Elementary Active Membranes Have the Power of Counting

Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron

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

Summary. We prove that uniform families of P systems with active membranes operating in polynomial time can solve the whole class of **PP** decision problems, without using nonelementary membrane division or dissolution rules. This result also holds for families having a stricter uniformity condition than the usual one.

1 Introduction

P systems with active membranes [9] are known to solve computationally hard problems in polynomial time by trading space for time: an exponential number of membranes is created in polynomial time by using division rules, and then massive parallelism is exploited, e.g., to explore the whole solution space of an **NP**-complete problem in parallel.

When we allow nonelementary division rules, i.e., rules that can be applied to membranes containing further membranes, even **PSPACE**-complete problems become solvable in polynomial time [10, 2]. The general idea is that nonelementary division allows us to construct a binary tree-shaped membrane structure, isomorphic to the parse tree of the formula resulting from the expansion of universal and existential quantifiers into conjunctions and disjunctions, according to the equivalences

$$\forall x\, \varphi(x) \Leftrightarrow \varphi(0) \wedge \varphi(1) \qquad\qquad \exists x\, \varphi(x) \Leftrightarrow \varphi(0) \vee \varphi(1).$$

We also know that no problem outside **PSPACE** can be solved in polynomial time, as this is also an upper bound [11]: in symbols, we have $\mathbf{PMC}_{\mathcal{AM}} = \mathbf{PSPACE}$.

On the other hand, when no division at all is allowed the resulting P systems can be shown to be no more powerful than polynomial-time Turing machines ("Milano Theorem" [12]).

The "intermediate" case, when the only membranes that can divide are elementary (i.e., leaves of the tree corresponding to the membrane structure), is possibly

the most interesting one. The exponential number of membranes that may be created cannot be structured into a binary tree: hence, the algorithm above can only be applied to formulae having just one kind of quantifier. This is enough to solve the SAT problem [12] and its complement (which are respectively **NP**- and **coNP**-complete), where only existentially (resp., universally) quantified variables are allowed.[1] However, the corresponding complexity class $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$ still lacks a characterisation in terms of Turing machines. Alhazov et al. [1] have shown how **PP**- and #**P**-complete problems can be solved without nonelementary division, but their result is not directly related to the class $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n})}$, as it requires some form of post-processing or the use of non-standard rules. In this paper, we improve the previous $\mathbf{NP} \cup \mathbf{coNP}$ lower bound to **PP** within the standard framework of active membranes. This is an improved version of the paper "P systems with active membranes: Beyond NP and coNP" presented by the authors and the Eleventh International Conference on Membrane Computing [7].

2 Preliminaries

We use P systems with restricted elementary active membranes, which are defined as follows.

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

- Γ *is a finite alphabet of symbols (the objects);*
- Λ *is a finite set of labels for the membranes;*
- μ *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* Λ*, not necessarily in a one-to-one way;*
- $w_1, \ldots, w_d$ *are strings over* Γ*, describing the initial multisets of objects placed in the* d *regions of* μ*;*
- R *is a finite set of rules.*

Each membrane possesses, besides its label and position in μ, 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 α 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).

[1] Some further partial results relating quantifier alternations and nonelementary division depth, albeit in the slightly different framework of P systems with active membranes without charges, have been obtained [8].

- *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 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 charge of h is changed to β.
- *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 α 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 β.
- *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 α, 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 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.

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 Π is a finite sequence of configurations $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, 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 $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted.

P systems can be used as *recognisers* by employing two distinguished 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 $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_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 and uniformly for each input length.

Definition 2. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* (polynomial-time) uniform *if the mapping $x \mapsto \Pi_x$ can be computed by two deterministic polynomial-time Turing machines F (for "family") and E (for "encoding") as follows:*

- *The machine F, taking as input* the length n of x *in unary notation, constructs a P system Π_n with a distinguished input membrane (the P systems structure Π_n is common for all inputs of length n).*
- *The machine E, on input x, outputs a multiset w_x (an encoding of the specific input x).*
- *Finally, Π_x is simply Π_n with w_x added to the multiset placed inside its input membrane.*

Notice that this definition of uniformity is possibly weaker than the other one commonly used in membrane computing [6], where the Turing machine F maps each input x to a P system $\Pi_{s(x)}$, where $s\colon \Sigma^\star \to \mathbb{N}$ is a measure of the size of the input (in our case, $s(x)$ is always $|x|$). In particular, complexity classes defined using this restricted uniformity condition are not always formally known to be closed under polynomial-time reductions[2]. See [4] for further details on uniformity conditions, including constructions using weaker devices than polynomial-time Turing machines.

Any explicit encoding of Π_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. This restriction is enforced in

[2] This might complicate proofs of inclusions among complexity classes, although one can usually find a proof not relying on closure under polynomial-time reductions, as in the present paper.

order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space in proportion to their number. For instance, the membrane structure can be represented by brackets, and the multisets as strings (i.e., in unary notation); this is a *permissible encoding* in the sense of [4].

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

Definition 3. *A uniform family of P systems* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *is said to decide the language* $L \subseteq \Sigma^\star$ *(in symbols* $L(\boldsymbol{\Pi}) = L$*) in time* $f\colon \mathbb{N} \to \mathbb{N}$ *iff, for each* $x \in \Sigma^\star$*,*

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

In this paper we use uniform families of P systems to solve a variant of the SAT problem. Hence, we set the relevant notation and describe how Boolean formulae can be encoded in order to simplify a uniform solution.

Given a set of $m \geq 3$ variables $X_m = \{x_1, \ldots, x_m\}$, the number of clauses of 3 variables (without repeated variables, and ignoring permutations of literals) is given by $8\binom{m}{3}$, the number of 3-element subsets times the 2^3 ways to negate them. Hence, a 3CNF formula φ can be encoded as an $8\binom{m}{3}$-bit string, where the i-th bit is 1 iff the i-th clause (under some fixed ordering) appears in φ. Notice that $8\binom{m}{3} = \frac{4}{3}m^3 - 4m^2 + \frac{8}{3}m$ is a polynomial.

In this paper, we will order the clauses according to the enumeration printed by the recursive algorithm of Figure 1.

Example 1 (Encoding). The clauses over 4 variables $X_4 = \{x_1, \ldots, x_4\}$, in the order given by the algorithm above, are the following ones:

$x_1 \vee x_2 \vee x_3$	$x_1 \vee x_2 \vee \overline{x}_3$	$x_1 \vee \overline{x}_2 \vee x_3$	$x_1 \vee \overline{x}_2 \vee \overline{x}_3$
$\overline{x}_1 \vee x_2 \vee x_3$	$\overline{x}_1 \vee x_2 \vee \overline{x}_3$	$\overline{x}_1 \vee \overline{x}_2 \vee x_3$	$\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$
$x_1 \vee x_2 \vee x_4$	$x_1 \vee x_2 \vee \overline{x}_4$	$x_1 \vee \overline{x}_2 \vee x_4$	$x_1 \vee \overline{x}_2 \vee \overline{x}_4$
$\overline{x}_1 \vee x_2 \vee x_4$	$\overline{x}_1 \vee x_2 \vee \overline{x}_4$	$\overline{x}_1 \vee \overline{x}_2 \vee x_4$	$\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4$
$x_1 \vee x_3 \vee x_4$	$x_1 \vee x_3 \vee \overline{x}_4$	$x_1 \vee \overline{x}_3 \vee x_4$	$x_1 \vee \overline{x}_3 \vee \overline{x}_4$
$\overline{x}_1 \vee x_3 \vee x_4$	$\overline{x}_1 \vee x_3 \vee \overline{x}_4$	$\overline{x}_1 \vee \overline{x}_3 \vee x_4$	$\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_4$
$x_2 \vee x_3 \vee x_4$	$x_2 \vee x_3 \vee \overline{x}_4$	$x_2 \vee \overline{x}_3 \vee x_4$	$x_2 \vee \overline{x}_3 \vee \overline{x}_4$
$\overline{x}_2 \vee x_3 \vee x_4$	$\overline{x}_2 \vee x_3 \vee \overline{x}_4$	$\overline{x}_2 \vee \overline{x}_3 \vee x_4$	$\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4$

Then, the formula

$$\varphi = (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_4)$$

is encoded as the following sequence of $8\binom{4}{3} = 32$ bits

```
PRINT-CLAUSES(m)
    IF m > 3 THEN
        PRINT-CLAUSES(m − 1)
    END
    FOR i ← 1 TO m − 2 DO
        FOR j ← i + 1 TO m − 1 DO
            PRINT "x_i ∨ x_j ∨ x_m"
            PRINT "x_i ∨ x_j ∨ x̄_m"
            PRINT "x_i ∨ x̄_j ∨ x_m"
            PRINT "x_i ∨ x̄_j ∨ x̄_m"
            PRINT "x̄_i ∨ x_j ∨ x_m"
            PRINT "x̄_i ∨ x_j ∨ x̄_m"
            PRINT "x̄_i ∨ x̄_j ∨ x_m"
            PRINT "x̄_i ∨ x̄_j ∨ x̄_m"
        END
    END
END
```

Fig. 1. A recursive, polynomial-time algorithm that enumerates all clauses of 3 out of m variables.

$$\varphi = 0100\ 0001\ 0000\ 0000\ 0001\ 0000\ 0000\ 0010$$

because the clauses actually appearing are the 2nd, 8th, 20th, and 31st ones.

Besides being computable in polynomial time with respect to m, this ordering has the following important property: the sequence of clauses over m variables is a prefix of the sequence of clauses over m' variables whenever $m' \geq m$. As a consequence, each formula over m variables can also be considered as a formula over m' variables by padding its encoding to the correct length. For instance, the formula φ of Example 1 can be interpreted as a formula over *five* variables $x_1, \ldots, x_5$ if its encoding is padded to length $8\binom{5}{3} = 80$ by a string of zeroes, i.e., as $\varphi \cdot 0^{48}$.

We now consider the following decision problem.

Problem 1 (Threshold-3SAT). Given a Boolean formula φ over m variables and a non-negative integer $k < 2^m$, do more than k assignments (out of 2^m) satisfy it?

Notice that we can force all valid instances (φ, k) of Problem 1 to have a description of length exactly $8\binom{m}{3} + m$ for some m, as every number in the range $[0, 2^m)$ can be represented using m bits. This will be useful in the next section.

Proposition 1. THRESHOLD-3SAT *is* **PP**-*hard.*

Proof. We reduce the following standard **PP**-complete problem [5, p. 256] to THRESHOLD-3SAT.

Problem 2 (Majority-SAT). Given a Boolean formula φ in CNF, having c clauses over m variables and such that each variable occurs at most once per clause, do more than half the assignments (i.e., more than 2^{m-1} assignments) satisfy it?

The reduction is similar to that from SAT to 3SAT described in [3, p. 48]. We first transform φ into a formula having *at most* three literals per clause. Observe that φ is satisfied iff the formula obtained by replacing a clause of $p > 3$ literals $\bigvee_{i=1}^{p} \ell_i$ with

$$(y \Leftrightarrow \ell_1 \vee \ell_2) \wedge \left(y \vee \bigvee_{i=3}^{p} \ell_i \right)$$

is also satisfied, assuming y is a new variable. In CNF, that is equivalent to

$$(\bar{\ell}_1 \vee y) \wedge (\bar{\ell}_2 \vee y) \wedge (\ell_1 \vee \ell_2 \vee \overline{y}) \wedge \left(y \vee \bigvee_{i=3}^{p} \ell_i \right).$$

This substitution doubles the number of total assignments of the formula, due to the addition of a new variable, but the number of *satisfying* ones is left unchanged, as the value of y is forced to be equal to $\ell_1 \vee \ell_2$. The substitution decreases by one the number of literals of the initial clause; by repeating the process $p - 3$ times, and then again to any other clause having more than three literals, we obtain a formula φ' having at most three literals per clause, and the same number of satisfying assignments as φ. The number of variables of φ' is bounded by $m + cm$.

Next, we transform every clause of one or two literals into a clause of exactly three. A clause of a single literal ℓ is replaced by

$$(\ell \vee z_1 \vee z_2) \wedge (\ell \vee \overline{z}_1 \vee z_2) \wedge (\ell \vee z_1 \vee \overline{z}_2) \wedge (\ell \vee \overline{z}_1 \vee \overline{z}_2),$$

where z_1 and z_2 are new variables, which is clearly satisfied iff ℓ is. Each replacement like this one multiplies by $2^2 = 4$ the number of satisfying assignments of the whole formula, as the values of z_1 and z_2 are actually irrelevant.

A clause of two literals $\ell_1 \vee \ell_2$ is replaced by

$$(\ell_1 \vee \ell_2 \vee z) \wedge (\ell_1 \vee \ell_2 \vee \overline{z}),$$

where z is a new variable, which is also equivalent to the original clause but doubles the number of satisfying assignments of the formula.

Call φ'' the formula obtained from φ' by replacing single and 2-literal clauses by conjunctions of 3-literal clauses as described above, and let q be the number of variables added in the process (notice that q is $O(cm)$). Then it should be clear that φ has more than 2^{m-1} satisfying assignments iff φ' does, and the latter is equivalent to φ'' having more than 2^{m+q-1} satisfying assignment.

Since the mapping $R(\varphi) = (\varphi'', 2^{m+q-1})$ is computable in polynomial time with respect to c and m, it is a reduction from Majority-SAT to Threshold-3SAT. □

3 Solving Threshold-3SAT

In order to solve Threshold-3SAT we design a polynomial time, deterministic Turing machine F (for "family") such that, for each n of the form $8\binom{m}{3} + m$, the output of $F(1^n)$ is a P system Π_n that solves the problem for all inputs of length n.

The input provided to Π_n is computed by another polynomial time Turing machine E (for "encoding") that, given an m-variable 3CNF formula as described in the previous section and an integer k, outputs the following set of objects:

$$\begin{aligned} E(\varphi, k) = \{&\mathrm{C}_i : \text{the } i\text{-th clause does } \textit{not} \text{ appear in } \varphi, \text{ for } 1 \leq i \leq 8\tbinom{m}{3}\} \cup \\ \{&\mathrm{K}_i : \text{the } i\text{-th bit of } k \text{ (counting from 0) is 1, for } 1 \leq i \leq m-1\} \end{aligned}$$

Example 2. The formula φ of Example 1, together with the integer $k = 12$, are encoded as $E(\varphi, k) = \{\mathrm{C}_i : 1 \leq i \leq 32 \text{ and } i \notin \{2, 8, 20, 31\}\} \cup \{\mathrm{K}_2, \mathrm{K}_3\}$.

The initial configuration of Π_n, input multiset excluded, is the following one:

$$\mathcal{C}_0 = \left[\left[\mathrm{I}_{n-m}\right]^0_{\mathrm{E}} \left[\ \right]^0_{\mathrm{K}0} \cdots \left[\ \right]^0_{\mathrm{K}m-1} \mathrm{O}_{t+1}\ \mathrm{NO}_{t+3}\right]^0_{\mathrm{IN}}$$

where $t = 4n - 3m + 4$. The multiset encoding $E(\varphi, k)$ is placed inside the input membrane in, and then the computation proceeds according to the following five phases:

1. Initialise the contents of the membranes.
2. Generate all possible assignments for φ.
3. Check if each assignment satisfies the input formula φ.
4. Count the number of assignments, testing whether it is larger than k.
5. Output the correct answer.

The fourth phase, first suggested by Alhazov et al. [1], differentiates our solution from the standard algorithm schema, common in membrane computing, for solving **NP**-complete problems.[3]

Phase 1 (Initialise). In the first computation steps, the objects C_i, corresponding to the clauses that do not appear in the input formula φ, are moved to membrane E using the communication rules

$$\mathrm{C}_i\ [\ ]^0_{\mathrm{E}} \to [\mathrm{C}_i]^0_{\mathrm{E}} \qquad \text{for } 1 \leq i \leq n - m. \tag{R_1}$$

This takes a number of steps at most equal to $n-m$ (i.e., to the maximum number of clauses in φ). In the mean time, the object I_{n-m} has its subscript decreased by one for $n-m-1$ computation steps, and is finally replaced during the $(n-m)$-th step, using the rules

[3] Indeed, by eliminating the fourth phase (or, equivalently, by choosing $k = 0$) we obtain essentially a uniform version of the original solution to SAT described by Zandron et al. [12].

$$[\mathrm{I}_i \to \mathrm{I}_{i-1}]^0_{\mathrm{E}} \qquad \text{for } 1 \le i \le n-m. \tag{R_2}$$

$$[\mathrm{I}_0 \to \mathrm{X}_1 \cdots \mathrm{X}_m \mathrm{W}_m]^0_{\mathrm{E}} \tag{R_3}$$

Hence, after $n-m$ computation steps, membrane E contains C_i for each missing clause, and the variable-objects $\mathrm{X}_1, \ldots, \mathrm{X}_m$.

At the same time, the objects K_i are first moved to their respective membranes in the first time step, making them positively charged

$$\mathrm{K}_i\, [\,]^0_{\mathrm{K}_i} \to [\mathrm{K}_i]^+_{\mathrm{K}_i} \qquad \text{for } 0 \le i \le m-1 \tag{R_4}$$

then each K_i divides its membrane i times:

$$[\mathrm{K}_i]^+_{\mathrm{K}_j} \to [\mathrm{K}_{i-1}]^+_{\mathrm{K}_j}\, [\mathrm{K}_{i-1}]^+_{\mathrm{K}_j} \qquad \text{for } 0 \le j \le m-1 \text{ and } 1 \le i \le j. \tag{R_5}$$

After at most m steps (the largest possible subscript is $m-1$), there are exactly k positively charged membranes among those having label $\mathrm{K}_0, \ldots, \mathrm{K}_{m-1}$.

The total duration of Phase 1 is $n-m$ steps.

Phase 2 (Generate). The variable-objects $\mathrm{X}_1, \ldots, \mathrm{X}_m$ are used to generate all the truth assignment inside multiple copies of membrane E. This is accomplished by using the division rules

$$[\mathrm{X}_i]^0_{\mathrm{E}} \to [\mathrm{T}_i]^0_{\mathrm{E}}\, [\mathrm{F}_i]^0_{\mathrm{E}} \qquad \text{for } 1 \le i \le m. \tag{R_6}$$

After m steps, we have 2^m copies of membrane E, each one containing a different truth assignment to the variables $x_1, \ldots, x_m$ of φ: the occurrence of T_i (resp., F_i) indicates that x_i is set to true (resp., false) in that particular assignment.

Simultaneously, the subscript of object W_m (standing for "wait m steps") is decreased by one each step:

$$[\mathrm{W}_i \to \mathrm{W}_{i-1}]^0_{\mathrm{E}} \qquad \text{for } 1 \le i \le m. \tag{R_7}$$

When the counter reaches 0, the objects W_0 are sent out from each copy of membrane E while changing its charge according to the following rule:

$$[\mathrm{W}_0]^0_{\mathrm{E}} \to [\,]^+_{\mathrm{E}}\, \mathrm{W}_0. \tag{R_8}$$

When membrane E is positively charged, the objects T_i and F_i are replaced by the set of clause-objects corresponding to all the clauses satisfied by that particular value of variable x_i (whether they are actually part of formula φ or not):

$$[\mathrm{T}_i \to \mathrm{C}_{i_1} \cdots \mathrm{C}_{i_\ell}]^+_{\mathrm{E}} \quad \text{for } 1 \le i \le m, \text{ where clause } i_j \text{ contains literal } x_i \tag{R_9}$$

$$[\mathrm{F}_i \to \mathrm{C}_{i_1} \cdots \mathrm{C}_{i_\ell}]^+_{\mathrm{E}} \quad \text{for } 1 \le i \le m, \text{ where clause } i_j \text{ contains literal } \bar{x}_i. \tag{R_{10}}$$

Notice that $\ell = 4\binom{m-1}{2} = 2m^2 - 6m + 4$, as this is the number of clauses over m variables where a particular literal occurs.

At the same time, each copy of W_0 is brought back as S_0 to a copy of membrane E by using the following rule in a maximally parallel way:

$$\mathrm{W}_0\, [\,]^+_{\mathrm{E}} \to [\mathrm{S}_0]^+_{\mathrm{E}}. \tag{R_{11}}$$

The total duration of Phase 2 is $m+2$ steps.

Phase 3 (Check). The occurrence of S_i inside a copy of membrane E denotes the fact that the first i clauses (according to the enumeration described above) have been found to be satisfied by the assignment corresponding to that membrane. We assume that the clauses which do *not* appear in φ are satisfied by default; indeed, this is precisely the reason why the corresponding C_i objects were placed inside membrane E in Phase 1.

When membrane E is positively charged, the object C_1 is sent out from E (as the "junk" object #), changing the charge to negative:

$$[\mathrm{C}_1]^+_{\mathrm{E}} \to [\,]^-_{\mathrm{E}}\ \#. \tag{R_{12}}$$

When E is negative, object S_i is sent out; at the same time, the objects C_i, for $i \geq 2$, are temporarily "primed", and all remaining copies of C_1 are discarded:

$$[\mathrm{S}_i]^-_{\mathrm{E}} \to [\,]^-_{\mathrm{E}}\ \mathrm{S}_i \qquad \text{for } 0 \leq i \leq n-m-1 \tag{R_{13}}$$

$$[\mathrm{C}_i \to \mathrm{C}'_i]^-_{\mathrm{E}} \qquad \text{for } 2 \leq i \leq n-m \tag{R_{14}}$$

$$[\mathrm{C}_1 \to \#]^-_{\mathrm{E}}. \tag{R_{15}}$$

In the next step, the objects C'_i become C_{i-1}; this way, during this phase we only need to check for the presence of object C_1 for $n-m$ times.

$$[\mathrm{C}'_i \to \mathrm{C}_{i-1}]^-_{\mathrm{E}} \qquad \text{for } 2 \leq i \leq n-m. \tag{R_{16}}$$

At the same time, the object S_i is brought back in (if $i < n-m$), its subscript incremented by one, while changing the charge of E to positive in order to resume the checking of clauses:

$$\mathrm{S}_i\ [\,]^-_{\mathrm{E}} \to [\mathrm{S}_{i+1}]^+_{\mathrm{E}} \qquad \text{for } 0 \leq i \leq n-m-1. \tag{R_{17}}$$

If S_{n-m} is finally found inside E, it is sent out to signal that the formula is fully satisfied under that particular assignment:

$$[\mathrm{S}_{n-m}]^+_{\mathrm{E}} \to [\,]^0_{\mathrm{E}}\ \mathrm{S}_{n-m}. \tag{R_{18}}$$

Hence, after the $3n-3m+1$ steps of Phase 3, the outermost membrane IN contains a copy of S_{n-m} for each assignment that satisfies φ.

Phase 4 (Count). In the next step, k copies of S_{n-m} (or all of them, if less than k exist) are "deleted" from membrane IN by sending them into any of the membranes having label $\mathrm{K}_0, \ldots, \mathrm{K}_{m-1}$; these membranes are set to negative in the process, to avoid absorbing multiple objects:

$$\mathrm{S}_{n-m}\ [\,]^+_{\mathrm{K}_i} \to [\#]^-_{\mathrm{K}_i} \qquad \text{for } 0 \leq i \leq m-1. \tag{R_{19}}$$

Recall that the number of positively charged membrane K_i is exactly k. Hence, after this single step there are one or more copies of S_{n-m} left inside membrane IN if and only if the number of satisfying assignments of φ was greater than k.

Phase 5 (Output). The objects o_t and NO_{t+2}, initially located inside membrane IN, work as counters during Phases 1–4 (whose total duration is precisely $t = 4n - 3m + 4$ steps) according to the following rules:

$$[\mathrm{o}_i \to \mathrm{o}_{i-1}]^0_{\mathrm{IN}} \qquad \text{for } 1 \le i \le t+1 \qquad (R_{20})$$

$$[\mathrm{NO}_i \to \mathrm{NO}_{i-1}]^0_{\mathrm{IN}} \qquad \text{for } 2 \le i \le t+3. \qquad (R_{21})$$

When the subscript of o_i reaches 0, Phase 5 begins and o_0 is sent out, thus "opening" membrane IN for output by setting its charge to positive:

$$[\mathrm{o}_0]^0_{\mathrm{IN}} \to [\,]^+_{\mathrm{IN}} \#. \qquad (R_{22})$$

If any object s_{n-m} is found inside IN, it is sent out as YES in the next step, changing the charge to negative:

$$[\mathrm{s}_{n-m}]^+_{\mathrm{IN}} \to [\,]^-_{\mathrm{IN}}\ \mathrm{YES}. \qquad (R_{23})$$

Otherwise, membrane IN remains positive, and the object NO_0, produced by the rule

$$[\mathrm{NO}_1 \to \mathrm{NO}_0]^+_{\mathrm{IN}} \qquad (R_{24})$$

is sent out as NO in the following step:

$$[\mathrm{NO}_0]^+_{\mathrm{IN}} \to [\,]^-_{\mathrm{IN}}\ \mathrm{NO}. \qquad (R_{25})$$

The duration of Phase 5 is either 2 or 3 steps, depending on whether the number of assignments satisfying φ is greater than k or not.

This algorithm allows us to solve the Threshold-3SAT problem in linear time $O(n+m) = O(n)$.

Theorem 1. Threshold-3SAT $\in \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$.

Proof. Let $\boldsymbol{\Pi}$ be the family of P systems described above. We show that $\boldsymbol{\Pi}$ can be constructed in polynomial time by two Turing machines F and E.

The machine F, on input 1^n (where $n = |(\varphi, k)|$) first computes the unique positive root of the polynomial

$$p(m) = 8\tbinom{m}{3} + m - n$$

thus establishing the number of variables. This can be done in polynomial time with respect to n simply by trying all integers up to n.

Then F outputs the initial configuration $\mathcal{C}_0$ of Π_n, which can be easily computed in polynomial time from n and m. Finally, the set of rules $R_1 \cup \cdots \cup R_{25}$ is output. Each of the sets R_i can be computed in polynomial time; the most complicated ones are R_9 and R_{10}, which require enumerating the clauses using the algorithm of Figure 1.

The P system Π_n itself, on input $E(\varphi, k)$, only requires $O(n)$ time (and $O(n2^m)$ space) to output YES or NO, without using any nonelementary division or dissolution rules; this establishes that the problem is in $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$.

If the machines F and E receive a malformed input, i.e., any input having length $n \neq 8\binom{m}{3} + m$ for all $m \leq n$, then F produces a fixed P system that sends out the NO object immediately (while E produces an empty multiset). □

4 Solving the other PP problems

Being able to solve one **PP**-complete problem implies $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$ if the uniformity condition is defined as in [6], as closure under polynomial-time reductions is immediate. However, our uniformity condition is possibly weaker, as the P system associated with each input only depends on its size and not on the specific input itself, and the class $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$ defined this way is currently not known to be closed under polynomial-time reductions. Hence, to prove the **PP** inclusion we operate as follows.

Theorem 2. $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$.

Proof. Let $L \in \mathbf{PP}$, and let R be a Turing machine reducing L to the problem THRESHOLD-3SAT in polynomial time $p(n)$, where n is the length of the instance of L. We describe two polynomial-time Turing machines F' and E' constructing a family of P systems $\boldsymbol{\Pi}'$, also running in polynomial time, such that $L(\boldsymbol{\Pi}') = L$.

The machine F', on input 1^n (where $n = |x|$), constructs a P system able to solve the largest THRESHOLD-3SAT formula that might be produced as the output of R; if the actual output of R is smaller than that, we can pad it to the correct length by adding enough zeroes. Let f be defined as follows:

$$f(n) = \min\left\{n' : n' \geq n \text{ and } n' = 8\tbinom{m'}{3} + m' \text{ for some } m'\right\}$$

that is, $f(n)$ is the smallest integer of the form $8\binom{m'}{3} + m'$ greater than or equal to n. Then, F' behaves as follows:

$$F'(1^n) = F\left(1^{f(p(n))}\right) = \Pi_{f(p(n))}.$$

Since R runs in time $p(n)$, the P system $\Pi_{f(p(n))}$ is large enough to receive as input any formula φ obtained via the reduction R, as $|R(x)| = |(\varphi, k)| \leq p(|x|)$, as long as it is padded to length $f(p(n))$ as described above.

Notice that the value $f(n)$ can be obtained in polynomial time with respect to n by simply computing $8\binom{m'}{3} + m'$ for all integers m' until n is reached or exceeded; furthermore, $f(n)$ itself is at most polynomial in n (e.g., a trivial upper bound is $8\binom{n}{3} + n$).

The encoding machine E', on input x, produces an output formula encoding φ', obtained from $(\varphi, k) = R(x)$ as follows:

$$\varphi' = \varphi \cdot 0^{\ell} \qquad \text{where } \ell = f(p(n)) - |\varphi|.$$

Recall from Section 2 that $\varphi \cdot 0^{\ell}$ is indeed a valid encoding of a formula, having exactly the same clauses of φ *but over* m' *variables instead of the original* m (where m' satisfies $f(p(n)) = 8\binom{m'}{3} + m'$). The number of required assignments k has to be adjusted accordingly: every assignment of the original formula φ corresponds to $2^{m'-m}$ assignments of φ' (obtained by extending it with arbitrary values to the new variables) that satisfy it iff the original assignment satisfies φ, since the new $m' - m$ variables do not actually appear in φ'. Hence, we define $k' = 2^{m'-m} \cdot k$.

Summarising, the machine E' behaves as follows:

$$E'(x) = E\big(\varphi \cdot 0^{\ell}, 2^{m'-m}k\big) \qquad \text{where } (\varphi, k) = R(x).$$

Since $(\varphi', k') \in$ Threshold-3SAT iff $(\varphi, k) \in$ Threshold-3SAT by construction, and the latter is equivalent to $x \in L$ by reduction, we obtain $L \in \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$. But L was an arbitrary **PP** language: hence the inclusion $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$ holds as required. □

5 Conclusions

Uniform families of P systems with active membranes without nonelementary division and dissolution rules have been proved to be able to solve all **PP** problems in polynomial time; this property holds even when the uniformity condition is the same as that used for traditional families of circuits. The current bounds on the computing power of these P systems, in terms of complexity classes for Turing machines, are thus

$$\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})} \subseteq \mathbf{PSPACE} = \mathbf{PMC}_{\mathcal{AM}},$$

where neither inclusion is known to be proper. Further improvements on the **PP** lower bound are expected, as it is plausible that P systems like those of the family $\boldsymbol{\Pi}$ solving Threshold-3SAT described in this paper can be used as "modules" in larger P systems, thus providing a way to simulate an oracle for a **PP**-complete problem. Furthermore, it is still possible that $\mathbf{PMC}_{\mathcal{AM}(-\mathrm{n},-\mathrm{d})}$ actually coincides with **PSPACE**, thus showing that nonelementary membrane division (and possibly dissolution) do not increase the efficiency of P systems with active membranes.

References

1. Alhazov, A., Burtseva, L., Cojocaru, S., Rogozhin, Y.: Solving PP-complete and #P-complete problems by P systems with active membranes. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing 9th International Workshop, WMC 2008, Lecture Notes in Computer Science, vol. 5931, pp. 108–117. Springer (2009)

2. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. Fundamenta Informaticae 58(2), 67–77 (2003)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. (1979)
4. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
5. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
6. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–284 (2003)
7. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with elementary active membranes: Beyond NP and coNP. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 11th International Conference, CMC 2010, Lecture Notes in Computer Science, vol. 6501, pp. 338–347. Springer (2011)
8. Porreca, A.E., Murphy, N.: First steps towards linking membrane depth and the polynomial hierarchy. In: Martínez-del-Amor, M.A., Păun, G., Pérez-Hurtado, I., Riscos-Núñez, A. (eds.) Eight Brainstorming Week on Membrane Computing, RGNC Reports, vol. 1/2010, pp. 255–266. Fénix Editora (2010)
9. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
10. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
11. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences 73(1), 137–152 (2007)
12. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, pp. 289–301. Springer (2001)