The counting power of P systems with antimatter

Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy

Abstract

We give a characterisation of the class of problems solved in polynomial time by uniform and semi-uniform families of P systems with active membranes, using matter/antimatter annihilation rules and elementary membrane division. Like several other variants of P systems with elementary division, this class is exactly $\mathbf{P}^{\#\mathbf{P}}$, that is, the problems solvable efficiently with access to oracles for counting problems. We also consider the *monodirectional* case, where objects in the P system can only move from inner regions towards outer regions. In that case, the above model of P systems characterises the class $\mathbf{P}_{\|}^{\#\mathbf{P}}$, where each query is independent of the result of the others; this contrasts with traditional P systems with active membranes, which characterise the (conjecturally proper) subclass $\mathbf{P}_{\|}^{\mathbf{NP}}$.

Keywords: membrane computing, P systems with active membranes, P systems with antimatter, computational complexity

1. Introduction

We consider *P systems with antimatter*, a shorthand to indicate P systems with active membranes using elementary membrane division and matter/antimatter annihilation rules, without charges, membrane dissolution and non-elementary membrane division. These P systems have already proved to be able to solve **NP**-complete problems in polynomial time [2] when the annihilation rules have priority over the application of any other rule. If this is not the case, then the class of solvable problems is reduced to **P** [1]; this is due to a loss of *context-sensitivity*, which is the ability of objects to interact with each other in some way, like it happens in cooperative rules, membranes with charges, and dissolution. Without context-sensitivity each object acts independently, and it can be shown that the result of the computation only depends on the presence of a single object in the initial configuration [2]. Hence, also P systems with active membranes without charges and without dissolution can only solve problems in **P** when working in polynomial time, even by using non-elementary division [4].

In many cases, when some kind of context-sensitivity is introduced, the class of problems which can be solved in polynomial time increases to $\mathbf{P}^{\#\mathbf{P}}$ [5], the class of problems solvable in polynomial time by deterministic Turing machines with oracles for counting problems, or even **PSPACE** if non-elementary division is used [13]. The exact amount of context-sensitivity needed to solve classically intractable problems by P systems is still unclear. For instance, while dissolution introduces context sensitivity, checking if it allows to solve problems outside **P** is the well-known P conjecture [12, Problem F], which is still an open problem.

In this paper we show that the context-sensitivity introduced by matter/antimatter annihilation rules allows not only to solve **NP**-complete problems, but also to match the computational power of traditional P systems with active membranes. In particular, we provide a method to actually exploit the exponential number of objects

Email addresses: alberto.leporati@unimib.it (Alberto Leporati), luca.manzoni@disco.unimib.it (Luca Manzoni), mauri@disco.unimib.it (Giancarlo Mauri), porreca@disco.unimib.it (Antonio E. Porreca), zandron@disco.unimib.it (Claudio Zandron)

resulting from the parallel simulation of a nondeterministic Turing machine, and we show that $\mathbf{P}^{\#\mathbf{P}}$ characterises the computing power of polynomial-time P systems with antimatter. This class already characterises the problems solved efficiently by P systems with active membranes without non-elementary division [5] (as well as being an upper bound for the computing power of tissue P systems with symport and antiport rules [7]). The lower bound can be determined through the simulation of both a deterministic Turing machine and a nondeterministic one that computes the $\#\mathbf{P}$ function; to define the upper bound one can exploit the $\#\mathbf{P}$ oracle to simulate the dividing membranes. These approaches are similar to the analogous proofs for traditional P systems with active membranes which can be found in [6, 5]. In the light of these results, it appears that $\mathbf{P}^{\#\mathbf{P}}$ is the natural class to express the computing power of P systems with elementary membrane division and some form of context-sensitivity.

We also show that when the P systems are *monodirectional*, i.e., no send-in rule is present, antimatter appears to be more powerful than charges. While the computing power of traditional monodirectional polynomial-time P systems with active membranes is limited to $\mathbf{P}^{\mathbf{NP}}$ or $\mathbf{P}_{\|}^{\mathbf{NP}}$ (the classes of decision problems solvable in deterministic polynomial time by Turing machines having respectively sequential or parallel access to an **NP** oracle), depending on the presence or absence of dissolution [8], P systems with antimatter can effectively use an exponential number of objects *in a single membrane*, thus attaining $\mathbf{P}_{\|}^{\#\mathbf{P}}$, which includes the whole polynomial hierarchy.

2. Basic notions

Let us start by defining the model of P systems we consider in this paper.

Definition 1. A *P system with antimatter* of initial degree $d \geq 1$ is a structure $\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$ where:

- $\Gamma = A \cup \bar{A}$, with $\bar{A} = \{\bar{a} : a \in A\}$ is an alphabet, i.e., a finite non-empty set of symbols, usually called *objects*; the elements of $\bar{A}$ are sometimes called *anti-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 labelled by elements of Λ in a one-to-one way;
- $w_{h_1}, \ldots, w_{h_d}$, with $h_1, \ldots, h_d \in \Lambda$, are the multisets over Γ initially contained in the d membranes of μ;
- R is a finite set of rules.

The rules in R are of the following types:

(a) *Object evolution rules*, of the form $[a \to w]_h$
They can be applied inside a membrane labelled by h 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 the objects in w).

(b) *Send-in communication rules*, of the form $a\,[\ ]_h \to [b]_h$
They can be applied to a membrane labelled by h such that the external region contains an occurrence of the object a; the object a is sent into h becoming b.

(c) *Send-out communication rules*, of the form $[a]_h \to [\ ]_h\, b$
They can be applied to a membrane labelled by h containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b.

(e) *Elementary division rules*, of the form $[a]_h \to [b]_h\,[c]_h$
They can be applied to a membrane labelled by h containing an occurrence of the object a but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes with the same label h; the object a is replaced, respectively, by b and c, while the other objects of the multiset are replicated in both membranes.

(g) *Annihilation rules*, of the form $[a\,\bar{a} \to \epsilon]_h$
These are applied inside a membrane labelled by h containing an occurrence of the object a and an occurrence of the anti-object $\bar{a}$, which both disappear. In this paper we assume implicitly the existence of this rule for each object a and label h.

The instantaneous *configuration* of a membrane consists of its label h and the multiset w of objects it contains at a given time. It is denoted by $[w]_h$. The *(full) configuration* $\mathcal{C}$ of a P system Π at a given time is a rooted, unordered tree. The root is a node corresponding to the external environment of Π, and has a single subtree corresponding to the current membrane structure of Π. Furthermore, the root is labelled by the multiset located in the environment, and the remaining nodes by the configurations $[w]_h$ of the corresponding membranes.

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 and annihilation rules: inside each membrane, several evolution or annihilation rules can be applied simultaneously.
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, division or annihilation rules must be subject to exactly one of them. Analogously, each membrane can only be subject to one communication or division rule (types (b)–(e)) per computation step. In other words, the only objects and membranes that do not evolve are those associated with no rule.
- Object annihilation rules are assumed to have priority over the other rules. This means that all pairs $a, \bar{a}$ consisting of an object and its anti-object are annihilated before applying any other kind of rule, leaving either only copies of a, or only copies of $\bar{a}$ (which can be subject to other kinds of rules), or neither (depending on the relative multiplicities).
- 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.
- In each computation step, all the chosen rules are applied simultaneously. However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps as follows. First, all annihilation and evolution rules are applied inside the elementary membranes, followed by all communication and division rules involving the membranes themselves; this process is then repeated for the membranes containing them, and so on towards the root of the membrane structure. In other words, the membranes evolve only after their internal configuration has been updated. For instance, before a membrane division occurs, all chosen annihilation and object evolution rules must be applied inside it; this way, the objects that are duplicated during the division are already the final ones.
- The outermost membrane cannot be divided, and objects sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ of configurations, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable from $\mathcal{C}_i$ via a single computation step, and no rules of Π are applicable in $\mathcal{C}_k$. A *non-halting* computation $\vec{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ 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 language *recognisers* by employing two distinguished objects *yes* and *no*: we assume that all computations are halting, and that either object *yes* or object *no* (but not both) is sent out from the outermost membrane, and only in the last computation step, in order to signal acceptance or rejection, respectively. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. When each reachable configuration has at most one successor, the P system is said to be *deterministic*.

In order to solve decision problems (or, equivalently, decide languages), we use *families* of recogniser P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. To each input x is associated a P system Π_x, deciding the membership of x in a language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for inputs of any length, as discussed in detail in [9].

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

- $F(1^n) = \Pi_n$, where n is the length of the input x and Π_n is a common P system for all inputs of length n with a distinguished input membrane.
- $E(x) = w_x$, where w_x is a multiset encoding the specific input x.

- Finally, Π_x is simply Π_n with w_x added to its input membrane.

The family $\mathbf{\Pi}$ is said to be (polynomial-time) semi-uniform if there exists a single deterministic polynomial-time Turing machine H such that $H(x) = \Pi_x$ for each $x \in \Sigma^\star$.

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description (e.g. binary notation for the quantities is not allowed), 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 [9] for further details on the encoding of P systems.

For brevity, the class of P systems with antimatter defined above, usually denoted by $\mathcal{AM}^0(+\mathrm{ant}, -\mathrm{d}, -\mathrm{ne})$, will be indicated with $\mathcal{A}$ (for "antimatter") in this paper. The *monodirectional* variant of this class, where send-in rules are disallowed, is here denoted by $\mathcal{A}(-\mathrm{i}) = \mathcal{AM}^0(+\mathrm{ant}, -\mathrm{i}, -\mathrm{d}, -\mathrm{ne})$. The complexity class of problems solvable in polynomial time by uniform (resp., semi-uniform) families of confluent P systems with antimatter is denoted by $\mathbf{PMC}_{\mathcal{A}}$ (resp., $\mathbf{PMC}^\star_{\mathcal{A}}$), and the corresponding class for monodirectional P systems is denoted by $\mathbf{PMC}_{\mathcal{A}(-\mathrm{i})}$ (resp., $\mathbf{PMC}^\star_{\mathcal{A}(-\mathrm{i})}$). In the case of deterministic P systems, the symbol **PMC** is replaced by **DPMC**.

We recall that the complexity class $\#\mathbf{P}$ consists of all functions $f : \Sigma^\star \to \mathbb{N}$ such that there exists a nondeterministic Turing machine having exactly $f(x)$ accepting computations on input $x \in \Sigma^\star$. The class $\mathbf{P}^{\#\mathbf{P}}$ consists of all decision problems solvable in deterministic polynomial time by Turing machines with access to an oracle for a $\#\mathbf{P}$ function, i.e., Turing machines that can compute the value $f(x)$ in a single computation step. Two restrictions of this class are $\mathbf{P}^{\#\mathbf{P}[1]}$, where only a single evaluation of f is allowed, and $\mathbf{P}_{\|}^{\#\mathbf{P}}$, where all evaluations of f are performed in parallel, i.e., all inputs on which f is to be evaluated must be fixed in advance, without choosing them based on the results of previous evaluations. As shown in Section 5, the equality $\mathbf{P}^{\#\mathbf{P}[1]} = \mathbf{P}_{\|}^{\#\mathbf{P}}$ holds. We refer the reader to Papadimitriou's book [10] for further details about complexity classes for Turing machines.

3. Simulating Turing machines

Let M be a deterministic Turing machine working in polynomial space $p(n)$, with tape alphabet $\Sigma = \{a_1, \ldots, a_k\}$, set of states $Q = \{q_1, \ldots, q_\ell\}$ disjoint from Σ, and transition function $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, +1\}$.

Let $x \in \Sigma^\star$ be an input for M of length n, and let $m = p(n)$. Suppose that, during the computation on input x, a configuration is reached where M is in state $q \in Q$, the tape head is located on cell $i \in \{1, \ldots, m\}$, and the tape contains the string $y = y_1 \cdots y_m \in \Sigma^m$. This configuration can be encoded by a membrane having the following configuration:

$$\mathcal{C} = [(y_1, 1) \cdots (y_i, i) \cdots (y_m, m)\,(q, i)]_M \tag{1}$$

Each object (y_j, j) describes the content of the j-th tape cell of M, and the object (q, i) encodes the state and the tape head position. Suppose that the next transition of M is described by $\delta(q, a) = (r, b, d)$ with $r \in Q$, $b \in \Sigma$ and $d \in \{-1, 0, +1\}$. This transition is simulated by the P system as follows.

The object (q, i) is rewritten by the rule

$$[(q, i) \to \overline{(a_1, i)} \cdots \overline{(a_k, i)}\,(q, a_1, i) \cdots (q, a_k, i)]_M \qquad \text{for } q \in Q \text{ and } 1 \le i \le m \tag{2}$$

where the objects $\overline{(a, i)}$, with $a \in \Sigma$, represent the antimatter counterpart of all possible objects (y_i, i) encoding the content of the i-th tape cell, while the objects (q, a, i), with $a \in \Sigma$, represent all possible tuples of the form (current state, scanned symbol, head position) for a fixed choice of q and i. The configuration at the beginning of the next step is thus

$$[(y_1, 1) \cdots (y_i, i) \cdots (y_m, m)\,\overline{(a_1, i)} \cdots \overline{(a_k, i)}\,(q, a_1, i) \cdots (q, a_k, i)]_M$$

By construction $\overline{(y_i, i)} = \overline{(a, i)}$ for some $a \in \Sigma$, and the objects (y_i, i) and $\overline{(y_i, i)}$ annihilate each other. The remaining objects $\overline{(a, j)}$ are rewritten by the rules

$$[\overline{(a, j)} \to \overline{(q_1, a, j)}' \cdots \overline{(q_\ell, a, j)}']_M \qquad \text{for } a \in \Sigma \text{ and } 1 \le j \le m \tag{3}$$

while the objects (q,a,i) are primed:

$$[(q,a,i) \to (q,a,i)']_M \qquad \text{for } q \in Q, a \in \Sigma \text{ and } 1 \leq i \leq m \tag{4}$$

In the next computation step, all objects $(q,a,i)'$ with $a \neq y_i$ generated by rule (4) annihilate with the objects generated by rule (3), leaving only the object $(q,y_i,i)'$. Any remaining anti-object is deleted by rewriting it to the empty multiset:

$$[\overline{(q,a,j)}' \to \epsilon]_M \qquad \text{for } a \in \Sigma \text{ and } 1 \leq j \leq m \tag{5}$$

After this annihilation, the content of the membrane is identical to that of configuration $\mathcal{C}$ of (1), except that the objects (y_i,i) and (q,i) have been merged into $(q,y_i,i)'$. The latter object contains enough information about the configuration of the Turing machine M to simulate the transition $\delta(q,y_i) = (r,b,d)$. This object is now rewritten into an object containing the result of the transition

$$[(q,a,i)' \to (r,b,d,i)'']_M \qquad \text{for } a \in \Sigma \text{ and } 1 \leq i \leq m \tag{6}$$

leading to the configuration

$$[(y_1,1) \cdots (y_{i-1},i-1)\,(r,b,d,i)''\,(y_{i+1},i+1) \cdots (y_m,m)]_M$$

In the next computation step, the object $(r,b,d,i)''$ is "unpacked" as

$$[(r,b,d,i)'' \to (b,i)\,(r,i+d)]_M \qquad \text{for } a \in \Sigma \text{ and } 1 \leq i \leq m$$

leading to the configuration

$$[(y_1,1) \cdots (y_{i-1},i-1)\,(b,i)\,(y_{i+1},i+1) \cdots (y_m,m)\,(r,i+d)]_M$$

which, according to the encoding established above, represents the configuration of the Turing machine M after one computation step.

Having shown how to simulate a generic computation step of M in three computation steps of the simulating P system, it suffices to initialise membrane M with the encoding of the initial configuration of the Turing machine and to send out the objects corresponding to accepting (resp., rejecting) final states as *yes* (resp., *no*) in order to obtain an efficient simulation of its computation. We have thus proved the following theorem.

Theorem 3. *A polynomial-space deterministic Turing machine running in time $t(n)$ can be simulated by a uniform family of deterministic single-membrane P systems with antimatter, running in time $3t(n)+1$ and with linear space overhead.* □

This simulation of a deterministic Turing machine can be easily extended to nondeterministic *polynomial-time* Turing machines if the membrane is allowed to divide (using elementary membrane division rules[1]), exploiting the usual time-space trade-off [11]. Without loss of generality, suppose that the simulated nondeterministic Turing machine N only makes binary nondeterministic choices, of the form $\delta(q,a) = \{(r,b,d),(s,c,e)\}$. Then, any such transition can be simulated by replacing rules (6) with the following elementary division rules:

$$[(q,a,i)']_N \to [(r,b,d,i)'']_N\,[(s,c,e,i)'']_N \qquad \text{for } a \in \Sigma \text{ and } 1 \leq i \leq m$$

The simulation then continues in parallel in the two copies of membrane N. Let us stress the fact that we simulate the nondeterminism of the Turing machine N using parallelism, thus obtaining a *deterministic* simulation.

When an instance of membrane N reaches the end of a simulated computation of the Turing machine, it sends out a *yes* or *no* object, depending on the result of the computation:

$$\begin{aligned} &[(q,i)]_N \to [\,]_N\,yes && \text{for } 1 \leq i \leq m, \text{ if } q \text{ is accepting} \\ &[(q,i)]_N \to [\,]_N\,no && \text{for } 1 \leq i \leq m, \text{ if } q \text{ is rejecting} \end{aligned} \tag{7}$$

These objects, which can be exponential in number, are collected in the outermost membrane h.

Theorem 4. *A polynomial-time nondeterministic Turing machine working in time $t(n)$ can be simulated by a uniform family of deterministic P systems with antimatter of depth 1 in time $3t(n)+1$ and with exponential space overhead.* □

[1] According to the usual semantics of P systems, the outermost membrane cannot divide; hence, the membrane employed for the simulation of a nondeterministic Turing machine must have an extra membrane surrounding it, thus obtaining a depth 1 membrane structure.

4. Simulating Turing machines with #P oracles

Let M be a deterministic Turing machine working in polynomial time $p(n)$ and having access to an oracle for a function $f \in \#\mathbf{P}$. We assume that M has a single tape (whose leftmost cell will be referred to as cell 1), where the odd-numbered cells are used for the input and as working space, and the even-numbered cells are used for writing the oracle query strings y and reading the outputs of the oracle $f(y)$, which overwrite the query strings. We also assume that, before entering the query state $q_?$, the tape head moves to tape cell 1.

Let N be a nondeterministic Turing machine working in polynomial time $q(n)$ and computing f, i.e., having exactly $f(x)$ accepting computations on input $x \in \Sigma^\star$. By hypothesis, machine M uses at most space $p(n)$, and its query strings have thus length at most $p(n)$. Hence, if we used machine N to answer the oracle queries of M instead of an actual oracle, each computation of N would run in time $q(p(n))$ and use at most $q(p(n))$ space. Since M can ask at most $p(n)$ queries on inputs of length n, the combined runtime of M and N would be $O\big(p(n)q(p(n))\big)$. We also establish the convention that N receives its input on the even-numbered tape cells, and that the odd-numbered ones may have an arbitrary initial content, rather than being blank. The rationale behind this convention is that we can use the tape of M when it enters its query state as the initial tape of N, thus simplifying the simulation described below.

We define a uniform family $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of P systems with antimatter running in polynomial time and simulating the combination of Turing machines M and N. Let $x = x_1 \cdots x_n \in \Sigma^\star$ be an input string of length n for M. The associated P system Π_x has the initial configuration

$$\big[\,[\,]_N\,(q_0,1)\,(x_1,1)\,(\sqcup,2)\,(x_2,3)\,(\sqcup,4)\,\cdots\,(\sqcup,2n-2)\,(x_n,2n-1)\,(\sqcup,2n)\,\cdots\,(\sqcup,m)\big]_M$$

where q_0 is the initial state of M, the symbol $\sqcup$ denotes a blank tape cell, and $m = q(p(n))$, which is an upper bound to the amount of space used by both M and N.

The computation of Π_x simulates the computation of M, as in the proof of Theorem 3 and ignoring membrane N, until M enters its query state $q_?$ with its tape head pointing at cell 1. The multiset encoding the tape of M is now replicated inside membrane N, as detailed below in Lemma 7. An object $(r_0, 1)$, representing the initial state of N with tape head in position 1, is also sent into N. The computation inside membrane M is now paused, while membrane N simulates the nondeterministic Turing machine N as in Theorem 4.

This simulation is carried out in time $O\big(q(p(n))\big)$, and the output objects generated by each instance of membrane N obtained by division are collected inside membrane M. The instances of membrane N are now emptied (see Lemma 8), and the objects *no* are deleted. The objects *yes*, instead, are counted (Lemma 11) and their number, in binary notation, is written in the leftmost even positions of the tape of M. The object $(r, 1)$, with r being the post-query state of M, is now produced and the simulation of M continues.

The first query of M potentially leaves behind up to exponentially many instances of membrane N, obtained by membrane division. In order to be able to carry out further queries, the tape content of M is replicated inside *all* instances of N. In fact, if we tried to send in a single copy of the objects encoding the tape, there would be no way to ensure that all objects end up inside the same instance of N, because the targets of a send-in rule $a\,[\,]_N \to [\,b]_N$ are nondeterministically chosen among all membranes with label N. Instead, by sending in enough copies of the first object simultaneously, then of the second object, and so on, it is guaranteed that each membrane gets a copy of the entire multiset, due to the maximally parallel application of the send-in rules. However, *only one* object $(r_0, 1)$ is sent in, ensuring that machine N is actually simulated only once.

4.1. Simulating the machine-oracle interface

We begin by proving that exponentially large rewriting rules can be simulated in polynomial time.

Lemma 5. *An object evolution rule of the form* $[a \to w]_h$ *can be simulated by* $O(\log|w|)$ *rules of the form* $[a' \to w']_h$ *with* $|w'| \le 2$ *in time* $\lceil \log|w| \rceil + 1$.

Proof. If $|w| < 2$, then the statement of the lemma already holds. Assume then $w \ge 2$. Furthermore, let assume that $|w| = 2^k$ for some k; if this is not the case, then we can pad w with "junk" objects # (i.e., objects not appearing in the left-hand side of any rule, or only appearing in deletion rules of the form $[\# \to \epsilon]_h$) without more than doubling the length of w.

We prove, by induction on $|w|$, that each object evolution rule of the form $[a \to w]_h$ can be simulated in two steps by rules having right-hand side of length at most $\frac{1}{2}|w|$. Assume $w = uv$ with $|u| = |v| = \frac{1}{2}|w|$, and let $\langle u \rangle$ and $\langle v \rangle$ be new atomic objects representing u and v, respectively. The rule $[a \to w]_h$ is then simulated in two steps by the three rules

$$[a \to \langle u \rangle\ \langle v \rangle]_h \qquad [\langle u \rangle \to u]_h \qquad [\langle v \rangle \to v]_h$$

The statement of the lemma follows by induction. □

We can now show how to distribute uniformly an object to all inner membranes in polynomial time, even when the number of membranes is exponential, and the exact number is unknown except for an upper bound.

Lemma 6. *Let Π be a P system with antimatter with alphabet Γ and let*

$$\mathcal{C} = \Big[\underbrace{[u_1]_k \cdots [u_\ell]_k}_{\ell \text{ instances}}\ a\ w \Big]_h$$

be a configuration of Π, with $a \in \Gamma$ and $u_1, \ldots, u_\ell, w \in \Gamma^\star$. Suppose that configuration $\mathcal{C}$ is inert, that is, no rule is applicable. Let L be the maximum number of instances of membrane k inside h across all computations of Π. Then, it is possible to augment Π with $O(\log L)$ rules such that, for all values of $\ell \leq L$, the P system reaches the configuration

$$\mathcal{D} = \Big[\underbrace{[u_1\ b]_k \cdots [u_\ell\ b]_k}_{\ell \text{ instances}}\ v\ w \Big]_h$$

That is, the object a triggers the appearance of object b inside each instance of k, and is rewritten into the multiset v. This procedure can be carried out in $O(\log L)$ time steps.

Proof. First of all, the rewriting rule $[a \to b_k^L\ (b'_k)^L\ a']_h$, generating L copies of b_k and b'_k, as well as priming a, is simulated in $O(\log L)$ steps according to Lemma 5. This leads to the configuration

$$\mathcal{C}_1 = \Big[\underbrace{[u_1]_k \cdots [u_\ell]_k}_{\ell \text{ instances}}\ b_k^L\ (b'_k)^L\ a'\ w \Big]_h$$

The objects b_k now enter the membranes k, rewritten as b; due to the maximally parallel semantics of P systems, each membrane k receives one of these objects, while the remaining copies of b_k stay inside membrane h. At the same time, the objects b'_k are rewritten as the anti-objects $\bar{b}_k$, and the object a' is primed again:

$$b_k\ [\ ]_k \to [b]_k \qquad [b'_k \to \bar{b}_k]_h \qquad [a' \to a'']_h$$

The P system thus reaches the following configuration:

$$\mathcal{C}_2 = \Big[\underbrace{[u_1\ b]_k \cdots [u_\ell\ b]_k}_{\ell \text{ instances}}\ b_k^{L-\ell}\ \bar{b}_k^L\ a''\ w \Big]_h$$

At the beginning of the next computation step, the anti-objects $\bar{b}_k$ annihilate all the remaining copies of b_k; then, by the following rules, any further copy of $\bar{b}_k$ is deleted and the object a'' is rewritten into v:

$$[\bar{b}_k \to \epsilon]_h \qquad [a'' \to v]_h$$

This produces configuration $\mathcal{D}$, as required. □

This allows us to show how the encoding of the tape of M can be replicated inside all instances of membrane N when performing the querying procedure, i.e., when object $(q_?, 1)$ appears.

Lemma 7. *Suppose the simulated Turing machine M enters its oracle query state $q_?$, and thus the P system is in a configuration of the form*

$$\mathcal{C} = \Big[\underbrace{[\ ]_N \cdots [\ ]_N}_{\ell \text{ instances}} (q_?, 1)\,(a_1, 1) \cdots (a_m, m) \Big]_M$$

Then, it is possible to replicate the encoding of the tape $(a_1, 1) \cdots (a_m, m)$ inside each instance of the internal membrane N, as follows:

$$\mathcal{D} = \Big[\underbrace{[(a_1, 1) \cdots (a_m, m)]_N \cdots [(a_1, 1) \cdots (a_m, m)]_N}_{\ell \text{ instances}}\ z_{3m+2}\,(r_0, 1)\,(a_1, 1) \cdots (a_m, m) \Big]_M$$

This procedure can be carried out in time $O(m \log L)$, where L is the maximum number of instances of membrane N among all reachable configurations.

Proof. The object $(q_?, 1)$ initially behaves according to rules (2)–(5) (i.e., as if simulating a transition of M). When the object $(q_?, a_1, 1)'$ appears, instead of applying rule (6), it is instead rewritten as follows:

$$[(q_?, a_1, 1)' \to (q_?, a_1, 1)''\,(a_1, 1)]_M$$

thus reaching the configuration

$$\Big[\underbrace{[\ ]_N \cdots [\ ]_N}_{\ell \text{ instances}} (q_?, a_1, 1)''\,(a_1, 1) \cdots (a_m, m) \Big]_M$$

By applying Lemma 6 with object $(q_?, a_1, 1)''$ as trigger, we can obtain the following configuration

$$\Big[\underbrace{[(a_1, 1)]_N \cdots [(a_1, 1)]_N}_{\ell \text{ instances}} (q_?, 2)\,(a_1, 1) \cdots (a_m, m) \Big]_M$$

in $O(\log L)$ time steps. By recursively applying the same procedure, we can copy the whole multiset encoding the tape of M inside all instances of membrane N. The final object $(q_?, m+1)$ is rewritten as

$$[(q_?, m+1) \to z_{3m+2}\,(r_0, 1)]_M$$

producing the required configuration $\mathcal{D}$ in time $O(m \log L)$. □

After having replicated the tape of M inside all instances of membrane N, the object $(r_0, 1)$, encoding the initial state and tape head position of Turing machine N, is sent inside one membrane N, thus starting the simulation of the Turing machine inside that membrane; simultaneously, the subscript of z_{3m+2} is decremented:

$$\begin{aligned} &(r_0, 1)\,[\ ]_N \to [(r_0, 1)]_N \\ &[z_{3m+2} \to z_{3m+1}]_M \end{aligned} \tag{8}$$

So doing, the P system reaches the configuration

$$\Big[\underbrace{[(a_1, 1) \cdots (a_m, m)]_N \cdots [(a_1, 1) \cdots (a_m, m)]_N}_{\ell-1 \text{ instances}} [(a_1, 1) \cdots (a_m, m)\,(r_0, 1)]_N\ z_{3m+1}\,(a_1, 1) \cdots (a_m, m) \Big]_M \tag{9}$$

The simulation of N can now begin; as established by the conventions above, the initial tape encoded by the multiset $(a_1, 1) \cdots (a_m, m)$ has the query string (i.e., the input to N) in the even-numbered positions of the tape, and arbitrary content in the odd-numbered positions. The instances of membrane N not containing the object $(r_0, 1)$ will stay inert during the simulation.

While the simulation of Turing machine N is carried out, the object z_{3m+1} counts down for a number of steps sufficient to ensure that the simulation is completed:

$$[z_t \to z_{t-1}]_M \qquad \text{for } 3m+1 \geq t \geq 1$$

When the object z_t becomes z_0, it is rewritten as object e:

$$[z_0 \to e]_M$$

Simultaneously, the instances of membrane N involved in the simulation of the current query send out their result objects *yes*. According to the proof of Theorem 4, the membranes simulating rejecting computations send out a *no* result object; however, the objects *no* are not needed here, so we can replace rule (7) with

$$[(q,i) \to \epsilon]_N \qquad \text{for } 1 \leq i \leq m, \text{ if } q \text{ is rejecting}$$

The membranes with label N that complete the simulation, as well as those that contain the encoding of the tape of M before entering the query but remained inert, can now be emptied to prepare them for any further query. This procedure will also generate the object t, that will trigger the execution of the next phase.

Lemma 8. *If $u_1, \ldots, u_\ell$ are multisets encoding tapes of the Turing machine M, and the assumptions of Lemma 6 apply, then the following (multi-step) transition between configurations*

$$\mathcal{C} = \Big[\underbrace{[u_1]_N \cdots [u_\ell]_N}_{\ell \text{ instances}} e\ w \Big]_M \qquad \Rightarrow^\star \qquad \Big[\underbrace{[\,]_N \cdots [\,]_N}_{\ell \text{ instances}} t\ w \Big]_M = \mathcal{D}$$

can be computed in $O(m)$ computation steps for any multiset $w \in \Gamma^\star$.

Proof. By applying Lemma 6, triggered by object e, the system reaches the configuration

$$\Big[\underbrace{[u_1\ e]_N \cdots [u_\ell\ e]_N}_{\ell \text{ instances}} e'\ w \Big]_M$$

The objects e inside N are rewritten as the antimatter counterparts of all objects in $\Sigma \times \{1, \ldots, m\}$, in order to annihilate the multisets $u_1, \ldots, u_\ell$:

$$[e \to u]_N \qquad \text{where } u = \left\{ \overline{(a,i)} : a \in \Sigma, 1 \leq i \leq m \right\}$$

In the next computation step, the annihilation takes place, deleting $u_1, \ldots, u_\ell$. The remaining objects $\overline{(a,i)}$ trigger rule (3) and, in the next step, rule (5), thus leaving all instances of membrane N empty.

While the membranes are emptied, the object e' in M waits for two steps before finally producing object t:

$$[e' \to e'']_M \qquad [e'' \to e''']_M \qquad [e''' \to t]_M$$

which gives the final configuration $\mathcal{D}$. □

To allow the simulated Turing machine M to access the output of the oracle, it is necessary to write the number of objects *yes* currently present in membrane M on the leftmost even-numbered positions of its tape.

Let B be the index (counting from zero) of the most significant bit of the largest possible output of the oracle during a computation of M on an input of length n. Since the output of the oracle consists of a number of bits at most equal to the number of even-numbered cells in a tape of length $p(n)$, we have $B = \lfloor p(n)/2 \rfloor - 1$. We will extract the bits of that number one by one, starting from the B-th one.

Since this process involves deleting the objects using annihilation, it is first necessary to provide a way to create a "backup" copy of the multiset:

Lemma 9. *Let $\mathcal{C} = [a^\ell\ p\ w]_M$ be a configuration of membrane M, with w being the encoding of the tape of Turing machine M. Assume that*

- *the number of copies of a, b and c inside M never exceeds $L = 2^{B+1} - 1$ in any reachable configuration;*
- *the object p appears with multiplicity at most 1 inside M in any reachable configuration;*
- *no preexisting rule is enabled in $\mathcal{C}$ or involves $\bar{a}$ on the left-hand side.*

Then, it is possible to augment the P system with $O(\log L)$ rules such that $\mathcal{C}$ leads to configuration $\mathcal{D} = [\, b^\ell \; c^\ell \; w \; v \,]_M$ for some $v \in \Gamma^\star$, that is, each copy of object a is replicated as both b and c, while object p is rewritten as v, and w is left untouched; this can be performed in $O(\log L)$ steps.

Proof. First, we apply the rule $[\, p \to p' \; \bar{a}^L \,]_M$, which is simulated according to Lemma 5 and leads to configuration $\mathcal{C}' = [\, a^\ell \; \bar{a}^L \; p' \; w \,]_M$ in $\lceil \log(L+1) \rceil + 1$ steps. At the beginning of the next computation step, the multiset $\bar{a}^L$ deletes all ℓ copies of a; the remaining $L - \ell$ copies of $\bar{a}$ evolve according to the rule $[\, \bar{a} \to \bar{x} \; \bar{y} \,]_M$, where x and y are new auxiliary objects, and p' evolves according to $[\, p' \to p'' \,]_M$. This leads to configuration $\mathcal{C}'' = [\, \bar{x}^{L-\ell} \; \bar{y}^{L-\ell} \; p'' \; w \,]_M$. From here, the rule $[\, p'' \to p''' \; x^L \; y^L \,]_M$ is simulated as in Lemma 5, leading to the configuration $\mathcal{C}''' = [\, x^L \; \bar{x}^{L-\ell} \; y^L \; \bar{y}^{L-\ell} \; p''' \; w \,]_M$ after $\lceil \log(2L+1) \rceil + 1$ steps. In the next computation step, $L - \ell$ copies of x and y are deleted, the remaining copies of x and y are rewritten into b and c, respectively, by the rules $[\, x \to b \,]_M$ and $[\, y \to c \,]_M$, and the object p''' is rewritten as $[\, p''' \to v \,]_M$. This leads to the configuration $\mathcal{D} = [\, b^\ell \; c^\ell \; w \; v \,]_M$, as required. □

The bit-extraction procedure begins by renaming the objects *yes* as a_B. These objects are, by hypothesis, at most $2^{B+1} - 1$; in general, there will be at most $2^{i+1} - 1$ copies of object a_i, with $0 \le i \le B$, during this procedure. Suppose that we have already extracted the first $B - i$ most significant bits of the number of *yes* objects; then, the next most significant bit to extract is the i-th bit of the number of objects a_i. The procedure is triggered by the object t_i, and is described in the following lemma.

Lemma 10. *Let $\mathcal{C} = [\, a_i^\ell \; t_i \; w \,]_M$ be a configuration of membrane M with $0 \le i \le B$, $0 \le \ell < 2^{i+1}$, and w an encoding of the tape of the Turing machine M. Let $L = 2^{B+1} - 1$. Then, it is possible to augment the P system with $O(\log L)$ rules extracting the i-th bit of ℓ (counting from 0); that is, configuration $\mathcal{C}$ leads to $\mathcal{D} = [\, a_{i-1}^{\ell - 2^i} \; t_{i-1} \; 1_i \; w \,]_M$ if $\ell \ge 2^i$, or to $\mathcal{D} = [\, a_{i-1}^{\ell} \; t_{i-1} \; 0_i \; w \,]_M$ if $\ell < 2^i$; in both cases, the number of copies of a_{i-1} is $\ell \bmod 2^i$. The procedure is carried out in $O(\log L)$ steps, and the multiset w is left untouched.*

Proof. Starting from configuration $\mathcal{C} = [\, a_i^\ell \; t_i \; w \,]_M$, we can reach configuration $\mathcal{C}_1 = [\, a_{i-1}^\ell \; b_{i,i+2}^\ell \; t_i' \; w \,]_M$ in $O(\log L)$ steps with $O(\log L)$ extra rules, as proved by Lemma 9. In $\mathcal{C}_1$, the rewriting rule $[\, t_i' \to t_i'' \; \bar{b}_i^{2^i} \,]_M$ is simulated, as per Lemma 5, in $\lceil \log(2^i + 1) \rceil + 1 = i + 2$ steps. Simultaneously, the objects $b_{i,i+1}$ are rewritten according to the rules

$$[\, b_{i,j} \to b_{i,j-1} \,]_M \quad \text{for } 1 < j \le i + 2 \qquad\qquad [\, b_{i,1} \to b_i \,]_M$$

So doing, after $i + 2$ steps the ℓ copies of b_i and the 2^i copies of $\bar{b}_i$ appear simultaneously, leading to configuration $\mathcal{C}_2 = [\, a_{i-1}^\ell \; b_i^\ell \; \bar{b}_i^{2^i} \; t_i'' \; w \,]_M$. In the next computation step, the object t_i'' is rewritten by rule $[\, t_i'' \to y_i \; n_i \,]_M$, while the copies of b_i and $\bar{b}_i$ pairwise annihilate. Any remaining copy of b_i is deleted by the rule $[\, b_i \to \epsilon \,]_M$, while any remaining copy of $\bar{b}_i$ is rewritten by $[\, \bar{b}_i \to \bar{y}_i \,]_M$. The next configuration depends on whether $\ell \ge 2^i$ or $\ell < 2^i$:

- If $\ell \ge 2^i$, then we reach configuration $\mathcal{C}_3 = [\, a_{i-1}^\ell \; y_i \; n_i \; w \,]_M$. From here object y_i evolves by rule $[\, y_i \to y_i' \; \bar{n}_i' \,]_M$, while n_i is rewritten by $[\, n_i \to n_i' \,]_M$; this leads to configuration $\mathcal{C}_4 = [\, a_{i-1}^\ell \; y_i' \; \bar{n}_i' \; n_i' \; w \,]_M$. While objects $\bar{n}_i'$ and n_i' annihilate, the rule $[\, y_i' \to y_i'' \; \bar{a}_{i-1}^{2^i} \,]_M$ is simulated according to Lemma 5; hence, in $\lceil \log(2^i + 1) \rceil + 1$ steps we reach configuration $\mathcal{C}_5 = [\, a_{i-1}^\ell \; y_i'' \; \bar{a}_{i-1}^{2^i} \; w \,]_M$. Finally 2^i copies of a_{i-1} and $\bar{a}_{i-1}$ annihilate, and rule $[\, y_i'' \to t_{i-1} \; 1_i \,]_M$ is applied, leading to $\mathcal{D} = [\, a_{i-1}^{\ell - 2^i} \; t_{i-1} \; 1_i \; w \,]_M$ as required.
- If $\ell < 2^i$, then configuration $\mathcal{C}_3 = [\, a_{i-1}^\ell \; \bar{y}_i^{2^i - \ell} \; y_i \; n_i \; w \,]_M$ is reached. Here, the object y_i is annihilated by one copy of $\bar{y}_i$, while the remaining copies are deleted by the rule $[\, \bar{y}_i \to \epsilon \,]_M$. As above, rule $[\, n_i \to n_i' \,]_M$ is also applied, leading to configuration $\mathcal{C}_4 = [\, a_{i-1}^\ell \; n_i' \; w \,]_M$. Now rule $[\, n_i' \to t_{i-1} \; 0_i \,]_M$ is applied, leading to the configuration $\mathcal{D} = [\, a_{i-1}^\ell \; t_{i-1} \; 0_i \; w \,]_M$ as required.

In both cases, the final configuration $\mathcal{D}$ is reached in $O(\log L)$ computation steps, using $O(\log L)$ additional rules. □

Lemma 11. *Let $\mathcal{C} = [yes^{\ell}\ t\ w]_M$ be a configuration of membrane M with $0 \leq \ell < 2^{B+1}$ and w being the encoding of the tape of Turing machine M. Let $L = 2^{B+1} - 1$. Then, it is possible to augment the P system with $O\big((\log L)^2\big)$ rules converting ℓ in binary; that is, configuration $\mathcal{C}$ leads to $\mathcal{D} = [\ell_B\ \cdots\ \ell_1\ \ell_0\ s\ w]_M$, where $\ell_B\ \cdots\ \ell_1\ \ell_0$ is the binary encoding of ℓ, with each ℓ_i being either 0_i or 1_i. The procedure is carried out in $O\big((\log L)^2\big)$ steps, and the multiset w is left untouched.*

Proof. By applying Lemma 10 with $a_B = yes$ and $t_B = t$ and $i = B$, either configuration $[a_{B-1}^{\ell - 2^B}\ 1_B\ t_{B-1}\ w]_M$ or configuration $[a_{B-1}^{\ell}\ 0_B\ t_{B-1}\ w]_M$ is reached, depending on the value of the most significant bit of ℓ, in time $O(\log L)$. The procedure can then be repeated further $B - 1 = O(\log L)$ times to obtain the other bits of ℓ. The final configuration reached has the form $\mathcal{D} = [\ell_B\ \cdots\ \ell_1\ \ell_0\ t_{-1}\ w]_M$, and by letting $s = t_{-1}$ the statement of the lemma follows. □

After having obtained the binary encoding of the answer of the oracle, we reach a configuration of the form

$$\Big[\underbrace{[\ ]_N \cdots [\ ]_N}_{\text{any number}}\ (a_1, 1)\ (a_2, 2)\ \cdots\ (a_m, m)\ \ell_B\ \ell_{B-1}\ \cdots\ \ell_0\ s\Big]_M$$

It is now necessary to clear the first $B + 1$ even positions $(2, 4, \ldots, 2B + 2)$ of the tape of Turing machine M, and replace their content with the bits $\ell_B \cdots \ell_0$. In order to do so, the object s is rewritten as

$$[s \to \bar{0}_B\ \bar{1}_B\ \bar{0}_{B-1}\ \bar{1}_{B-1}\ \cdots\ \bar{0}_0\ \bar{1}_0\ s']_M$$

Half of these objects are annihilated in the next computation step, while the remaining ones produce the anti-objects clearing up the initial segment of the even cells of the tape

$$[\bar{0}_i \to \overline{(a_1, 2i+2)}\ \cdots\ \overline{(a_k, 2i+2)}\ \bar{0}'_i]_M \qquad \text{for } 0 \leq i \leq B$$
$$[\bar{1}_i \to \overline{(a_1, 2i+2)}\ \cdots\ \overline{(a_k, 2i+2)}\ \bar{1}'_i]_M \qquad \text{for } 0 \leq i \leq B$$

At the same time, the object s' is primed again:

$$[s' \to s'']_M$$

The objects of the form $\overline{(a_j, 2i+2)}$ that are not annihilated are rewritten by rule (3), while the remaining objects are further primed:

$$[\bar{0}'_i \to \bar{0}''_i]_M \qquad [\bar{1}'_i \to \bar{1}''_i]_M \qquad \text{for } 0 \leq i \leq B$$
$$[s'' \to s''']_M$$

The objects produced by rule (3) are deleted in the next step by rule (5), while the objects $\bar{0}''_i$ and $\bar{1}''_i$ update the content of the tape, and object s''' produces the encoding of post-query state r and tape head position 1:

$$[\bar{0}''_i \to (1, 2i+2)]_M \qquad [\bar{1}''_i \to (0, 2i+2)]_M \qquad \text{for } 0 \leq i \leq B$$
$$[s''' \to (r, 1)]_M$$

This completes the simulation of the querying procedure. The computation can now proceed inside membrane M according to Theorem 3, and further queries can be simulated with the same construction.

The slowdown of the simulation is polynomial, and the rules of the P system can be easily computed in polynomial time, independently of the specific input string x for M. Furthermore, the simulation of both Turing machines M and N is deterministic, as well as the interface between them. Indeed, the only potentially problematic rule is (8), which sends the object $(r_0, 1)$ into a nondeterministically chosen instance of membrane N; however, all instances of N are indistinguishable, since they have (by construction) the same content in that moment, thus leading to a single possible successor configuration.

Theorem 12. *A polynomial-time deterministic Turing machine with an oracle for* #**P** *can be simulated by a uniform family of deterministic P systems with antimatter of depth* 1*, with polynomial slowdown and exponential space overhead.* □

Corollary 13. $\mathbf{P}^{\#\mathbf{P}} \subseteq \mathbf{DPMC}_{\mathcal{A}} \subseteq \mathbf{PMC}_{\mathcal{A}}$. □

5. Monodirectional simulation of Turing machines with #P oracles

The simulation in Section 4 employs send-in communication rules to interface the simulated Turing machines M and N. Now let us consider *monodirectional* P systems [8], where no send-in rule is available, and communication only proceeds from internal membranes to more external ones. In this variant, membranes having the same nesting depth in the membrane structure evolve independently, since they cannot exchange objects between them (as this would require a send-in rule). Hence, when simulating nondeterministic Turing machines N as in Theorem 4, each membrane only simulates a single computation of N, and the total number of accepting computations of N is only accessible (as the number of *yes* objects) in the parent membrane. Since the latter membrane is non-elementary, it is unable to divide and thus to simulate further oracle queries using the algorithm described above.

It is however possible to simulate a single #**P** query by beginning the simulation of Turing machine M inside the innermost membrane; when the machine enters the query state, the multiset encoding its tape is duplicated and sent to the outside region. Turing machine N is then simulated in the innermost membrane as in Theorem 4. The output is collected outside, and the simulation of M can then proceed as before (in the outer region this time).

We begin by observing that a polynomial-time Turing machine asking a single #**P** query is actually equivalent to one asking polynomially many *parallel* queries (i.e., a query string cannot depend on the result of previous queries). The corresponding complexity classes $\mathbf{P}^{\#\mathbf{P}[1]}$ and $\mathbf{P}_{\|}^{\#\mathbf{P}}$ are thus equivalent:

Lemma 14. $\mathbf{P}^{\#\mathbf{P}[1]} = \mathbf{P}_{\|}^{\#\mathbf{P}}$.

Proof. By definition, a single query does not depend on the result of previous queries, hence $\mathbf{P}^{\#\mathbf{P}[1]} \subseteq \mathbf{P}_{\|}^{\#\mathbf{P}}$.

Let M be a Turing machine working in polynomial time $p(n)$ and asking parallel oracle evaluations of the function $f \in \#\mathbf{P}$. Let $x_1, \ldots, x_m$ be the query strings for a particular input of M. Then, the same information $f(x_1), \ldots, f(x_m)$ obtained by querying the oracle can be obtained via a single query to an oracle for the function

$$g(x_1\$x_2\$\cdots\$x_m\$K) = \sum_{i=1}^{m} K^{i-1} f(x_i)$$

where \$ is a separator, and K is a strict upper bound for the value of f on strings of length at most $p(n)$. A suitable and easily computed value of K is $2^{q(n)} + 1$, where $q(n)$ is the runtime of any polynomial-time nondeterministic Turing machine having $f(x)$ accepting computations on each input string x of length n. Indeed, $2^{q(n)}$ is the maximum number of computations of the machine on inputs of length n. Essentially, we are encoding $f(x_1), \ldots, f(x_m)$ as a single K-ary number; the value $f(x_i)$ can be recovered in polynomial time as

$$f(x_i) = \left\lfloor \frac{g(x_1\$x_2\$\cdots\$x_m\$K)}{K^{i-1}} \right\rfloor \bmod K$$

The function g is itself in #**P**, since this class is closed under addition and multiplication [3]. □

Now let M be a deterministic Turing machine which can perform a single query to a function $f \in \#\mathbf{P}$, and let N be a nondeterministic Turing machine having $f(x)$ accepting computations on each input $x \in \Sigma^\star$. A P system Π_x simulating the combination of M and N on input x consists of two nested membranes $[[\ ]_k]_h$. The simulation of M *before* asking its query is performed, as in Theorem 3, inside membrane k. When M enters its query state $q_?$, the encoding of its tape is duplicated in the outermost membrane h:

Lemma 15. *Suppose the P system Π_x is in a configuration of the form*

$$\mathcal{C} = \left[[(a_1, 1) \cdots (a_m, m)\, (q_?, 1)]_k \right]_h$$

Then, it is possible to reach the configuration

$$\mathcal{D} = \left[[(a_1, 1) \cdots (a_m, m)\, (r_0, 1)]_k (a_1, 1) \cdots (a_m, m)\, z_{3m+1} \right]_h$$

in time $O(m)$ using $O(m)$ rules.

Proof. As in the proof of Lemma 7, the object $(q_?, 1)$ initially behaves as if simulating a transition of M, except that the object $(q_?, a_1, 1)'$, instead of triggering rule (6), is rewritten as

$$[(q_?, a_1, 1)' \to (q_?, a_1, 1)''\, (a_1, 1)\, (a_1, 1)']_k$$

In the next computation step, the object $(a_1, 1)'$ is sent out as $(a_1, 1)$, and the object $(q_?, a_1, 1)''$ is rewritten as $(q_?, 2)$:

$$[(a_1, 1)']_k \to [\ ]_k\, (a_1, 1) \qquad\qquad [(q_?, a_1, 1)'' \to (q_?, 2)]_k$$

The P system thus reaches the configuration

$$\mathcal{C}_1 = \left[[(a_1, 1) \cdots (a_m, m)\, (q_?, 2)]_k\, (a_1, 1) \right]_h$$

Analogously, the object $(q_?, 2)$ can trigger the duplication of $(a_2, 2)$ inside membrane h, and so on. After having duplicated the entire encoding of the tape of M, the P system reaches the configuration

$$\mathcal{C}_2 = \left[[(a_1, 1) \cdots (a_m, m)\, (q_?, m+1)]_k\, (a_1, 1) \cdots (a_m, m) \right]_h$$

The object $(q_?, m+1)$ is then rewritten:

$$[(q_?, m+1) \to (q_?, m+1)'\, z'_{3m+1}]_k$$

and, in the following step, the desired configuration $\mathcal{D}$ is obtained by applying the rules

$$[z'_{3m+1}]_k \to [\ ]_k\, z_{3m+1} \qquad\qquad [(q_?, m+1)' \to (r_0, 1)]_k$$ □

The configuration $\mathcal{D}$ obtained via Lemma 15 is the same as configuration (9) with $\ell - 1 = 0$ and with labels $M = h$ and $N = k$. The oracle can thus be simulated as before, and its output obtained in membrane h. Since machine M only asks this single query, the simulation of the remaining part of its computation can be carried out entirely inside membrane h, without the need for send-in rules. This proves the following result:

Theorem 16. *A polynomial-time deterministic Turing machine with an oracle for #***P** *asking parallel queries can be simulated by a uniform family of deterministic* monodirectional *P systems with antimatter of depth* 1*, with polynomial slowdown and exponential space overhead.* □

Corollary 17. $\mathbf{P}_{\parallel}^{\#\mathbf{P}} \subseteq \mathbf{DPMC}_{\mathcal{A}(-\mathrm{i})} \subseteq \mathbf{PMC}_{\mathcal{A}(-\mathrm{i})}$. □

6. Upper bounds to P systems with antimatter

An upper bound to the computing power of families of confluent P systems with antimatter can be obtained by comparing them to traditional P systems with active membranes without antimatter [11]. In particular, traditional P systems with active membranes without charges and membrane division limited to elementary membranes have a superset of the rules of the P systems with antimatter investigated in this paper, with the exception of the annihilation of objects and anti-objects[2]. The class of problems solved in polynomial time by semi-uniform

[2] Traditional P systems with active membranes also include membrane dissolution rules.

families of traditional P systems with elementary active membranes without charges is known to have a $\mathbf{P}^{\#\mathbf{P}}$ upper bound, which holds for both the uniform and the semi-uniform case [5]. The simulation algorithm that proves this result can be extended to any type of rule not involving membrane division, as long as it can be simulated in polynomial time given the configuration of a membrane with object multiplicities in binary notation. This is indeed the case for the annihilation of objects. Furthermore, the simulation supports giving higher priority to certain kinds of rules; this allows us to faithfully simulate P systems with antimatter and thus obtain the same upper bound, which, together with Corollary 13, implies a precise characterisation of their computing power:

Theorem 18. $\mathbf{PMC}_{\mathcal{A}}^{[\star]} = \mathbf{DPMC}_{\mathcal{A}}^{[\star]} = \mathbf{P}^{\#\mathbf{P}}$, *where* $[\star]$ *denotes optional semi-uniformity.* □

We are also able to prove that $\mathbf{P}_{\|}^{\#\mathbf{P}} = \mathbf{P}^{\#\mathbf{P}[1]}$ is an exact upper bound for the class of problems solved by monodirectional P systems with antimatter. The reasoning is similar to the proof of the $\mathbf{P}_{\|}^{\mathbf{NP}}$ upper bound to the power of traditional monodirectional P systems with active membranes working in polynomial time [8].

The idea is that the non-elementary (thus non-dividing) membranes of a monodirectional P system with antimatter can be simulated deterministically in polynomial time. This result is known as the "Milano Theorem" in the case of traditional P systems with active membranes [14], and can be easily extended to annihilation rules with priority, as described above. In order to complete the simulation, it is necessary to update the content of the membranes containing elementary membranes with the objects sent out at each computation step by the latter. This can be accomplished with access to an oracle for the following query:

> Given the configuration of an elementary membrane h, an integer $t \in \mathbb{N}$ in unary notation, and an object type $a \in \Gamma$, how many copies of a are sent out at time t by membranes descending from h?

where "descending" means obtained from the original one via zero or more membrane divisions.

This query is easily shown to be in $\#\mathbf{P}$. Indeed, consider a nondeterministic Turing machine N simulating membrane h; as long as no membrane division occurs, the simulation can be performed deterministically in polynomial time, with priority to annihilation, as mentioned above [14]. When an elementary division occurs, the machine chooses nondeterministically which of the two resulting membranes to simulate. This way, each computation of N simulates a single instance of membrane h. After simulating t time steps of h, the machine accepts if and only if a copy of object a was sent out in the last step. Hence, the number of accepting computations of N coincides with the number of copies of a sent out by descendants of h.

Since we are dealing with *monodirectional* P systems, the computation of an elementary membrane is entirely determined by its own initial configuration, since it cannot receive any object from the outside[3]. As a consequence, the outputs at each time step of all elementary membranes h can be computed by multiple independent (i.e., parallel) queries, one for each object type $a \in \Gamma$ and each time t (which is polynomial by hypothesis).

Together with Corollary 17, this implies the following characterisation of the computing power of monodirectional P systems with antimatter working in polynomial time.

Theorem 19. $\mathbf{PMC}_{\mathcal{A}(-\mathrm{i})}^{[\star]} = \mathbf{DPMC}_{\mathcal{A}(-\mathrm{i})}^{[\star]} = \mathbf{P}_{\|}^{\#\mathbf{P}} = \mathbf{P}^{\#\mathbf{P}[1]}$, *where* $[\star]$ *denotes optional semi-uniformity.* □

Remark 20. Notice that the upper bounds of this paper, and trivially the lower bounds, continue to hold even if charges (and dissolution, in the bidirectional case) are allowed; this can be proved by slightly adapting the proofs of the previously known upper bounds [5, 8].

7. Discussion

The results of this paper, combined with the analogous results for traditional P systems with active membranes and tissue P systems [5, 6, 7], suggest that whenever a variant of P system with elementary division is efficient enough to solve **NP**-complete problems in polynomial time, then it is also able to embed that solution as a subroutine, which can be employed to simulate a $\#\mathbf{P}$ oracle, thus reaching the complexity class $\mathbf{P}^{\#\mathbf{P}}$. As long

[3] If the P systems were not monodirectional, the query would need, as an additional parameter, the list of objects sent in during the computation up to time step t, which introduces further complications [5].

as the rules applied inside each membrane (i.e., excluding membrane division) can be simulated in polynomial time, the class $\mathbf{P}^{\#\mathbf{P}}$ also constitutes an upper bound, giving us an exact characterisation. It is an open problem to establish whether there is a way to formalise this intuition, or if there exist variants of P systems with elementary membrane division that escape this characterisation, that is, they solve a class of problems that is neither **P**, nor $\mathbf{P}^{\#\mathbf{P}}$, nor **PSPACE**.

Furthermore, the monodirectional variants of these classes of P systems seem to often characterise classes of the form $\mathbf{P}_{\|}^{C}$, although in this case the oracle class C seems to be dependent on the precise combination of rules of the model. In particular, the results of this paper highlight a crucial difference between the context-sensitivity provided by charges [8] and that provided by antimatter: indeed, the latter allows us to actually exploit an exponential number of objects inside a single membrane (namely, the result objects *yes* representing the accepting computations of the nondeterministic Turing machine in Section 4.1), while in the case of monodirectional P systems with charges the evolution of the system only depends on a polynomial-sized amount of objects, as shown in [8, Lemma 1].

Acknowledgements

This work has been supported by Fondo d'Ateneo (FA) 2015 of Università degli Studi di Milano-Bicocca: "Complessità computazionale e applicazioni crittografiche di modelli di calcolo bioispirati".

Bibliography

[1] D. Díaz-Pernil, A. Alhazov, R. Freund, M.A. Gutiérrez-Naranjo, A. Leporati, Recognizer P systems with antimatter, Romanian Journal of Information Science and Technology 18 (2015) 201–217. URL: http://www.imt.ro/romjist/Volum18/Number18_3/abstracts.htm.

[2] D. Díaz-Pernil, F. Peña Cantillana, A. Alhazov, R. Freund, M.A. Gutiérrez-Naranjo, Antimatter as a frontier of tractability in membrane computing, Fundamenta Informaticae 134 (2014) 83–96. URL: http://dx.doi.org/10.3233/FI-2014-1092.

[3] L. Fortnow, Counting complexity, in: L.A. Hemaspaandra, A.L. Selman (Eds.), Complexity Theory Retrospective II, Springer, 1997, pp. 81–107.

[4] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Nuñez, F.J. Romero-Campero, Computational efficiency of dissolution rules in membrane systems, International Journal of Computer Mathematics 83 (2006) 593–611. URL: http://dx.doi.org/10.1080/00207160601065413.

[5] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Simulating elementary active membranes, with an application to the P conjecture, in: M. Gheorghe, G. Rozenberg, P. Sosík, C. Zandron (Eds.), Membrane Computing, 15th International Conference, CMC 2014, volume 8961 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 284–299. URL: http://dx.doi.org/10.1007/978-3-319-14370-5_18.

[6] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Membrane division, oracles, and the counting hierarchy, Fundamenta Informaticae 138 (2015) 97–111. URL: http://dx.doi.org/10.3233/FI-2015-1201.

[7] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Tissue P systems can be simulated efficiently with counting oracles, in: G. Rozenberg, A. Salomaa, J.M. Sempere, C. Zandron (Eds.), Membrane Computing, 16th International Conference, CMC 2015, volume 9504 of *Lecture Notes in Computer Science*, pp. 251–261. URL: http://dx.doi.org/10.1007/978-3-319-28475-0_17.

[8] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Monodirectional P systems, Natural Computing (2016). URL: http://dx.doi.org/10.1007/s11047-016-9565-2, in press.

[9] N. Murphy, D. Woods, The computational power of membrane systems under tight uniformity conditions, Natural Computing 10 (2011) 613–632. URL: http://dx.doi.org/10.1007/s11047-010-9244-7.

[10] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1993.

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

[12] Gh. Păun, Further twenty six open problems in membrane computing, in: M.A. Gutíerrez-Naranjo, A. Riscos-Nuñez, F.J. Romero-Campero, D. Sburlan (Eds.), Proceedings of the Third Brainstorming Week on Membrane Computing, Fénix Editora, 2005, pp. 249–262. URL: http://www.gcn.us.es/3BWMC/Volumen.htm.

[13] P. Sosík, A. Rodríguez-Patón, Membrane computing and complexity theory: A characterization of PSPACE, Journal of Computer and System Sciences 73 (2007) 137–152. URL: http://dx.doi.org/10.1016/j.jcss.2006.10.001.

[14] C. Zandron, C. Ferretti, G. Mauri, Solving NP-complete problems using P systems with active membranes, in: I. Antoniou, C.S. Calude, M.J. Dinneen (Eds.), Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, Springer, 2001, pp. 289–301. URL: http://dx.doi.org/10.1007/978-1-4471-0313-4_21.