

P systems simulating oracle computations

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

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

Abstract We show how existing P systems with active membranes can be used as modules inside a larger P system; this allows us to simulate subroutines or oracles. As an application of this construction, which is (in principle) quite general, we provide a new, improved lower bound to the complexity class $\mathbf{PMC}_{\mathcal{AM}(-d,-n)}$ of problems solved by polynomial-time P systems with (restricted) elementary active membranes: this class is proved to contain $\mathbf{P}^{\mathbf{PP}}$ and hence, by Toda's theorem, the whole polynomial hierarchy.

1 Introduction

P systems with active membranes [5] are known to be able to solve problems beyond \mathbf{P} in polynomial time, by trading space for time. Using membrane division rules, an exponential number of membranes is created in polynomial time, which then compute in parallel, for instance exploring the solution space of an \mathbf{NP} problem.

The exact computing power of P systems with active membranes depends on which membrane division rules are available. First of all, it can be proved (Milano theorem [10]) that *some* form of division is actually necessary in order to go beyond \mathbf{P} , otherwise the P system can be simulated sequentially in polynomial time. When the systems are allowed to divide membranes containing further membranes (called *nonelementary membranes*), thus replicating the whole structure in each copy of the dividing membrane, even \mathbf{PSPACE} -complete problems become solvable in polynomial time. Usually, this involves constructing a membrane structure representing a full binary tree with exponentially many nodes [6,1].

When only membranes not containing further membranes (called *elementary*) can divide, the situation becomes more interesting: the systems are still able to create exponentially many membranes, but they all appear as leaves of the tree, without the possibility of creating more complicated tree structures. This is still sufficient not only to solve \mathbf{NP} -complete problems [10], but also to count the number of “positive” candidate solutions in polynomial time; thus, the power of the whole complexity class \mathbf{PP} is captured [4].

In this paper we improve this lower bound by showing how P systems known to solve some problem (in polynomial time) can be used to simulate oracle quer-

ies, by embedding them into P systems simulating polynomial-time deterministic Turing machines. The idea is to provide the input multiset to the embedded P systems not as part of their initial configuration, but only when the “outer” P systems require an oracle answer to continue its computation. Whenever this procedure can be carried out for a family of P systems deciding a language L , then the whole class \mathbf{P}^L of languages decidable in polynomial time by Turing machines with an oracle for L can also be decided efficiently by P systems. We argue that this applies to most existing solutions described in the literature, though the formal details may vary.

As a concrete application, we choose L to be the problem of checking whether the number of assignments satisfying a Boolean formula is greater than a given threshold. This problem is known to be \mathbf{PP} -complete [4], and as a consequence the class $\mathbf{P}^{\mathbf{PP}}$ turns out to be solvable in polynomial time by P systems, without requiring nonelementary division or dissolution rules.

2 Definitions

P systems with active membranes are defined as follows.

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

- Γ is an alphabet, i.e., a finite non-empty set of symbols of symbols, usually called objects;
- Λ is a finite set of labels for the membranes;
- μ is a membrane structure (i.e., a rooted unordered tree, usually represented by nested brackets) consisting of d membranes enumerated by $1, \dots, d$; furthermore, each membrane is labeled by an element of Λ , not necessarily in a one-to-one way;
- w_1, \dots, 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 $[a \rightarrow w]_h^\alpha$
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).
- *Send-in communication rules*, of the form $a []_h^\alpha \rightarrow [b]_h^\beta$
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 $[a]_h^\alpha \rightarrow []_h^\beta b$
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 .
- *Dissolution rules*, of the form $[a]_h^\alpha \rightarrow b$
They can be applied to a membrane labeled by h , having charge α 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 $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$
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.
- *Nonelementary division rules*, of the form

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

They can be applied to a membrane labeled by h , having charge α , containing the positively charged membranes h_1, \dots, h_k , the negatively charged membranes h_{k+1}, \dots, h_n , and possibly some neutral membranes. The membrane h is divided into two copies having charge β and γ , respectively; the positive children are placed inside the former, their charge changed to δ , while the negative ones are placed inside the latter, their charges changed to ϵ . 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 non-deterministic 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 the application proceeds 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.

The precise variant of P systems we use in this paper does not use dissolution or nonelementary division rules.

Definition 2. A P system with restricted elementary active membranes is a P system with active membranes where only object evolution, send-in, send-out, and elementary division rules are used.

A *halting computation* of the P system Π is a finite sequence of configurations $\mathcal{C} = (C_0, \dots, C_k)$, where C_0 is the initial configuration, every C_{i+1} is reachable by C_i via a single computation step, and no rules can be applied anymore in C_k . A *non-halting computation* $\mathcal{C} = (C_i : i \in \mathbb{N})$ 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. All P systems in this paper are confluent.

In order to solve decision problems (i.e., decide languages), we use *families* of recogniser P systems $\Pi = \{\Pi_x : x \in \Sigma^*\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^*$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for each input length [2].

Definition 3. A family of P systems $\Pi = \{\Pi_x : x \in \Sigma^*\}$ 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 , which is common for all inputs of length n , with a distinguished input membrane.

- 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.¹

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 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 proportional to their number; see also [2] for further details on the encoding of P systems.

Finally, we describe how time complexity for families of recogniser P systems is measured, and their complexity classes [3].

Definition 4. A uniform family of P systems $\Pi = \{\Pi_x : x \in \Sigma^*\}$ is said to decide the language $L \subseteq \Sigma^*$ (in symbols $L(\Pi) = L$) in time $f: \mathbb{N} \rightarrow \mathbb{N}$ iff, for each $x \in \Sigma^*$,

- 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.

Definition 5. The complexity class $\mathbf{PMC}_{\mathcal{AM}(-d, -n)}$ consists of all the languages L decidable in polynomial time by a uniform family of P systems with restricted elementary active membranes.

3 Simulating Turing machines

Let M be a deterministic Turing machine working in polynomial time $p(n)$. We design a uniform family of recogniser P systems Π_M simulating M on inputs of different lengths. This construction is a variant of the one presented in [9].

A Turing machine F_M , on input 1^n , produces as output a P system $\Pi_{M,n}$ having the following initial configuration:

$$[H []_0^0 []_1^0 []_2^0 \cdots []_{p(n)}^0]_s^0.$$

Each of the membranes having numerical label $0, 1, 2, \dots, p(n)$ represents one of the tape cells of M .² The symbol contained in that cell is encoded by the charge of the associated membrane: a neutral charge represents a blank, a negative charge a 0, and a positive one a 1. In the following, we denote by lower case Greek letters α, β both the tape symbols and the corresponding charges.

¹ Notice that this definition of uniformity is (possibly) weaker than the other one commonly used in membrane computing [3], where the Turing machine F maps each input x to a P system $\Pi_{s(x)}$, where $s: \Sigma^* \rightarrow \mathbb{N}$ is a measure of the size of the input; in our case, $s(x)$ is always $|x|$.

² Recall that M cannot visit more than $p(n) + 1$ tape cells on an input of length n .

During the first computation step, the object H is replaced by H_{0,q_0} using the following evolution rule:

$$[H \rightarrow H_{0,q_0}]_S^0.$$

(The reason for having a different object rather than directly H_{0,q_0} in the initial configuration will be clarified below.) The object $H_{i,q}$ simulates the tape head of M : its subscripts denote the current position i on the tape, from 0 to $p(n)$, and the current state q of M . Their initial values are 0 and q_0 , i.e., the leftmost position on the tape and the initial state of M .

The transition function $\delta: Q \times \Sigma \rightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ of M (where Q is the set of states, Γ the tape alphabet and \leftarrow, \rightarrow denote movement to the left and right respectively) is implemented by using communication rules, that are replicated for each possible membrane corresponding to a tape cell. A transition $\delta(q_1, \alpha) = (q_2, \beta, \leftarrow)$ corresponds to the following rules:

$$\left. \begin{array}{l} H_{i,q_1} []_i^\alpha \rightarrow [H_{i-1,q_2}]_i^\beta \\ [H_{i-1,q_2}]_i^\beta \rightarrow []_i^\beta H_{i-1,q_2} \end{array} \right\} \text{for } 0 < i \leq p(n).$$

Notice how the first set of rules changes the charge of membrane i (representing the symbol in the i -th cell) and the two subscripts of H_{i,q_1} (representing head position and state) according to the transition function δ . The second set of rules is only used to move back the object H_{i,q_2} to its starting position, i.e., inside the outermost membrane S . Transitions such as $\delta(q_1, \alpha) = (q_2, \beta, \rightarrow)$, that move the head to the right, are analogous:

$$\left. \begin{array}{l} H_{i,q_1} []_i^\alpha \rightarrow [H_{i+1,q_2}]_i^\beta \\ [H_{i+1,q_2}]_i^\beta \rightarrow []_i^\beta H_{i+1,q_2} \end{array} \right\} \text{for } 0 \leq i < p(n).$$

Assuming without loss of generality that the transition function δ is undefined on the accepting and rejecting states q_{YES} and q_{NO} , two further sets of rules are needed in order for $\Pi_{M,n}$ to give the same result as M on its input:

$$\left. \begin{array}{l} [H_{i,q_{\text{YES}}}]_S^0 \rightarrow []_S^0 \text{ YES} \\ [H_{i,q_{\text{NO}}}]_S^0 \rightarrow []_S^0 \text{ NO} \end{array} \right\} \text{for } 0 \leq i \leq p(n).$$

The size of the membrane structure, the number of different objects and the rules needed to simulate M are bounded by $O(p(n))$. Both the membrane structure and the rules can be computed by the machine F in polynomial time with respect to that amount.

The input for $\Pi_{M,n}$, encoding the actual string x on which M is to be run, is computed by another Turing machine E_M simply by subscripting the symbols in x by their position in the string. For instance,

$$E_M(11001) = 1_0 1_1 0_2 0_3 1_4.$$

This encoding can be clearly computed in polynomial time with respect to the length of x .

The encoding multiset $E_M(x)$ is placed inside membrane s of $\Pi_{M,n}$, and it is used to initialise the charge of the first n membranes corresponding to tape cells, according to the following rules:

$$\left. \begin{array}{l} 1_i []_i^0 \rightarrow [\#]_i^+ \\ 0_i []_i^0 \rightarrow [\#]_i^- \end{array} \right\} \text{for } 0 \leq i < n,$$

where $\#$ is a “junk” object that has no role for the rest of the computation. These rules are applied in parallel during the first computation step, and avoiding interference with them is the reason why the object $H_{i,q}$ is also introduced only in that step, after which the real simulation begins.

The family Π_M is thus constructed by F_M and E_M in polynomial time, and simulates M on inputs of various lengths in linear time with respect to $p(n)$.

4 Simulating oracle machines

We now show how polynomial-time Turing machines with an oracle for a language L can also be simulated, assuming that L itself can be recognised by a uniform family of P systems Π_L . While the construction we are going to describe is, in principle, quite general, the technical details depend on the specific family Π_L and, in particular, on the encoding chosen for the instances of L . Hence, as a concrete example, we assume that L is the problem THRESHOLD-3SAT, defined as follows.

Problem 1 (THRESHOLD-3SAT). Given a 3CNF 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?

This problem, which is **PP**-hard, has been recently proved [4] to be solvable in polynomial time by a uniform family of P systems with restricted elementary active membranes; as a consequence, the result $\mathbf{PP} \subseteq \mathbf{PMC}_{\mathcal{AM}(-d,-n)}$ holds. By selecting THRESHOLD-3SAT as the oracle language we can prove a better lower bound to this complexity class.

Theorem 1. $\mathbf{P}^{\mathbf{PP}} \subseteq \mathbf{PMC}_{\mathcal{AM}(-d,-n)}$.

The instances of THRESHOLD-3SAT are Boolean formulae in ternary conjunctive normal form, that is, conjunctions of clauses consisting of a disjunction of three literals (i.e., optionally negated variables) where each variable occurs at most once in each clause. There are $8\binom{m}{3}$ possible clauses over m variables (there are $\binom{m}{3}$ subsets of 3 variables out of m , and 2^3 ways to negate them); hence a Boolean formula φ with that number of variables can be encoded as a binary string of length $8\binom{m}{3}$, where the i -th bit is 1 if the i -th clause (under some fixed ordering) actually appears in φ , and 0 otherwise. Since the integer

k can be encoded using further m bits, the total length of the encoding of an instance of THRESHOLD-3SAT is $8\binom{m}{3} + m$.

Now let M be a polynomial-time Turing machine with access to an oracle for THRESHOLD-3SAT. We make a number of assumptions about the functioning of M in order to simplify the simulation. (None of these assumptions causes a loss of generality, since we only care about polynomial running time and not about the *precise* polynomial.) First of all, we assume that M has a *single tape*, which is used as input space, scratch space, *and* as a place to write the oracle queries. The oracle querying procedure works as follows: first M writes down the query string y on the tape, then it moves the head to the first symbol of y , and finally enters the query state $q_?$. In the next computation step, the state of M will be changed, in order to reflect the answer of the oracle to the question “is $y \in L?$ ”, to q_Y or q_N , and the tape head will be repositioned to the first tape cell. A further assumption we are allowed to make is that *all query strings have the same length* $\ell = 8\binom{m}{3} + m$, where ℓ is the largest integer of that form to be less than or equal to $p(n)$; this is due to the fact that a pair (φ_1, k_1) , where φ_1 is a formula over m_1 variables, is a positive instance of THRESHOLD-3SAT if and only if (φ_2, k_2) is, where φ_2 is a formula having the same clauses of φ_1 but *over* $m_2 \geq m_1$ variables (i.e., φ_2 is a padded version of φ_1) and $k_2 = 2^{m_2 - m_1} \times k_1$ (i.e., the number of required assignments is increased in order to reflect the fact that the “new” variables $x_{m_1+1}, \dots, x_{m_2}$ do not actually appear in φ_2 , hence their truth values do not change the overall evaluation of the formula). By choosing an appropriate encoding, each instance can be brought to length ℓ just by padding φ_1 and k_1 with enough zeroes; see the original paper [4] for the details.

Let $\Pi_\ell \in \mathbf{II}_L$ be the P system associated to the THRESHOLD-3SAT instances of length $\ell = 8\binom{m}{3} + m$. This P system has the following initial configuration [4]:

$$[[I_{\ell-m}]_E^0 []_{K_0}^0 \cdots []_{K_{m-1}}^0 O_{t+1} NO_{t+3}]_{IN}^0$$

where $t = 4\ell - 3m + 4$. The input for Π_ℓ is placed inside membrane IN, and it consists of an encoding of (φ, k) described as follows:

$$E_L(\varphi, k) = \{C_i : \text{the } i\text{-th clause does not appear in } \varphi, \text{ for } 1 \leq i \leq 8\binom{m}{3}\} \cup \\ \{K_i : \text{the } i\text{-th bit of } k \text{ (counting from 0) is 1, for } 0 \leq i \leq m-1\}.$$

We construct the P system $\Pi_{M,n}$ simulating M on inputs of length n as follows:

$$[H []_0^0 []_1^0 []_2^0 \cdots []_{2p(n)}^0 [II_\ell]_Q^0 [II_\ell]_Q^0 \cdots [II_\ell]_Q^0 []_\Lambda^0]_S^0.$$

This initial configuration contains $p(n)$ copies of Π_ℓ , each one enclosed by a further membrane having label Q. The number $p(n)$ is chosen as it is the maximum number of queries that M can perform³. However, the initial configuration of the

³ Although, for the sake of a simpler exposition, this configuration contains several membranes having the same label, the labels can be made unique by subscripting them and replicating the rules accordingly (this does not affect the construction time by more than a polynomial amount).

embedded P systems Π_ℓ is changed by erasing the initial objects and keeping only the membrane structure and the rules; this is required in order for these P systems to avoid starting their computation immediately, as their input will be provided later during the computation of $\Pi_{M,n}$. Membrane A will be used to store (in its charge) the result of an oracle query.

Notice how the number of simulated tape cells has been increased to $2p(n)+1$: this is due to the fact that we require all query strings to be of length ℓ , hence we need to leave extra space on the tape for padding them to this length.

The simulation of M by $\Pi_{M,n}$ works exactly as in Section 3 as long as M does not enter its query state. We shall describe how oracle queries are simulated, first in an informal way, then by giving all the technical details.

4.1 Informal description of the simulation of oracle queries

This is an overview of the oracle query simulation:

1. The tape positions corresponding to the query string are inspected, and the multiset of objects w encoding it is produced.
2. At the same time, one of unused copies of the embedded P systems Π_ℓ is chosen; it will be used to simulate the current query.
3. The objects in w are moved to their initial position inside that copy of Π_ℓ .
4. The objects missing from the initial configuration of Π_ℓ are created and moved to the correct membranes.
5. Now the embedded P system performs its computation as in [4], and produces the answer to the query.
6. Finally, the answer is communicated to the object simulating the tape head of M ; it switches to the corresponding state and resumes simulating the Turing machine.

In more details, the procedure works as follows. When M enters the query state $q?$, the object $H_{i,q?}$ is produced and moved to membrane s . According to the convention described above, the query string y (necessarily of length ℓ) is now located on tape cells $i, \dots, i + \ell - 1$.

First, the object $H_{i,q?}$ is rewritten into the following multiset:

$$H'_{i,q?} C'_{1,i} C'_{2,i+1} \cdots C'_{\ell-m,i+\ell-m-1} K'_{0,i+\ell-m} K'_{1,i+\ell-m+1} \cdots K'_{m-1,i+\ell-1}$$

The objects $C_{j,i}$ and $K_{j,i}$ represent *potential* input objects C_j and K_j for an instance of Π_ℓ , which will be actually produced depending on the particular query string y . The procedure is the following one: each object $C_{j,i}$ and $K_{j,i}$ enters the corresponding membrane j , and is either rewritten into a “junk” object $\#$, or sent out as C_j or K_j depending on the symbol contained in the tape cell (thus simulating the encoding machine E_L described above).

In the mean time, the object $H'_{i,q?}$ nondeterministically selects one of the neutral membranes having label Q and “opens” it by setting its charge to positive. The P system Π_ℓ contained inside that membrane will be used to answer the

current query. The object is moved back as W_ℓ to membrane S, where it “waits” for ℓ steps by decreasing its subscript.

While the object w waits, the objects C_j and K_j that were actually produced (there are at most ℓ of them) move through the positive membrane Q and inside the membrane S of the selected copy of Π_ℓ , thus reaching their initial position. At that point, the subscript of w will have reached 0. The object w_0 then enters the active instance of Π_ℓ , and inside membrane S it produces by evolution the objects $I_{\ell-m+1}$, O_{t+2} , and NO_{t+4} , and is itself rewritten into w'_0 . Notice how the subscripts of I , O , and NO are incremented by one: this is done in order to give the opportunity to $I_{\ell-m+1}$ to move (as $I_{\ell-m}$) to its initial position inside membrane E , and also to w'_0 to exit Π_ℓ and move back to membrane Q .

While w'_0 moves to membrane S , the chosen embedded P system Π_ℓ finally starts computing as if it were in stand-alone form, as in [4]. This computation requires a polynomial amount of time, after which either the object YES or NO will be sent out to the surrounding membrane Q . That result object then exits Q , setting its charge to negative (thus signalling that the enclosed P system Π_ℓ has been used, and cannot be reused again) and moving to membrane A . When entering it, the charge is set to positive (if the result object is YES) or negative (if it is NO); the result object is simultaneously rewritten into $\#$.

When membrane A is non-neutral, the object w'_0 can enter it, and be sent out as H_{0,q_v} or H_{0,q_n} depending on the result; the charge of A is also reset to neutral to allow further queries. The simulation now proceeds again as in Section 3, until another oracle query is made or until the computation of M finally terminates.

4.2 Technical details

When the simulated Turing machine enters the query state, an object H_{i,q_τ} appears inside the outermost membrane S of $\Pi_{M,n}$. It is immediately subject to the following evolution rule, which is replicated for $0 \leq i \leq 2p(n)$:

$$[H_{i,q_\tau} \rightarrow H'_{i,q_\tau} C_{1,i} \cdots C'_{\ell-m,i+\ell-m-1} K'_{0,i+\ell-m} \cdots K'_{m-1,i+\ell-1}]_S^0$$

During the next step, the objects $C'_{j,i}$ enter the corresponding membrane labelled by i , where they are either deleted (if that membrane represents a tape cell containing 1, i.e., if the j -th clause occurs in the input formula φ), or sent back out as C_j (if the tape cell contains 0).

$$\left. \begin{array}{l} C'_{j,i} []_i^\alpha \rightarrow [C'_{j,i}]_i^\alpha \\ [C'_{j,i} \rightarrow \#]_i^+ \\ [C'_{j,i}]_i^- \rightarrow []_i^- C_j \end{array} \right\} \text{for } 0 \leq i \leq 2p(n) \text{ and } 1 \leq j \leq \ell - m \text{ and } \alpha \in \{+, -\}$$

The same occurs to the objects $K_{j,i}$, except that they are deleted when the corresponding tape cell contains a 0, according to the encoding performed by

the machine E_L .

$$\left. \begin{array}{l} \mathbf{K}'_{j,i} []_i^\alpha \rightarrow [\mathbf{K}'_{j,i}]_i^\alpha \\ [\mathbf{K}'_{j,i} \rightarrow \#]_i^- \\ [\mathbf{K}'_{j,i}]_i^+ \rightarrow []_i^+ \mathbf{K}_j \end{array} \right\} \text{for } 0 \leq i \leq 2p(n) \text{ and } 0 \leq j \leq m-1 \text{ and } \alpha \in \{+, -\}$$

While the objects encoding the THRESHOLD-3SAT instance are produced, the object $\mathbf{H}'_{i,q?}$ changes the charge of one of the neutral membranes labelled by \mathbf{Q} to positive, and moves back to \mathbf{S} as \mathbf{W}_ℓ .

$$\left. \begin{array}{l} \mathbf{H}'_{i,q?} []_{\mathbf{Q}}^0 \rightarrow [\mathbf{H}_{i,q?}]_{\mathbf{Q}}^+ \\ [\mathbf{H}'_{i,q?}]_{\mathbf{Q}}^+ \rightarrow []_{\mathbf{Q}}^+ \mathbf{W}_\ell \end{array} \right\} \text{for } 0 \leq i \leq 2p(n)$$

The objects \mathbf{C}_j and \mathbf{K}_j (which are at most ℓ in number) are then sequentially moved to the system \mathbf{II}_ℓ corresponding to the positive membrane \mathbf{Q} .

$$\left. \begin{array}{l} \mathbf{C}_j []_{\mathbf{Q}}^+ \rightarrow [\mathbf{C}_j]_{\mathbf{Q}}^+ \\ \mathbf{C}_j []_{\text{IN}}^0 \rightarrow [\mathbf{C}_j]_{\text{IN}}^0 \end{array} \right\} \text{for } 1 \leq j \leq \ell - m$$

$$\left. \begin{array}{l} \mathbf{K}_j []_{\mathbf{Q}}^+ \rightarrow [\mathbf{K}_j]_{\mathbf{Q}}^+ \\ \mathbf{K}_j []_{\text{IN}}^0 \rightarrow [\mathbf{K}_j]_{\text{IN}}^0 \end{array} \right\} \text{for } 0 \leq j \leq m-1$$

The object \mathbf{W}_ℓ “waits” until the last object has entered membrane \mathbf{Q} by decreasing its subscript

$$[\mathbf{W}_i \rightarrow \mathbf{W}_{i-1}]_{\mathbf{S}}^0 \quad \text{for } 1 \leq i \leq \ell$$

then it also enters the selected embedded \mathbf{P} system \mathbf{II}_ℓ in order to initialise its configuration:

$$\begin{array}{l} \mathbf{W}_0 []_{\mathbf{Q}}^+ \rightarrow [\mathbf{W}_0]_{\mathbf{Q}}^+ \\ \mathbf{W}_0 []_{\text{IN}}^0 \rightarrow [\mathbf{W}_0]_{\text{IN}}^0 \\ [\mathbf{W}_0 \rightarrow \mathbf{W}'_0 \mathbf{I}_{\ell-m+1} \mathbf{O}_{t+2} \mathbf{NO}_{t+4}]_{\text{IN}}^0 \end{array}$$

The object \mathbf{W}'_0 is sent back to membrane \mathbf{S} , while the newly created objects reach their actual position and/or subscript inside \mathbf{II}_ℓ , allowing the actual computation of the embedded \mathbf{P} system to start.

$$\begin{array}{l} [\mathbf{W}'_0]_{\text{IN}}^0 \rightarrow []_{\text{IN}}^0 \mathbf{W}'_0 \\ [\mathbf{W}'_0]_{\mathbf{Q}}^+ \rightarrow []_{\mathbf{Q}}^+ \mathbf{W}'_0 \\ \mathbf{I}_{\ell-m+1} []_{\mathbf{E}}^0 \rightarrow [\mathbf{I}_{\ell-m}]_{\mathbf{E}}^0 \\ [\mathbf{O}_{t+2} \rightarrow \mathbf{O}_{t+1}]_{\text{IN}}^0 \\ [\mathbf{NO}_{t+4} \rightarrow \mathbf{NO}_{t+3}]_{\text{IN}}^0 \end{array}$$

When the computation of the active instance of Π_ℓ terminates, a result object is sent out to membrane Q , and it is moved in order to set the charge of membrane A appropriately.

$$\begin{aligned} [\text{YES}]_Q^+ &\rightarrow []_Q^- \text{ YES} \\ [\text{NO}]_Q^+ &\rightarrow []_Q^- \text{ NO} \\ \text{YES } []_A^0 &\rightarrow [\#]_A^+ \\ \text{NO } []_A^0 &\rightarrow [\#]_A^- \end{aligned}$$

Now the object w_0 can “read” the answer from the charge of A , reset it to neutral and be rewritten into the corresponding object encoding the new state of the simulated machine M .

$$\begin{aligned} w_0 []_A^\alpha &\rightarrow [w_0]_A^\alpha \quad \text{for } \alpha \in \{+, -\} \\ [w_0]_A^+ &\rightarrow []_A^0 H_{0,q_Y} \\ [w_0]_A^- &\rightarrow []_A^0 H_{0,q_N} \end{aligned}$$

Now the simulation of M continues as in Section 3.

4.3 Main result

We are finally able to prove the result anticipated at the beginning of this section.

Proof (Theorem 1). The previous discussion shows how any polynomial-time Turing machine equipped with an oracle for THRESHOLD-3SAT can be simulated with a polynomial slowdown. Since THRESHOLD-3SAT is \mathbf{PP} -hard, any other problem in \mathbf{PP} can be efficiently reduced to it by the simulated Turing machine before performing the query. As a consequence, the whole class $\mathbf{P}^{\mathbf{PP}}$ is included in $\mathbf{PMC}_{\mathcal{AM}(-d,-n)}$. \square

By Toda’s theorem ($\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{PP}}$) [8], this result implies that the whole polynomial hierarchy \mathbf{PH} is contained in $\mathbf{PMC}_{\mathcal{AM}(-d,-n)}$, bringing this class closer to the known \mathbf{PSPACE} upper bound [7].

5 Conclusions

We showed how the \mathbf{P} systems of an existing uniform family Π with active membranes, deciding a language L , can be embedded into other \mathbf{P} systems simulating Turing machines in order to answer oracle queries for L . Although the technical details of the construction depend on the specific family Π and on the encoding of the instances of the problem, we are confident that it can be adapted to most families of \mathbf{P} systems with active membranes already described in the literature, and possibly to other variants of \mathbf{P} systems. However, from a formal standpoint, this result is still to be proved.

By using this construction, we also improved the previously known lower bound to the complexity class $\mathbf{PMC}_{\mathcal{AM}(-d,-n)}$ from \mathbf{PP} to $\mathbf{P}^{\mathbf{PP}}$; this also means that the polynomial hierarchy is contained in this class. Giving a more precise characterisation remains an open question; while at this point the equality $\mathbf{PMC}_{\mathcal{AM}(-d,-n)} = \mathbf{PSPACE}$ seems to be the most plausible outcome (that would imply that nonelementary membrane division rules are redundant), we think it is also worth investigating the possibility that the reverse inclusion $\mathbf{PMC}_{\mathcal{AM}(-d,-n)} \subseteq \mathbf{P}^{\mathbf{PP}}$ also holds; this result would be much more interesting, as it doesn't equate a complexity class for P systems with \mathbf{P} or \mathbf{PSPACE} as it usually happens.

Being able to generalise the oracle query simulation to arbitrary families would imply that whenever $L \in \mathbf{PMC}_{\mathcal{AM}(-d,-n)}$, automatically the inclusion $\mathbf{P}^L \subseteq \mathbf{PMC}_{\mathcal{AM}(-d,-n)}$ also holds. Furthermore, if the P systems implementing the oracle could be modified in order to reset to their initial configuration after their computation terminates (a condition we conjecture to be true in the case of the family solving THRESHOLD-3SAT [4]), a more elegant solution could be provided, where only a single embedded P system is needed, as it could be reused for subsequent queries.

References

1. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae* 58(2), 67–77 (2003)
2. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. *Natural Computing* 10(1), 613–632 (2011)
3. 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)
4. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Elementary active membranes have the power of counting. In: Martínez-del-Amor, M.A., Păun, G., Pérez-Hurtado, I., Romero-Campero, F.J., Valencia-Cabrera, L. (eds.) Ninth Brainstorming Week on Membrane Computing, pp. 329–342. No. 1/2011 in RGNC Reports, Fénix Editora (2011), available online at <http://www.gcn.us.es/9BWMC/volume/24Porreca.pdf>
5. Păun, G.: P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics* 6(1), 75–90 (2001)
6. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing* 2(3), 287–298 (2003)
7. 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)
8. Toda, S.: PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing* 20(5), 865–877 (1991)
9. Valsecchi, A., Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: An efficient simulation of polynomial-space Turing machines by P systems with active membranes. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A.

- (eds.) Membrane Computing, 10th International Workshop, WMC 2009, Lecture Notes in Computer Science, vol. 6501, pp. 461–478. Springer (2010)
10. 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)