Subroutines in P systems and closure properties of their complexity classes

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
{leporati,luca.manzoni,mauri,porreca,zandron}@disco.unimib.it

Abstract

The literature on membrane computing describes several variants of P systems whose complexity classes $\boldsymbol{C}$ are "closed under exponentiation", that is, they satisfy the inclusion $\mathbf{P}^{\boldsymbol{C}} \subseteq \boldsymbol{C}$, where $\mathbf{P}^{\boldsymbol{C}}$ is the class of problems solved by polynomial-time Turing machines with oracles for problems in $\boldsymbol{C}$. This closure automatically implies closure under many other operations, such as regular operations (union, concatenation, Kleene star), intersection, complement, and polynomial-time mappings, which are inherited from $\mathbf{P}$. Such results are typically proved by showing how elements of a family $\boldsymbol{\Pi}$ of P systems can be embedded into P systems simulating Turing machines, which exploit the elements of $\boldsymbol{\Pi}$ as subroutines. Here we focus on the latter construction, providing a description that, by abstracting from the technical details which depend on the specific variant of P system, describes a general strategy for proving closure under exponentiation. We also provide an example implementation using polarizationless P systems with active membranes and minimal cooperation.

Keywords: membrane computing, closure under exponentiation, oracle machines

1. Introduction

Complexity classes of the form $\mathbf{P}^{\boldsymbol{C}}$, whose elements are languages recognizable by polynomial-time Turing machines with oracles for languages in $\boldsymbol{C}$ [19], automatically inherit from $\mathbf{P}$ many closure properties. For instance, the determinism of Turing machines characterising $\mathbf{P}$ implies closure under complement, simply by switching the accepting and rejecting states of the machine. Under certain assumptions on $\boldsymbol{C}$, further closure properties are satisfied.

Let us say that a "reasonable" k-ary operation on languages is a function $f : \left(2^{\Sigma^*}\right)^k \to 2^{\Sigma^*}$ such that $f(L_1, \ldots, L_k)$ is in $\mathbf{P}^{L_1,\ldots,L_k}$, that is, it can be recognized efficiently with oracles for the given k languages. A class $\boldsymbol{C}$ is closed under reasonable operations if $f(L_1, \ldots, L_k) \in \boldsymbol{C}$ for each reasonable operation f and for each $L_1, \ldots, L_k \in \boldsymbol{C}$. The reasonable operations include all usual set theoretic ones (complement, union, intersection and all derived operations) and common language theoretic operations, such as the regular ones (concatenation, union, and Kleene star) [9].

For instance, a class $\boldsymbol{C}$ of the form $\mathbf{P}^{\boldsymbol{D}}$ is closed under reasonable operations whenever $\boldsymbol{D}$ is an *upward directed set*, that is, for each $L_1, L_2 \in \boldsymbol{D}$ there exists $L \in \boldsymbol{D}$ such that $L_1 \leq L$ and $L_2 \leq L$, where $\leq$ denotes polynomial-time reducibility; an oracle for L can thus answer queries for both L_1 and L_2 after a polynomial-time reduction. Any class $\boldsymbol{D}$ with a complete problem L is upward directed, since all languages in $\boldsymbol{D}$ can be reduced to L. This shows that classes such as $\mathbf{P}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{coNP}}$ and $\mathbf{P}^{\mathbf{PP}}$ (which coincides with $\mathbf{P}^{\#\mathbf{P}}$) are closed under reasonable operations. Furthermore, it clearly suffices to find a subset $\boldsymbol{E} \subseteq \boldsymbol{D}$ with $\mathbf{P}^{\boldsymbol{E}} = \mathbf{P}^{\boldsymbol{D}}$ and prove $\mathbf{P}^{\boldsymbol{E}}$ closed under reasonable operations to obtain the same result for $\mathbf{P}^{\boldsymbol{D}}$. This is the case for $\boldsymbol{D} = \mathbf{NP} \cup \mathbf{coNP}$ (and $\boldsymbol{E} = \mathbf{NP}$), a class that frequently appears

Email addresses: leporati@disco.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)

in the membrane computing literature [8, 21, 23, 31, 30] and is not known to be itself upward directed. In fact this would imply $\mathbf{NP} = \mathbf{coNP}$, since there would be a language $L \in \mathbf{NP} \cup \mathbf{coNP}$ such that $L_1 \leq L$ for an **NP**-complete language L_1 and $L_2 \leq L$ for a **coNP**-complete language L_2. Other classes $\boldsymbol{C}$ trivially closed under reasonable operations are those satisfying $\mathbf{P}^{\boldsymbol{C}} = \boldsymbol{C}$, such as **PH**, the polynomial hierarchy [29], and **CH**, the counting hierarchy [33].

Several variants of P systems have been proved able to simulate polynomial-time Turing machines with oracles, exploiting a family $\mathbf{\Pi}$ of recognizer P systems deciding a language L as "subroutines", by embedding them into the membrane structure of a larger P system that provides the inputs and processes the outputs of the members of $\mathbf{\Pi}$ [24, 12, 13, 14, 15]. This implies the closure under exponentiation of the corresponding complexity classes, in symbols $\mathbf{P}^{\mathbf{PMC}_{\mathcal{D}}} \subseteq \mathbf{PMC}_{\mathcal{D}}$, for some class of P systems $\mathcal{D}$. In this paper we describe this subroutine construction in a manner which is as independent from the specific variant of P systems as possible. In particular, as an abstraction from the existing literature, we consider the subroutine simulation for cell-like P systems (Sections 3) and how to adapt the solution to tissue P systems (Section 3.3). For monodirectional cell-like P systems, which have additional constraints, we introduce a new, more general, construction (Section 4). We also provide an implementation that shows how "subroutines" can be constructed using polarizationless P systems with active membranes and cooperative object evolution rules (minimal cooperation, with at most two elements on the left hand side, suffices for the construction).

Many computational complexity results in membrane computing have the form $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{D}}$, that is, some class $\mathcal{D}$ of P systems (e.g., active membranes without charges and dissolution using minimal cooperative rules [30, Corollary 6.6]) can solve in polynomial time all **NP** and **coNP** problems. We argue that this is unlikely to be an exact characterisation, since the features that allow us to solve **NP**-complete problems efficiently are the same that allow the subroutine construction, and this would imply $\mathbf{NP} = \mathbf{coNP}$.

2. Basic Notions

We recall some basic notions regarding the definition of polarizationless P systems with active membranes and cooperative object evolution rules.

Definition 1. A P system with active membranes and cooperative evolution rules of initial degree $d \geq 1$ is a tuple:

$$\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$$

where:

1. Γ is an alphabet, i.e., a finite set of symbols usually called *objects*;
2. Λ is a finite set of labels for the membranes;
3. μ is the initial membrane structure, that is, a rooted unordered tree (usually represented by a set of nested brackets), consisting of d membranes each one labelled with an element of Λ in a one-to-one way;
4. $w_{h_1}, \ldots, w_{h_d}$ with $h_1, \ldots, h_d \in \Lambda$ are finite multisets (sets with multiplicity) of objects in Γ, describing the initial contents of the d regions of μ;
5. R is a finite set of rules.

The rules in R can be of the following types:

- *Cooperative object evolution rules*, of the form $[v \to w]_h$
  They can be applied in membranes with label h and containing v as a sub-multiset. The objects in v are rewritten into the multiset w (i.e., the objects in the multiset v are removed from the content of the membrane and replaced by the objects in w). A rule where $|v| = 2$ is said to employ only *minimal cooperation*.

- *Send-in communication rules*, of the form $a\,[\ ]_h \to [b]_h$
  They can be applied to a membrane labeled with h with an occurrence of the object a located immediately outside. The object a is moved into h while being rewritten as b.

- *Send-out communication rules*, of the form $[a]_h \rightarrow [\ ]_h\ b$
  They can be applied to a membrane labeled with h and containing an occurrence of the object a. The object a is moved to the region directly outside h while being rewritten as b.

- *Elementary division rules*, of the form $[a]_h \rightarrow [b]_h\ [c]_h$
  They can be applied to a membrane labeled with h and containing an occurrence of the object a but not containing any other membrane. The membrane is divided into two membranes with label h; the object a is replaced in the two new membranes, respectively, by b and c, while the rest of the content of the original membrane is duplicated in both membranes.

- *(Weak) non-elementary division rules*, of the form $[a]_h \rightarrow [b]_h\ [c]_h$
  They can be applied to a membrane labeled with h and containing an occurrence of the object a; differently from elementary division rules, other membranes can be contained in h. The membrane is divided as two membranes having label h; the object a is replaced, respectively, by b and c, while the rest of the content (including whole membrane substructures) is duplicated in both membranes.

- *Dissolution rules*, of the form $[a]_h \rightarrow b$
  They can be applied to a membrane labeled with h and containing an occurrence of the object a. The object a is rewritten into b, membrane h is erased and its entire content (including the membrane structures that it contains) is moved into its parent membrane.

The instantaneous configuration of the entire P system Π is represented by a rooted unordered tree where the root denotes the environment; the only subtree directly connected to it represents the entire membrane structure of Π. Each node is labeled by a multiset representing the content of the corresponding membrane, and by a membrane label.

A computation step modifies the current configuration according to the following principles:

1. Each object and membrane can be subject to at most one rule per step, excluding cooperative object evolution rules: inside a membrane, several evolution rules can be applied simultaneously.
2. The application of rules is *maximally parallel*: each object (or multiset of objects) appearing on the left hand side of cooperative object evolution, communication, division, or dissolution rules must be subject to exactly one of them. Analogously, each membrane can be subject to at most one communication, division, or dissolution rule per computation step: these rules will be called *blocking rules* in the rest of the paper. In other words, the only objects and membranes that do not evolve are the ones associated with no rule.
3. When multiple 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.
4. In each computation step all chosen rules are applied simultaneously, in an atomic way. In order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps, where each membrane evolves after its internal configuration (including, recursively, the configuration of the membrane substructures it contains) has been updated. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; in this way, the objects that are duplicated during the division are already the final ones.
5. The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot reenter 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: in this case we assume that all computations are halting, and that either one copy of object yes or one of object no 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*. 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 [22]. Most of the existing literature deals with confluent P systems.

In order to solve decision problems (or, equivalently, decide languages), we use *families* of recogniser P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with 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 [18].

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. Π_n is also required to have 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 included in 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 [18] for further details on the encoding of P systems.

In this paper the constructions will make use only of cooperative object evolution and communication rules. Division and dissolution rules can be used by the P system employed as a subroutine and are, in fact, necessary to go beyond **P** in polynomial time (the "Milano theorem" is otherwise applicable [34]). We also recall that a P system is *monodirectional* [13] when objects can move in only one direction, namely from the innermost to the outermost membranes. In the case of P systems with active membranes, this means that send-in communication rules are forbidden.

3. Subroutines in cell-like P systems

Several Turing machine simulations by means of polynomial-time uniform families of P systems have been proposed in the literature; some of these apply to unrestricted Turing machines [26, 2], while others are limited to machines working in logarithmic space [25], polynomial time [7, 6], polynomial space [32, 24, 17, 10, 12, 13, 16, 14, 15], or exponential space [1]. Most of these solutions [32, 24, 1, 2, 7, 17, 10, 12, 13, 16, 6, 14, 15] are able to simulate Turing machines working in polynomial time with a polynomial slowdown.

The current configuration of the simulated Turing machine can be encoded in several equivalent ways by the simulating P system. A simple, common encoding [12, 13, 16, 14, 15] for a configuration of a polynomial-space Turing machine in state q, having the tape head in position i, and the tape containing the string $x = x_1 \cdots x_m$ is given by the multiset $q_i x_{1,1} \cdots x_{m,m}$, where the object q_i encodes state and head position, and the symbol x_j contained in tape cell j is encoded by the object $x_{j,j}$ of the multiset (i.e., it is subscripted with its position on the tape). The simplest mechanism to simulate a step of the Turing machine is using cooperative rewriting rules; suppose that $\delta(q, a) = (r, b, +1)$ describes the transition of the machine reading symbol a in state q to state r, symbol b and movement to the right. This can be trivially simulated by the cooperative rewriting rule $[q_i\ a_i \to r_{i+1}\ b_i]_M$, which is repeated for each cell position i up to the maximum length of the tape. Notice that *minimal cooperation*, with only two objects on the left-hand side of the rules, suffices for this purpose [31, 30]. All published solutions known to the authors [12, 13, 16, 14, 15] use alternative mechanisms (such as membrane charges, antimatter annihilation, antiport communication) to perform essentially the same operation.

Irrespective of the actual encoding of the configuration of the Turing machine and the mechanism employed to simulate a computation step, we will use the symbol in Figure 1 to denote a membrane structure with root M inside which the simulation is carried over.

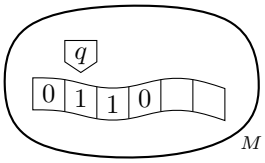

Figure 1: A pictorial notation for the root membrane where the simulation of a Turing machine is carried over.

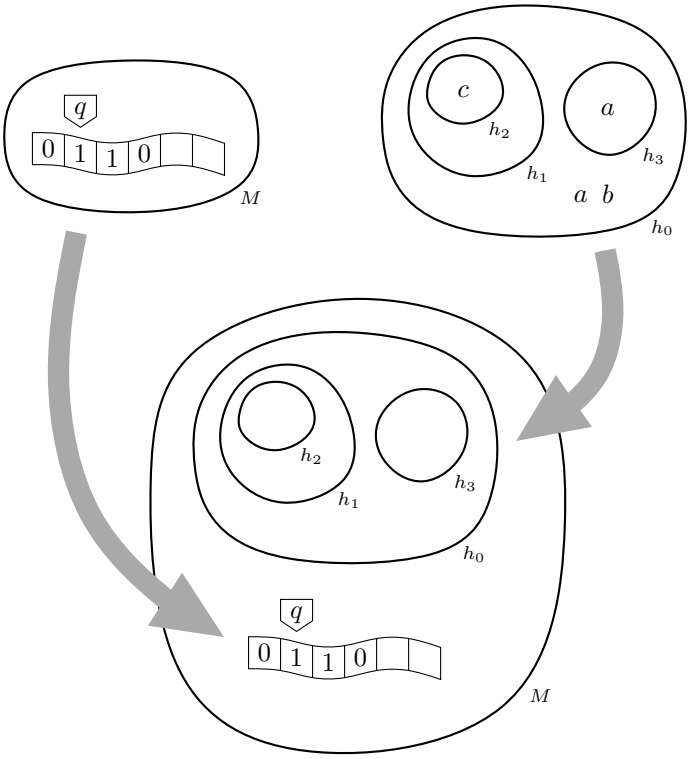

Figure 2: The (empty) membrane structure of a P system is embedded into another P system simulating a Turing machine M.

Let us now consider the case of a Turing machine M with an oracle for language $L \subseteq \Sigma^\star$. Suppose that M writes on its tape the query string $x \in \Sigma^\star$ and enters its query state. Suppose that language L is decided by a family $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of recogniser P systems [22, 18]. By embedding the (empty) membrane structure of the P system Π_x inside the membrane structure of the P system simulating M (Fig. 2) and initialising the configuration of Π_x when the simulation reaches the query state, we can simulate the oracle and read the result of the query as the output of Π_x [24]. It is necessary to send-in the initial multisets contained in Π_x, which typically requires some synchronisation using timers, so that all initial objects appear simultaneously (Fig. 3).

3.1. Implementation in polarizationless P systems with minimal cooperation

In the P system model employed by our examples, the previous construction can be carried out in the following way. We build a uniform family of recognizer P systems $\mathbf{\Pi} = \{\Pi_{M,x,m} : x \in \Sigma^\star, m \in \mathbb{N}\}$ such that the P system $\Pi_{M,x,m}$ simulates a polynomial-time Turing machine M on input x which performs a single query (with a query string of length exactly m) for a language L. We also assume that the query language L is decided by another polynomial-time *uniform* family $\mathbf{\Pi}' = \{\Pi'_y : y \in \Sigma^\star\}$ of recognizer P systems. Let Π'_m be the P system constructed by machine F of the uniformity condition of $\mathbf{\Pi}'$, on input 1^m. Let P be the set of all the paths in the membrane structure of Π'_m starting from the root and ending in one of the leaves, plus all of their suffix subpaths. Notice

that since Π'_m is constructed in polynomial time and its membrane structure is a tree, P is only polynomial in cardinality with respect to m. Each of these paths can be denoted by a sequence $(h_1, \ldots, h_\ell)$ of membrane labels. We employ the notation $p = (h, p')$ to denote a path p having h as the first membrane followed by the subpath p', and ϵ to denote the empty path. Then the membrane structure of $\Pi_{M,x,m} \in \mathbf{\Pi}$ consists of a membrane M containing, as a subtree, the entire membrane structure of Π'_m. The rules of the P system $\Pi_{M,x,m}$ include the rules of Π'_m, together with the rules necessary to simulate the Turing machine M or, more precisely, the variant M' of M described in the following paragraph, as well as the additional rules necessary to perform the interfacing between the Turing machine and Π'_m, as will be detailed in the following.

Let now M' be a Turing machine that behaves the same as M on input x until the query state $q_?$ is reached. M' then simulates the machines E and F of the uniformity condition of the family $\mathbf{\Pi}'$, with the query string y computed by M as input to E, and $1^{|y|} = 1^m$ as input to F. Once the description of Π'_y has been computed, M' extracts from this description all the objects contained in Π'_y; these include not only the multiset computed by E, but all the objects contained in the initial configuration of Π'_y. These objects are written at the beginning of the tape and are then subscripted by a path p, denoting the path from the outermost membrane of Π'_y to the membrane where the object is located in the initial configuration of Π'_y. Notice that these subscripted objects are processed by M' as atomic symbols, i.e., the alphabet of M' includes all of these objects. The Turing machine head moves to the first cell of the tape and enters a new "copy" state q, the simulation of M' stops. The system $\Pi_{M,x,m}$ can then execute rules of the form:

$$[q_i \; a_{p,i} \to q_{i+1} \; a_{p,i,T} \; \sqcup_i]_M \qquad \text{for } 0 \le i < p(m),\ p \in P, \text{ and } a \in \Gamma_m \tag{1}$$

where $p(m)$ is a polynomial bound on the number of objects in the initial configuration of any Π'_y with $|y| = m$, and $T = p(m) + d$, with d the depth of Π'_m, is the time required to move at most $p(m)$ objects from outside the outermost membrane of Π'_m to the innermost one, and Γ_m is the alphabet of Π'_m. These rules remove the symbol a_p from the i-th cell of the Turing Machine which is represented by the object $a_{p,i}$, replacing it with a blank symbol $\sqcup$, and move the tape head one cell to the right by incrementing the subscript of q_i. The object $a_{p,i}$ becomes $a_{p,i,T}$ by acquiring the additional subscript T which represents the maximum number of time steps still needed before having all objects of Π'_y correctly placed.

When the tape head encounters an object that is not part of the description of Π'_y, rules (1) are not applicable anymore and the objects encoding machine M' will remain inert until a result is produced by the embedded copy of Π'_y.

Once an object $a_{p,i,T}$ encoding an object of the initial configuration of Π'_y appears, it begins to move to its correct position in the membrane structure according to the following rules, for each membrane h and a in the set of objects of Π'_y:

$$a_{(h,p),i,t} \; [\;]_h \to [a_{p,i,t-1}]_h \qquad \text{for } p \in P \text{ and } t > i \tag{2}$$

$$[a_{\epsilon,i,t} \to a_{\epsilon,i,t-1}]_h \qquad \text{for } t > i \tag{3}$$

$$[a_{\epsilon,i,i} \to a]_h \tag{4}$$

Rules of type (2) move the object one level deeper in the membrane structure, while removing the first membrane of the path. Rules of type (3), which are enabled when the object is inside its target membrane, decrement the timer until enough time steps have passed in order for all objects to reach their target membrane. When the timer reaches i the subscripts of all the objects are simultaneously removed by rules of type (4), and the computation of Π'_y can begin, since all objects are now in the correct position for its initial configuration.

The embedded P system Π'_y will send out either yes or no in the membrane M after a polynomial number of steps, according to the membership of y in the language L of the family $\mathbf{\Pi}'$. Exactly one of the following rules can then be applied:

$$[q_i \; \mathsf{yes} \to q_{\mathsf{yes},i}]_M \qquad \text{for } 0 \le i < p(m)$$

$$[q_i \; \mathsf{no} \to q_{\mathsf{no},i}]_M \qquad \text{for } 0 \le i < p(m)$$

These rules change the state of the simulated Turing machine M' to reflect the answer to the query that has been computed by Π'_y. Turing machine M' will continue to simulate M until an accepting or rejecting state is reached. This is represented by the presence of an object yes_i or no_i:

$$[\mathsf{yes}_i]_M \to [\ ]_M\ \mathsf{yes} \qquad \text{for } 0 \leq i < \ell$$
$$[\mathsf{no}_i]_M \to [\ ]_M\ \mathsf{no} \qquad \text{for } 0 \leq i < \ell$$

where ℓ is the length of the tape of M'.

The resulting P system $\Pi_{M,x,m}$ is then able to simulate M on input x while performing a single query of length m to an oracle simulated by a P system of the family $\mathbf{\Pi}'$. The construction of $\Pi_{M,x,m}$ can be performed in polynomial time, since it requires to construct the membrane structure, rules, and objects of Π'_m (which, by hypothesis, is a task that can be carried out in polynomial time since Π'_m is a member of a polynomial-time uniform family), and to modify the simulation of M with the rules detailed above, which is also a task attainable within a polynomial time bound.

3.2. Performing multiple queries

Since the query string x is, in general, unknown before the beginning of the simulation, several P systems must be embedded in order to be able to process query strings of different lengths. Suppose that the query language L is decided by a uniform family $\mathbf{\Pi}' = \{\Pi'_y : y \in \Sigma^\star\}$ of P systems, where input strings of the same length m are associated to P systems sharing the same membrane structure and rules of Π'_m, and only differ with respect to the initial multisets they contain. Then, multiple empty membrane structures $\Pi'_0, \Pi'_1, \Pi'_2, \ldots$ can be embedded, up to the maximum possible query string length (an upper bound is given, for instance, by the length of the tape of M), and the correct one is selected at runtime by the P system simulating M (Fig. 4).

When the simulated Turing machine performs multiple queries with query strings of the same length during its computation, it is possible to embed multiple copies of each P system Π'_m [24]. Each of these copies can then be used for a single query (Fig. 5). A possible alternative is to reset the configuration of P system Π'_m after the query simulation has been performed [12].

3.3. Subroutines in tissue P systems

The oracle query construction for cell-like P systems embeds elements of a family of P systems into a membrane where a Turing machine is simulated. This construction can be adapted to tissue P systems by having the Turing machine simulation take place in a cell, and placing the elements of the family of tissue P systems deciding the oracle language on the side [14]. The communication needed in order to simulate the oracle queries in this variant is not hierarchical, but between adjacent cells (Fig. 6).

4. Subroutines in monodirectional cell-like P systems

In monodirectional cell-like P systems [13, 15], where send-in rules are disallowed, the construction of Section 3 does not apply. However, we can turn the construction of Fig. 2 inside out, by having the Turing machine simulation embedded *into the input membrane* of the P system Π simulating the oracle (Fig. 7). Instead of sending in the encoding of the query string, it is the encoding of the configuration of the Turing machine that is sent *out*, through the whole membrane structure of Π, where it waits for the oracle query result (Fig. 8) [13, 15]. This is needed because the P system simulating the oracle can only send its result outwards and, since the system is monodirectional, the only way to intercept it is to move out the entire simulation of the Turing machine.

Unlike the bidirectional case, the objects of the initial configuration of Π cannot be arranged during the query simulation; indeed, if the membrane structure of Π were not linear, some regions would only be reachable from the internal membrane where the simulation of the Turing machine is taking place by means of send-in rules. A solution is to have them already in their correct position in the initial configuration of the combined P system but with a timer subscript, which is deleted at a predefined time step t, when a query may take place. If the simulated Turing machine does not perform a query at time t, it can be adapted so that it makes a "dummy" query at that time, ignoring its result.

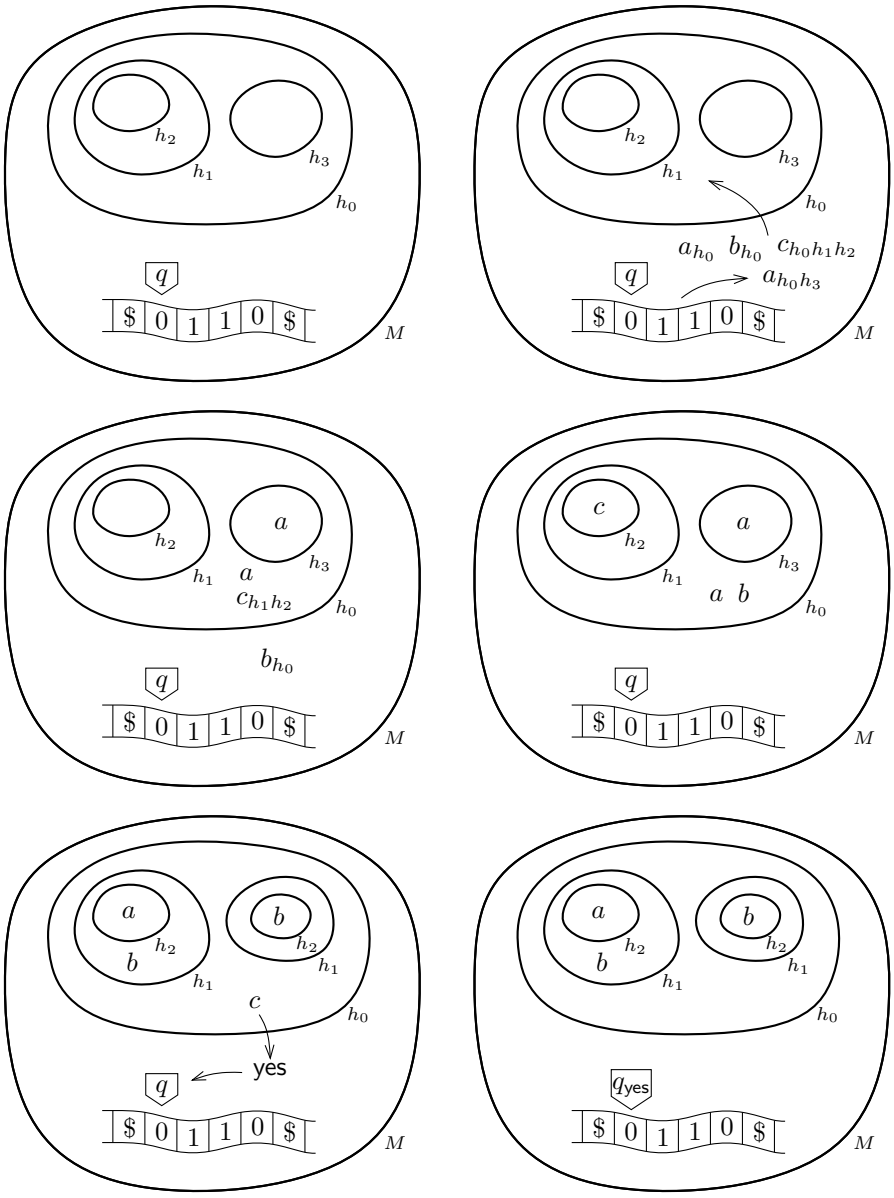

Figure 3: The query string x on the tape of the simulated Turing machine is encoded into the initial configuration of the embedded P system Π_x, together with the input-independent objects that were removed when embedding the P system into the external membrane. Each object is subscripted by the path to its target membrane, which it reaches by using send-in rules, and a timer (as shown in rule (1)). The picture only shows the path subscript for simplicity. When all objects have reached their destination, they simultaneously lose the subscripts, and the computation of Π_x begins. The output of Π_x is then read and incorporated into the state of the simulated Turing machine, as in the configuration following the query.

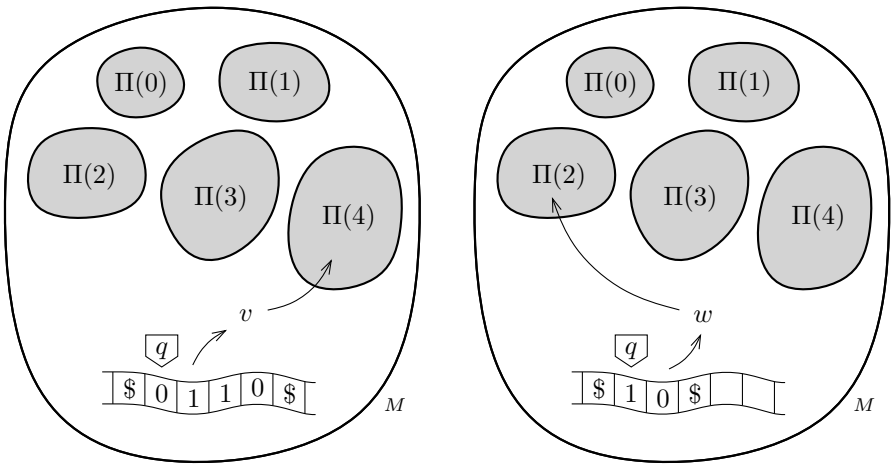

Figure 4: Query strings of different lengths can be processed by distinct embedded P systems, selected when the oracle query is simulated.

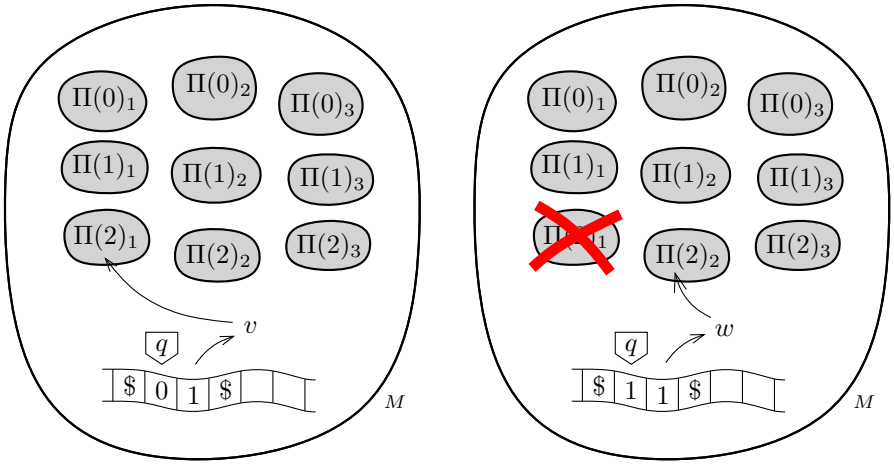

Figure 5: Multiple query strings of the same length can be processed by replicating the P systems simulating the oracle, and using each of them only once.

4.1. Implementation in monodirectional polarizationless P systems with minimal cooperation

As in the bidirectional case, we are going to build a P system $\Pi_{M,x,m}$ that simulates a polynomial-time Turing machine M on input x asking a query y of length m to an oracle simulated by a P system belonging to a polynomial-time uniform family $\mathbf{\Pi}' = \{\Pi'_y : y \in \Sigma^\star\}$. Let Π'_m be the P system obtained by running the machine F of the uniform family $\mathbf{\Pi}'$ on input m, and let h_{in} be its input membrane. Furthermore, let $T = p(n, d)$, where p is a polynomial upper bound depending on the runtime of M on input x, the depth d of h_{in} in the membrane structure of Π'_m, and the runtime of machine E which provide the uniformity for the family $\mathbf{\Pi}$. The value T will be an upper bound to the runtime of the machine M' that will be described in the following.

The structure of $\Pi_{M,x,m}$ consists of a membrane M containing the entire membrane structure of Π'_m. All objects in the description provided by the machine F of the uniform family $\mathbf{\Pi}'$ on input m are already located in the correct membranes "primed" and with T as an additional subscript, which acts as a timer counting down to zero using the rules:

$$[a'_t \to a'_{t-1}]_h \qquad \text{for } 0 < t \leq T, \, h \text{ a membrane label of } \Pi'_m$$

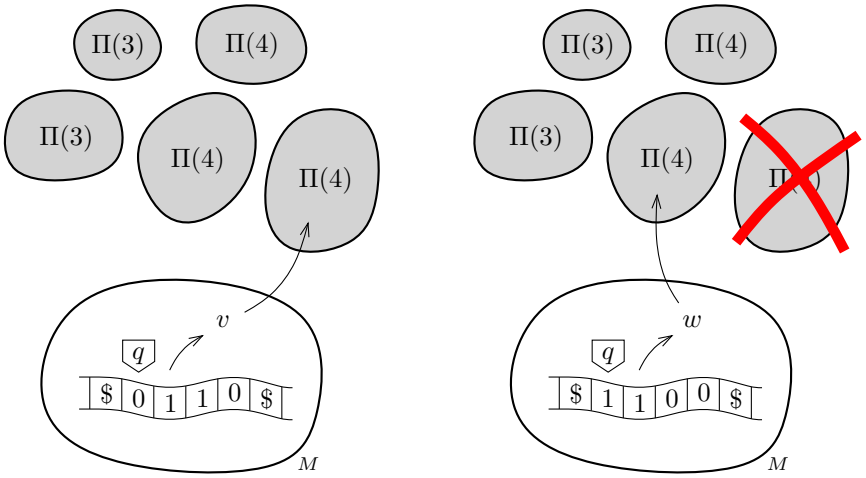

Figure 6: Simulating oracle queries in tissue P systems.

and then, when 0 is reached, the subscript is removed, thus obtaining the objects of the initial configuration of Π'_m:

$$[a'_0 \to a]_h$$

The P system $\Pi_{M,x,m}$ then simulates a Turing machine M' whose construction depends on both M and Π'_m. This machine will start by simulating M on input x in membrane h_{in}. This time the object encoding state and head position also keep track of the value of the timer, thus the initial configuration will contain $q_{0,T}$ where q is the initial state of M. The rules to simulate a Turing machine are modified in order to also decrement this timer:

$$[q_{i,t}\ a_i \to r_{i+d,t-1}\ b_i]_{h_{in}} \qquad \text{for } 0 \le i < T \text{ and } \delta(q,a) = (r,b,d)$$

Once the query state $q_?$ is reached, the simulation of M stops, and M' converts the query string on the tape to a string representing the input multiset w of Π'_m. This can be performed by having machine M' simulate the machine E of the uniformity condition of $\boldsymbol{\Pi}'$ after reaching the state $q_?$ in the simulation of M. After that, the entire tape of M is sent out to membrane M. Once the simulation of E has been completed, machine M' moves the tape head at the beginning of the tape and enters a new state q. A copy of the input multiset of Π'_m is preserved inside h_{in}, since it is necessary for Π'_m to produce the correct answer to the query:

$$[q_{i,t}\ a_i \to q_{i+1,t-1}\ a'_{i,d_{in},t-1}]_{h_{in}} \qquad \text{for } 0 \le i < T,\ t > 0, \text{ and } a \notin \Gamma'_m$$
$$[q_{i,t}\ a_i \to q_{i+1,t-1}\ a'_{t-1}\ a'_{i,d_{in},t-1}]_{h_{in}} \qquad \text{for } 0 \le i < T,\ t > 0, \text{ and } a \in \Gamma'_m$$

where d_{in} is the depth of the membrane h_{in}, i.e., the number of repeated send-out that an object must perform to reach membrane M, and Γ'_m is the alphabet of Π'_m. When the tape head has reached the end of the tape, say of length ℓ, the object representing the state is rewritten as follows:

$$[q_{\ell,t} \to q'_{?,d_{in},t-1}]_{h_{in}} \qquad \text{for } t > 0$$

The process of sending out the tape of M', and the state $q_?$ is performed by the following rules:

$$[a'_{d,t}]_h \to [\ ]_h\ a'_{d-1,t-1} \qquad \text{for } 0 < t < T,\ h \in \Lambda_m, \text{ and } 0 < d \le d_{in}$$
$$[q'_{?,d,t}]_h \to [\ ]_h\ q'_{?,d-1,t-1} \qquad \text{for } 0 < t < T,\ h \in \Lambda_m, \text{ and } 0 < d \le d_{in}$$

where Λ_m is the set of labels of the membranes of Π'_m. Once membrane M has been reached, all objects have their subscript removed:

$$[a'_{0,t} \to a]_M \qquad \text{for } 0 \le t \le T$$
$$[q'_{?,0,t} \to q_?]_M \qquad \text{for } 0 \le t \le T$$

Notice that the object $q_?$ remains inert until either yes or no is sent out from the embedded P system Π'_m. When Π'_m produces an output, it is incorporated in the state of machine M':

$$[\,\mathsf{yes}\; q_? \to q_{\mathsf{yes},0}\,]_M \qquad\qquad [\,\mathsf{no}\; q_? \to q_{\mathsf{no},0}\,]_M \tag{5}$$

Machine M' then continues the simulation of M. Finally, when M either accepts or rejects, the corresponding object is sent out into the environment:

$$\begin{aligned} [\,\mathsf{yes}_i\,]_M &\to [\;]_M\; \mathsf{yes} && \text{for } 0 \le i < \ell \\ [\,\mathsf{no}_i\,]_M &\to [\;]_M\; \mathsf{no} && \text{for } 0 \le i < \ell \end{aligned}$$

Notice that the entire construction can be carried out in polynomial time with respect to the description of M, x, and m. The runtime of $\Pi_{M,x,m}$ is also polynomial, since it requires a number of time steps that is linear in the sum of the execution time of M on input x and of a P system Π'_m whose execution time is polynomial with respect to m.

4.2. Performing multiple queries

As in the bidirectional case, the oracle strings are usually only available at runtime, and thus multiple P systems simulating oracles must be arranged in advance, in order to accommodate any possible query string. Given the restriction on the direction of communication, a natural solution is to *nest* these auxiliary P systems, placing each one inside the input membrane of the next one. When a query takes place, the Turing machine configuration is first moved to the first P system suitable for the query string, where the multiset encoding it is left; then, the Turing machine configuration is moved outside the whole set of auxiliary P systems, where it waits for the result of the query (Fig. 8).

If multiple queries are carried out, it suffices to repeatedly nest the array of auxiliary P systems, always using the input membranes as junction points, and repeating the query procedure as many times as necessary (Fig. 9) [13, 15].

5. Discussion and open problems

Many complexity theory results for P systems have the form $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{D}}$ for some class $\mathcal{D}$ of P systems, and by closure under complementation this implies $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{PMC}_{\mathcal{D}}$ [22]. Solving **NP**-complete problems usually requires some form of "context-sensitivity", such as cooperative evolution rules (even minimal cooperation), membranes charges, membrane dissolution, or antimatter annihilation [31, 30, 20, 4, 5]. However, these same features typically suffice to carry out the oracle simulation constructions described in Section 3 or 4, which implies that $\mathbf{PMC}_{\mathcal{D}}$ has the form $\mathbf{P}^{\boldsymbol{C}}$ for some class $\boldsymbol{C}$. This means that it is quite unlikely that $\mathbf{NP} \cup \mathbf{coNP}$ is an exact characterisation of $\mathbf{PMC}_{\mathcal{D}}$.

Indeed, if $\mathbf{NP} \cup \mathbf{coNP}$ were of the form $\mathbf{P}^{\boldsymbol{C}}$, then it would be $\mathbf{P}^{\mathbf{NP}\cup\mathbf{coNP}} = \mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}}$. But $\mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}} = \mathbf{P}^{\boldsymbol{C}}$: if $L_1 \in \mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}}$, there exists a Turing machine M_1 with oracle for $L_2 \in \mathbf{P}^{\boldsymbol{C}}$ such that $L(M_1) = L_1$, and a Turing machine M_2 with an oracle for $L_3 \in \boldsymbol{C}$ such that $L_2 = L(M_2)$. Let M be a Turing machine with oracle for L_3. This machine simulates M_1 until it enters its query state, then it simulates M_2 until it enters its own query state; since M and M_2 have the same oracle, the queries of M_2 can be answered directly. Then $L(M) = L(M_1) = L_1$, and we can conclude that $\mathbf{P}^{\mathbf{P}^{\boldsymbol{C}}} = \mathbf{P}^{\boldsymbol{C}}$. But then $\mathbf{P}^{\mathbf{NP}\cup\mathbf{coNP}} = \mathbf{P}^{\boldsymbol{C}} = \mathbf{NP} \cup \mathbf{coNP}$. Since $\mathbf{P}^{\mathbf{NP}\cup\mathbf{coNP}} = \mathbf{P}^{\mathbf{NP}}$, this class has complete problems, for instance the standard complete problem H of deciding if a Turing machine M with **NP** oracle accepts a string x within t steps (where t is given in unary notation). Then either $H \in \mathbf{NP}$ or $H \in \mathbf{coNP}$, and it is hard for both **NP** and **coNP**; this means that either there exists a **NP**-hard problem in **coNP**, or a **coNP**-hard problem in **NP**: in both cases, this implies $\mathbf{NP} = \mathbf{coNP}$.

Furthermore, it is often the case [24, 12, 15, 14] that the amount of context-sensitivity that allows us to simulate Turing machines with oracles also suffices, at least in the bidirectional case, to simulate not only oracles for decision problems, but their counting version, which is usually more powerful. For instance, several variants of P systems without non-elementary membrane division rules characterise the complexity class $\mathbf{P}^{\mathbf{PP}}$ (which coincides with $\mathbf{P}^{\#\mathbf{P}}$) [19, 11, 15, 14].

Finally, notice that the above remarks apply even to variants of P systems powerful enough to characterise **PSPACE** in polynomial time [27, 3, 28, 4], even if the algorithms developed in order to prove these results

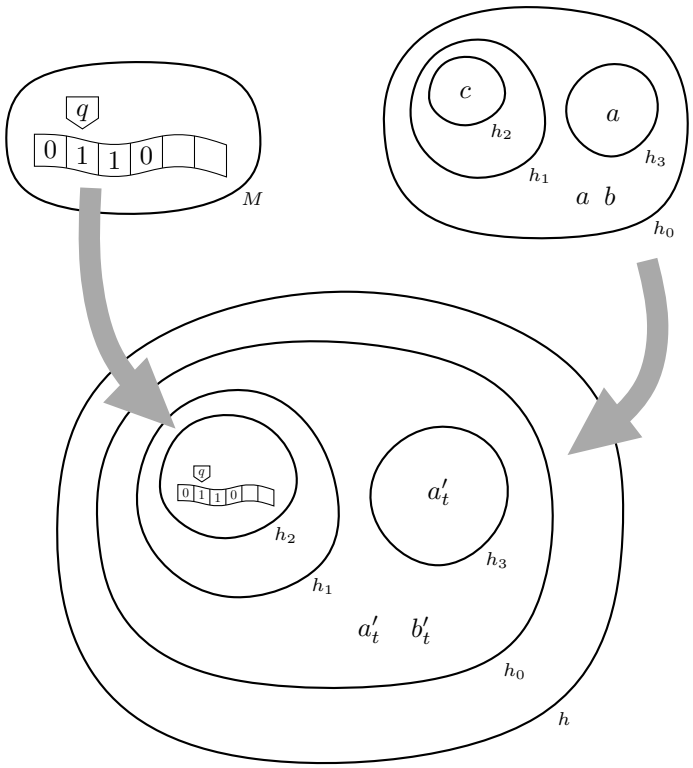

Figure 7: In the monodirectional case, the P system simulating the Turing machine is embedded into the input membrane of the P system Π simulating the oracle. The other objects in the initial configuration of Π that are not the input multiset (c in this case) have an associated timer for synchronisation purposes.

do not usually employ an oracle simulation construction. Indeed, the class **PSPACE** is closed under exponentiation: $\mathbf{P}^{\mathbf{PSPACE}} = \mathbf{PSPACE}$.

The property investigated in this paper, closure under exponentiation, that is, satisfying $\mathbf{P}^{\mathcal{C}} = \mathcal{C}$, seems to apply to a wide range of variants of P systems. A natural follow-up question is whether the more common *closure under oracles* (or *under subroutines*), where $\mathcal{C} = \mathcal{C}^{\mathcal{C}}$, does also apply for some of these variants. This also requires establishing a notion of "P system with oracle"; a first definition has been already given in [12], albeit for purely technical reasons.

An interesting open problem, although a presumably challenging one from a technical standpoint, is to formally characterise the membrane computing features (such as combination of rules or properties of the membrane structures) that enable the oracle simulation construction. Here, minimal cooperation has been shown to suffice. Since also antimatter [15] and charges [12] suffice, it still remains unknown what kinds of cooperation are strictly necessary.

Acknowledgments

This work was partially supported by Fondo d'Ateneo 2016 of Università degli Studi di Milano-Bicocca, project 2016-ATE-0492 "Sistemi a membrane: classi di complessità spaziale e temporale" and by the "Premio giovani talenti 2017" of Università degli Studi di Milano-Bicocca and Accademia dei Lincei.

Bibliography

[1] Artiom Alhazov, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. The computational power of exponential-space P systems with active membranes. In Miguel Ángel Martínez-del-Amor, Gheorghe Păun, Ignacio Pérez-Hurtado, and Francisco José

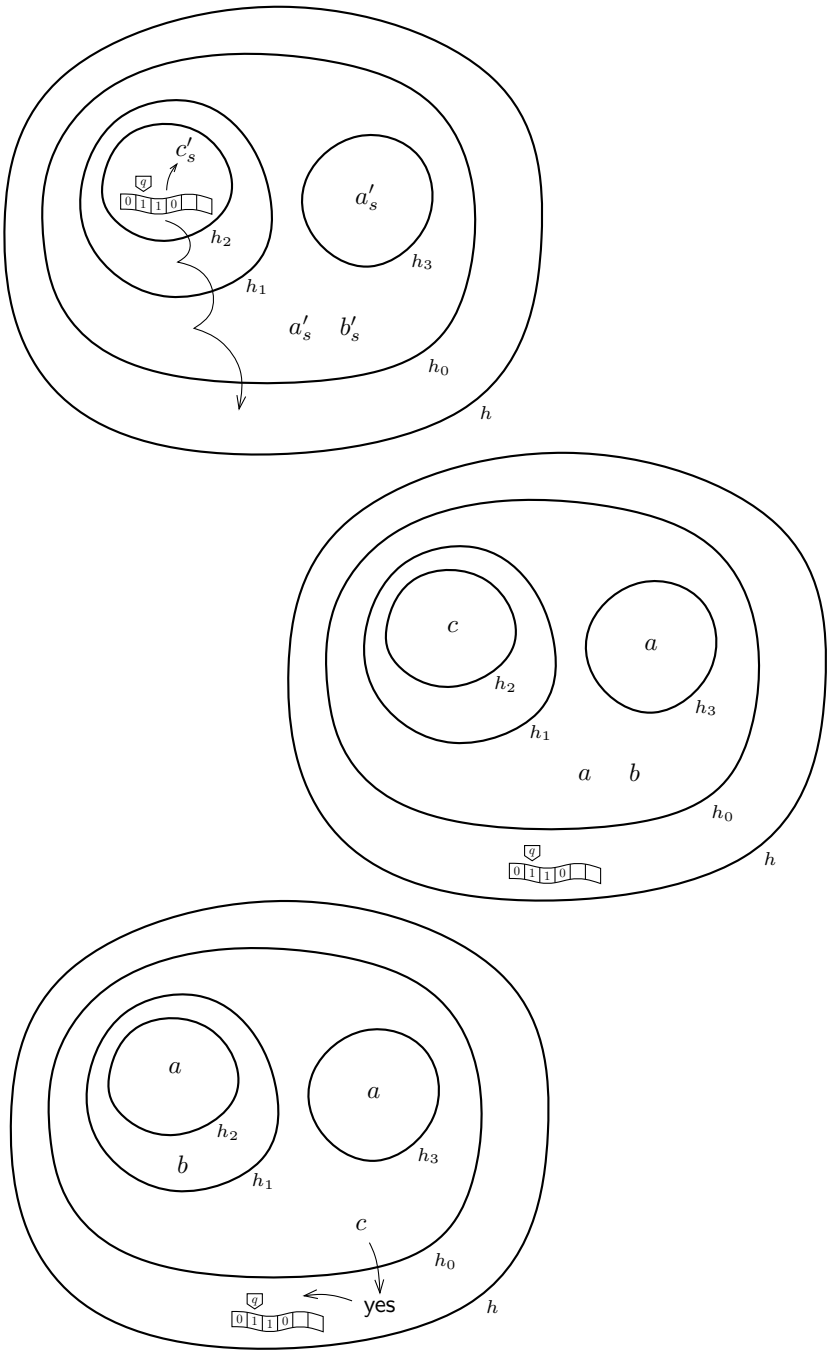

Figure 8: Query simulation procedure for monodirectional cell-like P systems.

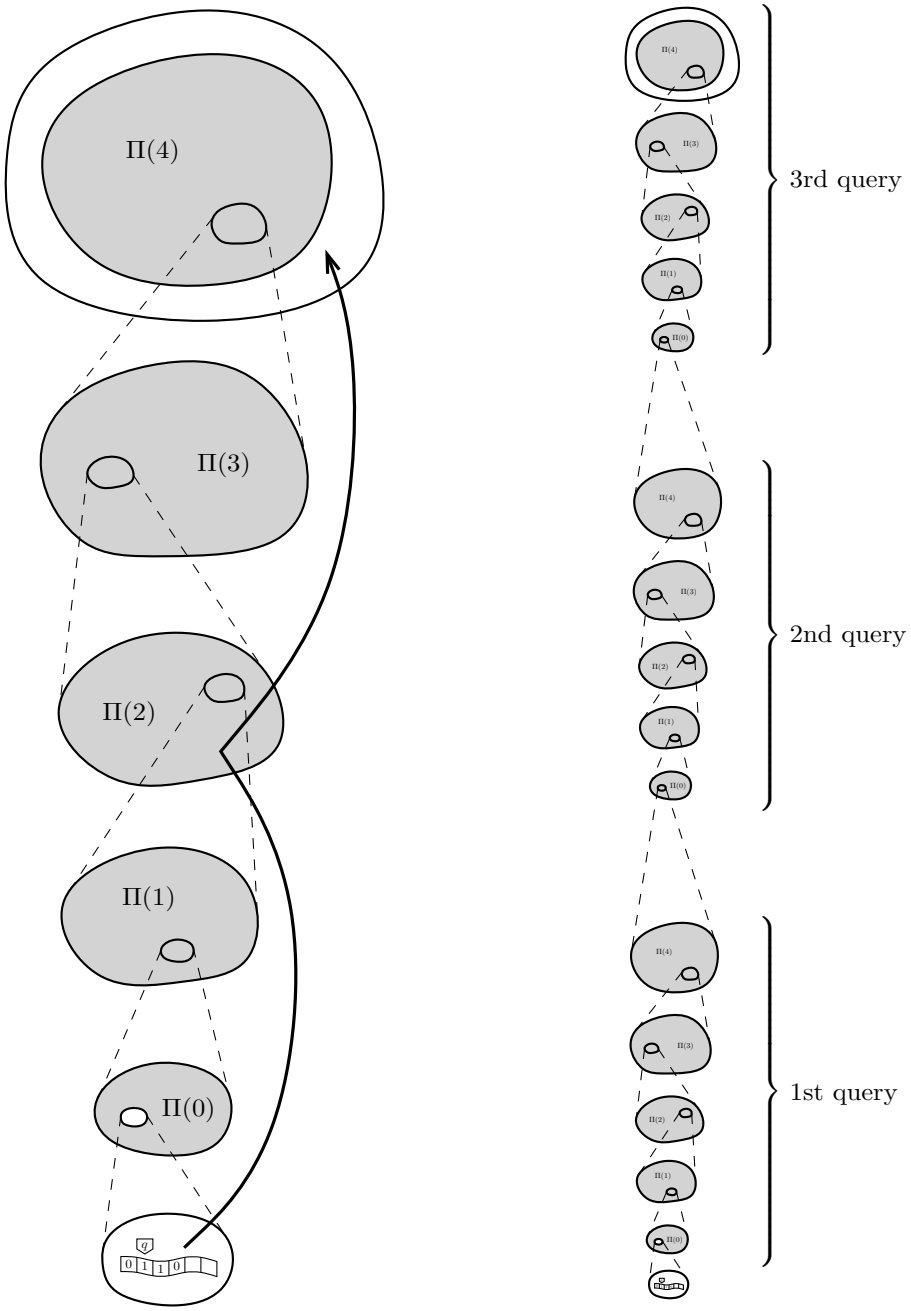

Figure 9: Repeated nesting of monodirectional cell-like P systems to perform queries of a priori unknown length (on the left) and for multiple queries of a priori unknown length (on the right).

Romero-Campero, editors, *Tenth Brainstorming Week on Membrane Computing, Volume I*, number 1/2012 in RGNC Reports, pages 35–60. Fénix Editora, 2012.

[2] Artiom Alhazov, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. Space complexity equivalence of P systems with active membranes and Turing machines. *Theoretical Computer Science*, 529:69–81, 2014.

[3] Artiom Alhazov, Carlos Martín-Vide, and Linqiang Pan. Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae*, 58(2):67–77, 2003.

[4] Artiom Alhazov and Mario J. Pérez-Jiménez. Uniform solution to QSAT using polarizationless active membranes. In Jérôme Durand-Lose and Maurice Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007*, volume 4664 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2007.

[5] Daniel Díaz-Pernil, Artiom Alhazov, Rudolf Freund, Miguel A. Gutiérrez-Naranjo, and Alberto Leporati. Recognizer P systems with antimatter. *Romanian Journal of Information Science and Technology*, 18(3):201–217, 2015.

[6] Zsolt Gazdag and Gábor Kolonits. Remarks on the computational power of some restricted variants of P systems with active membranes. In Alberto Leporati, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing, 17th International Conference, CMC 2016*, volume 10105 of *Lecture Notes in Computer Science*, pages 209–232. Springer, 2017.

[7] Zsolt Gazdag, Gábor Kolonits, and Miguel A. Gutiérrez-Naranjo. Simulating Turing machines with polarizationless P systems with active membranes. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing, 15th International Conference, CMC 2014*, volume 8961 of *Lecture Notes in Computer Science*, pages 229–240, 2014.

[8] Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Francisco J. Romero-Campero. A uniform solution to SAT using membrane creation. *Theoretical Computer Science*, 371(1–2):54–61, 2007.

[9] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3rd Edition*. Addison-Wesley, 2006.

[10] Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. Constant-space P systems with active membranes. *Fundamenta Informaticae*, 134(1–2):111–128, 2014.

[11] Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. Simulating elementary active membranes, with an application to the P conjecture. In Marian Gheorghe, Grzegorz Rozenberg, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing, 15th International Conference, CMC 2014*, volume 8961 of *Lecture Notes in Computer Science*, pages 284–299. Springer, 2014.

[12] Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. Membrane division, oracles, and the counting hierarchy. *Fundamenta Informaticae*, 138(1–2):97–111, 2015.

[13] Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. Monodirectional P systems. *Natural Computing*, 15(4):551–564, 2016.

[14] Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. Characterising the complexity of tissue P systems with fission rules. *Journal of Computer and System Sciences*, 90:115–128, 2017.

[15] Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. The counting power of P systems with antimatter. *Theoretical Computer Science*, 701:161–173, 2017.

[16] Alberto Leporati, Luca Manzoni, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. Shallow non-confluent P systems. In Alberto Leporati, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing, 17th International Conference, CMC 2016*, volume 10105 of *Lecture Notes in Computer Science*, pages 307–316. Springer, 2017.

[17] Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, and Claudio Zandron. A gap in the space hierarchy of P systems with active membranes. *Journal of Automata, Languages and Combinatorics*, 19(1–4):173–184, 2014.

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

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

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

[21] Mario J. Pérez-Jiménez. The P versus NP problem from the membrane computing view. *European Review*, 22(01):18–33, 2014.

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

[23] Mario J. Pérez-Jiménez and Petr Sosík. An optimal frontier of the efficiency of tissue P systems with cell separation. *Fundamenta Informaticae*, 138(1–2):45–60, 2015.

[24] Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. P systems simulating oracle computations. In Marian Gheorghe, Gheorghe Păun, Arto Salomaa, Grzegorz Rozenberg, and Sergey Verlan, editors, *Membrane Computing, 12th International Conference, CMC 2011*, volume 7184 of *Lecture Notes in Computer Science*, pages 346–358. Springer, 2012.

[25] Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. Sublinear-space P systems with active membranes. In Erzsébet Csuhaj-Varjú, Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, and György Vaszil, editors, *Membrane Computing, 13th International Conference, CMC 2012*, volume 7762 of *Lecture Notes in Computer Science*, pages 342–357. Springer, 2013.

[26] Álvaro Romero-Jiménez and Mario J. Pérez-Jiméz. Simulating Turing machines by P systems with external output. *Fundamenta Informaticae*, 49(1–3):273–287, 2002.

[27] Petr Sosík. The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2(3):287–298, 2003.

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

[29] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[30] Luis Valencia-Cabrera, David Orellana-Martín, Miguel A. Martínez-del-Amor, Agustín Riscos-Núñez, and Mario J. Pérez-Jiménez. Computational efficiency of minimal cooperation and distribution in polarizationless P systems with active membranes. *Fundamenta Informaticae*, 153(1–2):147–172, 2017.

[31] Luis Valencia-Cabrera, David Orellana-Martín, Miguel A. Martínez-del-Amor, Agustín Riscos-Núñez, and Mario J. Pérez-Jiménez. Reaching efficiency through collaboration in membrane systems: Dissolution, polarization and cooperation. *Theoretical Computer Science*, 701:226–234, 2017.

[32] Andrea Valsecchi, Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. An efficient simulation of polynomial-space Turing machines by P systems with active membranes. In Gheorghe Păun, Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing, 10th International Workshop, WMC 2009*, volume 6501 of *Lecture Notes in Computer Science*, pages 461–478. Springer, 2010.

[33] Klaus W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986.

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