting its charge to negative; in the mean time, $\ell_3$ enters the outermost membrane labeled by $\ell$ using $\ell_3\ [\ ]_\ell^0 \to [\ell_4]_\ell^0$:

$$\mathcal{C}_5 = \Big[r_3 [\ ]_{h_{i+1}}^{+} [\ ]_{h_{i+1}}^{-} \underbrace{[\ ]_{h_{i+1}}^{0} \cdots [\ ]_{h_{i+1}}^{0}}_{k-2 \text{ copies}} [\ell_4 [\ ]_\ell^0]_\ell^0\Big]_{h_i}^{+} [c]_c^{+}$$

Now $h_i$ contains both a negative and a positive membrane, and it can divide using $[[\ ]_{h_{i+1}}^{+} [\ ]_{h_{i+1}}^{-}]_{h_i}^{+} \to [[\ ]_{h_{i+1}}^{0}]_{h_i}^{0} [[\ ]_{h_{i+1}}^{0}]_{h_i}^{0}$; simultaneously, $\ell_4$ moves one membrane deeper via $\ell_4\ [\ ]_\ell^0 \to [\ell_5]_\ell^0$.

$$\mathcal{C}_6 = \overbrace{\Big[r_3 \underbrace{[\ ]_{h_{i+1}}^{0} \cdots [\ ]_{h_{i+1}}^{0}}_{k-1 \text{ copies}} [[\ell_5]_\ell^0]_\ell^0\Big]_{h_i}^{0}}^{2 \text{ copies}} [c]_c^{+}$$

In two steps, via $[\ell_5]_\ell^0 \to [\ ]_\ell^0\ \ell_6$ and $[\ell_6]_\ell^0 \to [\ ]_\ell^0\ \ell_7$, the two instances of $\ell_7$ are moved back to $h_i$:

$$\mathcal{C}_8 = \overbrace{\Big[\ell_7 r_3 \underbrace{[\ ]_{h_{i+1}}^{0} \cdots [\ ]_{h_{i+1}}^{0}}_{k-1 \text{ copies}} [[\ ]_\ell^0]_\ell^0\Big]_{h_i}^{0}}^{2 \text{ copies}} [c]_c^{+}$$

The objects $\ell_7$ now exit $h_i$ via $[\ell_7]_{h_i}^{0} \to [\ ]_{h_i}^{-}\ \ell_8$, followed by $r_3$ via the rule $[r_3]_{h_i}^{-} \to [\ ]_{h_i}^{+}\ r_4$. Hence, in two steps we obtain

$$\mathcal{C}_{10} = \overbrace{\ell_8 r_4 \Big[\underbrace{[\ ]_{h_{i+1}}^{0} \cdots [\ ]_{h_{i+1}}^{0}}_{k-1 \text{ copies}} [[\ ]_\ell^0]_\ell^0\Big]_{h_i}^{+}}^{2 \text{ copies}} [c]_c^{+}$$

Now $\ell_8$ goes back to $h_i$ via $\ell_8\ [\,]_{h_i}^+ \to [\ell_1]_{h_i}^-$; during the next step, $r_4$ follows via $r_4\ [\,]_{h_i}^- \to [r_1]_{h_i}^+$ and $\ell_1$ uses the rule $\ell_1\ [\,]_{h_{i+1}} \to [\ell_2]_{h_{i+1}}^+$ as described earlier:

$$\mathcal{C}_{12} = \overbrace{\Big[r_1\,[\ell_2]_{h_{i+1}}^+\,\underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k-2\text{ copies}}\,[[\,]_\ell^0]_\ell^0\Big]_{h_i}^+}^{2\text{ copies}}\ [c]_c^+$$

Notice that the two instances of $\ell_8$ and $r_4$ do not necessarily re-enter the same membrane $h_i$ from which they came, due to nondeterminism; however, this makes no real difference as the two copies of $h_i$ are identical, and the rules involving the two objects are applied in parallel.

Configuration $\mathcal{C}_{12}$ is completely analogous to $\mathcal{C}_3$, only with *two* occurrences of $h_i$, each one containing *one less* occurrence of $h_{i+1}$. Hence, we have entered a cycle which repeats in parallel the whole process of division of $h_i$, until each of these membranes only remains with one child. When this event occurs, the configuration becomes

$$\mathcal{C}'_3 = \overbrace{\Big[r_1\,[\ell_2]_{h_{i+1}}^+\,[[\,]_\ell^0]_\ell^0\Big]_{h_i}^+}^{2^{k-1}\text{ copies}}\ [c]_c^+$$

Object $r_1$ has no neutral membrane $h_{i+1}$ to enter, and remains stuck. The other object exits $h_{i+1}$ and travels through the two membranes labelled by $\ell$, but no division occurs (since we do not have negative membranes inside $h_i$); after five steps we obtain

$$\mathcal{C}'_8 = \overbrace{\Big[\ell_7 r_1\,[\,]_{h_{i+1}}^+\,[[\,]_\ell^0]_\ell^0\Big]_{h_i}^+}^{2^{k-1}\text{ copies}}\ [c]_c^+$$

By noticing the positive (instead of neutral) charge of $h_i$, we may deduce that no division occurred and, by construction, that each copy of $h_i$ only has one child labeled by $h_{i+1}$. The objects $\ell_7$ are sent out of $h_i$ and renamed to $\ell_9$ via $[\ell_7]_{h_i}^+ \to [\,]_{h_i}^0\ \ell_9$; this rule also sets $h_i$ to neutral.

$$\mathcal{C}'_9 = \overbrace{\ell_9\Big[r_1\,[\,]_{h_{i+1}}^+\,[[\,]_\ell^0]_\ell^0\Big]_{h_i}^+}^{2^{k-1}\text{ copies}}\ [c]_c^+$$

One of the instances of $\ell_9$ enters membrane $c$ via $\ell_9\ [\,]_c^+ \to [\ell_{10}]_c^0$; by setting its charge to neutral, it blocks all the other copies of $\ell_9$ forever. Object $c$ is now free to exit to membrane $h_{i-1}$ via $[c]_c^0 \to [\,]_c^0\ c$.

During the next step, object $c$ is sent out again via $[c]_{h_{i-1}}^0 \to [\,]_{h_{i-1}}^+\ \#$, thus changing the polarization to positive and activating the objects $\ell$ and $r$ one level above (see configurations $\mathcal{C}_0$ and $\mathcal{C}_1$). The whole process of division is then restarted, this time involving membrane $h_{i-1}$.

The division of membrane $h_{d-1}$, the first to which the procedure is applied, is started by the object $c$ located inside it (recall that $h_{d-1}$ contains this object instead of $c'$) by using the rule $[c]_{h_{d-1}}^0 \to [\,]_{h_{d-1}}^+\ \#$, as it happens on the other levels. On the other hand, on the outermost level (membrane $h_1$) it is the object $e$ to be released, instead of $c$, in order to signal that *every* level has completed the division process. An example of the division process, for $w = 3$ and $d = 4$, is shown in Figure 2.

Choosing $w = 3$ is enough to produce at least ${}^{d-2}2$ membranes on every level below the root, and in particular that amount of leaves is generated. By adding a child membrane containing the object $a$ to each membrane having label $h_d$ in $\Pi_{d,3}$ we obtain ${}^{d-2}2$ instances of $a$, located inside the elementary membranes. The object $e$ can then be used to release them, by changing the charge of the membranes that contain them; the instances of $a$ can then move through the membrane structure of $\Pi_{d,3}$ until they are sent out from $h_1$.

By placing $\Pi_{d,3}$ (for large enough $d$) inside membrane $s$ of the P system described in Section 3.1, we can obtain the instances of $a$ we need to simulate register machines where the value of each register is at most ${}^{p(n)}2$ for some polynomial $p$. Since register machines simulate Turing machines by using asymptotically the same space [2], we obtain a precise characterisation of the computing power of P systems using only communication and nonelementary division: they solve exactly the problems in **PTETRA**.

## 5 Solving problems in polynomial time

Now we turn our attention to the efficient (polynomial-time) solution of hard decision problems. We show two recent results in this area, the first of which shows that P systems with active membranes where nonelementary membranes cannot divide, who were only known to be able to solve **NP**-complete problems in polynomial time (see, e.g., [19]), are actually more powerful, at least under the common assumption that $\mathbf{NP} \neq \mathbf{PP}$.

The second result shows that membrane division can be avoided altogether while solving **NP**-complete problems if non-confluence (strong nondeterminism) is allowed. In this case, however, we obtain a precise characterisation of **NP**, as this kind of P systems can be simulated in polynomial time by nondeterministic Turing machines.

### 5.1 PP-complete problems

In Section 3.3 we described how membrane division rules and maximal parallelism can be exploited in order to solve the **NP**-complete problem 3SAT in polynomial time, by checking whether at least one assignment (among exponentially many) satisfies the input formula. It is, however, possible to solve even (supposedly) harder problems by comparing the number of satisfying assignments to a threshold [14]: the **PP**-complete problems.

**Definition 5.** The complexity class **PP** consists of all the decision problem solvable in polynomial time by nondeterministic Turing machines $N$ with the following acceptance criterion: $N$ accepts an input iff *more than half* of its computations are accepting.

An example of **PP**-complete problem is the following variant of 3SAT [1].

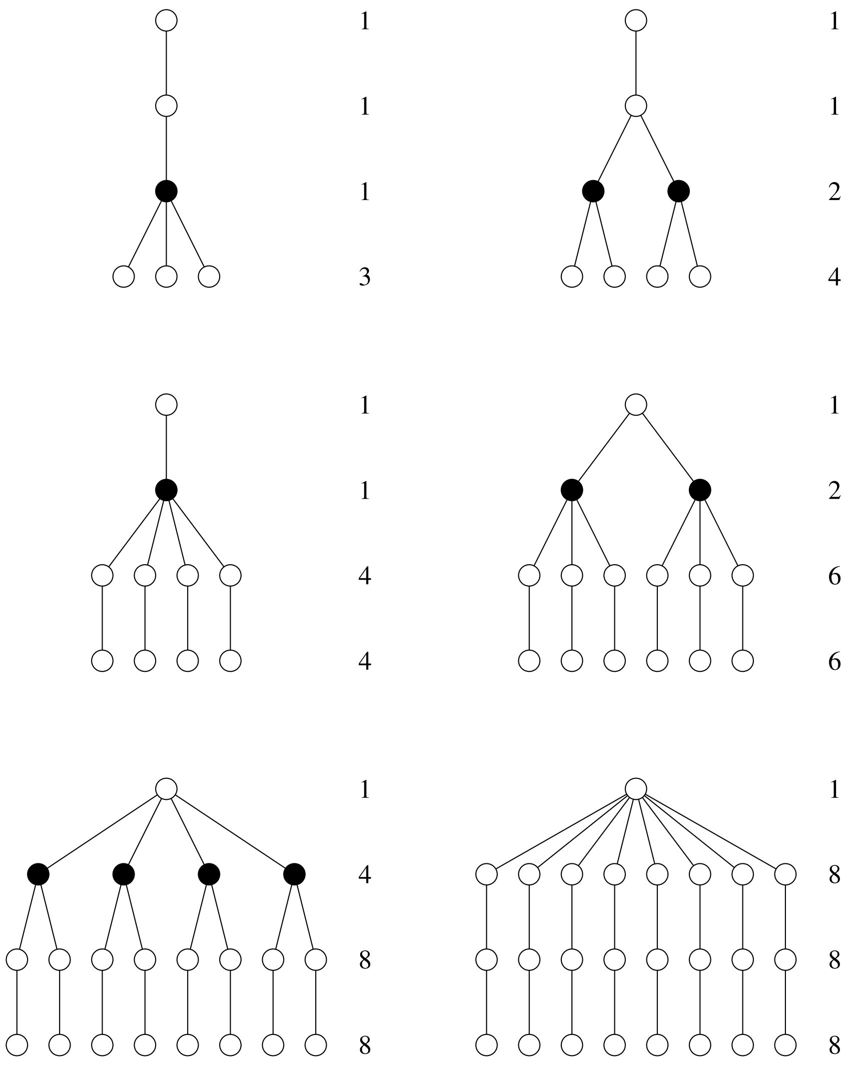

Figure 2: The division process occurring in module $\Pi_{4,3}$. Here the auxiliary membranes $\ell$, $c$ and $e$ are not shown, and only the main ones $h_1, \ldots, h_4$ are shown. As in Figure 1, the membranes to be divided during the next step are highlighted in black, and the numbers on the right represent the number of membranes on the corresponding level.

**Definition 6.** The problem Sqrt-3SAT consists of deciding whether at least $\sqrt{2^m}$ assignments (over a total of $2^m$) satisfy a 3CNF Boolean formula of $m$ variables.

Algorithm 1 can be used for solving Sqrt-3SAT by adding a further phase (number 3 in the following description).

**Algorithm 3.** Solving Sqrt-3SAT on input $\varphi$, a 3CNF formula of $m$ variables.

1. *Generate* $2^m$ membranes using elementary division rules, each one containing a different truth assignment to the $m$ variables.
2. *Evaluate* $\varphi$ inside all $2^m$ membranes in parallel; send out an object $t$ from each membrane corresponding to a satisfying assignment.
3. *Erase* $\lceil\sqrt{2^m}\rceil - 1$ instances of $t$ (or all of them, if the amount is less than this threshold).
4. *Output* the object YES if at least one instance of $t$ is still present; otherwise output NO.

The initial configuration of the P system $\Pi_n$ of Section 3.3 is modified as follows:

$$\big[q_0\ r_0\ [p_0\ x_1 \cdots x_m]_h^0\ [b_{i_1}]_d^0[b_{i_2}]_d^0 \cdots [b_{i_k}]_d^0\big]_s^0.$$

Here the number of membranes having label $d$, and the subscripts of the objects $b_i$ depend on the number of variables of $\varphi$: indeed, the values $k$ and $i_1 < i_2 < \cdots < i_k$ are the unique non-negative integers such that

$$\lceil\sqrt{2^m}\rceil - 1 = 2^{i_1} + 2^{i_2} + \cdots + 2^{i_k},$$

i.e., they correspond to the $k$ positions where bit 1 occurs in the binary notation of $\lceil\sqrt{2^m}\rceil - 1$; notice that $k$ is bounded by $\lceil\log(\lceil\sqrt{2^m}\rceil - 1)\rceil$. During the first phase (generation) each object $b_i$ is used to produce $2^i$ copies of membrane $d$ by division, in such a way that by the end of the phase exactly $\lceil\sqrt{2^m}\rceil - 1$ copies exist. This is accomplished via the following rules:

$$[b_i]_d^0 \to [b_{i-1}]_d^0[b_{i-1}]_d^0 \qquad \text{for } 1 \le i \le \lceil\log(\lceil\sqrt{2^m}\rceil - 1)\rceil.$$

The second phase (evaluation) is performed exactly as in Section 3.3.

The erasing phase consists of exactly one computation step, where each membrane labelled by $d$ absorbs one instance of object $t$ from the outermost membrane, becoming positively charged in the process (in order to ensure that at most one object is absorbed):

$$t\ [\ ]_d^0 \to [t]_d^+.$$

After this step, copies of $t$ remain inside the outermost membrane iff the initial number was at least $\sqrt{2^m}$.

In the fourth (output) phase, we check whether this is indeed the case. This phase is also performed as in Section 3.3, the only difference being that the timer limit $\ell$ has to be increased by one, due to the extra step required by phase 3.

The ability to solve a **PP**-complete problem (together with a few technical details related to the notion of uniformity [10]) implies the following result.

**Theorem 6.** $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$*, where* $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$ *is the class of problems solvable in polynomial time by uniform families of P systems with active membranes, using neither nonelementary division nor dissolution rules.* □

## 5.2 Nondeterminism and non-confluence

In this section we give a brief summary of the following result [17]: membrane division can be exchanged for nondeterminism in order to solve **NP**-complete problems in polynomial time. Furthermore, the converse holds: polynomial-time non-confluent P systems without division can be simulated by nondeterministic Turing machines with a polynomial slowdown.

In order to solve 3SAT without requiring membrane division, we change the first phase of Algorithm 1 as follows: we *guess* a truth assignment to the variables instead of generating every one of them, then we check whether it satisfies the input formula $\varphi$ as in the original algorithm. This way, only one copy of membrane $h$ is needed, containing a single truth assignment. The acceptance condition of non-confluent P systems (i.e., we accept iff at least one computation is accepting) ensures that the final result is the correct one.

Formally, we only need to replace the division rules in (2)

$$[x_i]_h^0 \to [t_i]_h^0 [f_i]_h^0 \qquad\qquad \text{for } 1 \leq i \leq m$$

by a pair of conflicting object evolution rules for each $i$:

$$[x_i \to t_i]_h^0 \qquad\qquad [x_i \to f_i]_h^0 \qquad\qquad \text{for } 1 \leq i \leq m.$$

The following result is then immediate.

**Theorem 7.** *The decision problem* 3SAT *can be solved in polynomial time by a uniform family of non-confluent P systems with active membranes without using membrane division rules (and, in particular, using only polynomial space).* □

In order to show that polynomial-time non-confluent P systems without division can be simulated in polynomial time by nondeterministic Turing machines, we need to analyse the time complexity of the simulation algorithm of Section 3.4 for this particular case.

Consider a P system $\Pi$ having an encoding of length $m$. Since no division rules are allowed, the only way to increase the space required by $\Pi$ is to use object evolution rules; these can rewrite an object into at most $m$ objects (due to the length of the encoding of $\Pi$). Since the initial number of objects is, again, at most $m$, we can conclude that each configuration of $\Pi$ after $t$ steps contains at most $m^{t+1} = 2^{O(t \log m)}$ objects, which can be represented in binary notation using $O(t \log m)$ space.

Simulating a P system requires multiple traversals of the membrane structure; in this case, its size is polynomially bounded, since no division rules are allowed, hence it can be traversed in polynomial time. In step **A** of Algorithm 2, the traversal is performed once for each rule. Since checking whether a rule can be applied only requires checking a finite number of items, and nondeterministically choosing an integer of length $O(t \log m)$ in step $\mathbf{A}_3$, the whole step **A** can be carried out in polynomial time. Step **B** scans the set of rules once for each membrane, checking whether any rule can be applied to unused objects or membranes. Step **C** performs another (bottom-up) traversal of the membrane structure; applying the rules requires performing simple arithmetic operations on $O(t \log m)$-bit integers to update the multisets, which can also performed in polynomial time. Finally, step **D** only requires constant time, if we keep a Boolean flag for the halting condition. Since any computation of $\Pi$ halts in a polynomial number of steps by hypothesis, the whole simulation of a non-confluent P system without division can be carried out in polynomial time by a nondeterministic Turing machine.

This simulation, together with Theorem 7, allows us to obtain the following characterisation:

**Theorem 8.** *The class of problems solvable in polynomial time by P systems with active membranes without division rules is exactly* **NP**. □

## 6 Conclusions and open problems

We provided an update to some of the current research trends related to the computational complexity of P systems with active membranes.

Research on the topics of this paper has not been exhausted yet. We believe an analysis of the computing power of P systems with active membranes operating in sub- and super-polynomial space (e.g., logarithmic and exponential space) is worth carrying out. Furthermore, a characterisation of the problems solvable in polynomial time without nonelementary division rules is still missing, and neither the **PP** lower bound, nor the **PSPACE** upper bound proved in [18] are known to be strict.

Other research directions involve the characterisation of small complexity classes like $\mathbf{AC}^0$, **L** and **NL** in terms of P systems with active membranes *without charges* [4]; avoiding charges apparently limits the power of these devices in a significant way.

## References

[1] Delbert D. Bailey, Víctor Dalmau, and Phokion G. Kolaitis. Phase transitions of PP-complete satisfiability problems. *Discrete Applied Mathematics* 155(12):1627–1639, 2007.

[2] Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory* 2(3):265–283, 1968.

[3] Fan R.K. Chung, Ronald L. Graham, and Donald Coppersmith. On trees containing all small trees. In Gary Chartrand, editor, *The Theory and Applications of Graphs*, pages 265–272. John Wiley & Sons, 1981.

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

[5] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1993.

[6] Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

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

[8] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[9] Mario J. Pérez-Jiménez, Álvaro Romero-Jiménez, and Fernando Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2(3):265–284, 2003.

[10] Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. Elementary active membranes have the power of counting. In Miguel Ángel Martínez-del-Amor, Gheorghe Păun, Ignacio Pérez-Hurtado, Francisco José Romero-Campero, and Luis Valencia-Cabrera, editors, *Ninth Brainstorming Week on Membrane Computing*, RGNC Report 1/2011, pages 329–342. Fénix Editora, 2011.

[11] Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. Introducing a space complexity measure for P systems. *International Journal of Computers, Communications & Control*, 4(3):301–310, 2009.

[12] Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. P systems with active membranes: Trading time for space. *Natural Computing* 10(1):167–182, 2011.

[13] Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. P systems with active membranes working in polynomial space. *International Journal of Foundations of Computer Science* 22(1):65–73, 2011.

[14] Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron, P systems with elementary active membranes: Beyond NP and coNP. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, editors, *Membrane Computing, 11th International Conference, CMC 2010*, *Lecture Notes in Computer Science* 6501, pages 338–347, 2011.

[15] Antonio E. Porreca, Alberto Leporati, Claudio Zandron, On a powerful class of non-universal P systems with active membranes. In Yuan Gao, Hanlin Lu, Shinnosuke Seki, and Sheng Yu, editors, *Developments in Language Theory, 14th International Conference, DLT 2010*, *Lecture Notes in Computer Science* 6224, pages 364–375, 2010.

[16] Antonio E. Porreca, Giancarlo Mauri, and Claudio Zandron. Complexity classes for membrane systems. *RAIRO Theoretical Informatics and Applications*, 40(2):141–162, 2006.

[17] Antonio E. Porreca, Giancarlo Mauri, and Claudio Zandron. Non-confluence in divisionless P systems with active membranes. *Theoretical Computer Science*, 411(6):878–887, 2010.

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

[19] Claudio Zandron, Claudio Ferretti, and Giancarlo Mauri. Solving NP-complete problems using P systems with active membranes. In I. Antoniou, C. Calude, M.J. Dinneen, editors, *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference*, pages 289–301. Springer, 2001.