Depth-two P systems can simulate Turing machines with **NP** oracles

Alberto Leporati [a,*], Luca Manzoni [b], Giancarlo Mauri [a], Claudio Zandron [a]

[a] *Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336, 20126, Milan, Italy*
[b] *Dipartimento di Matematica e Geoscienze, Università degli Studi di Trieste, Via Alfonso Valerio 12/a, 24127, Trieste, Italy*

ARTICLE INFO

Article history:
Received 8 June 2021
Received in revised form 21 October 2021
Accepted 16 November 2021
Available online 20 November 2021

Keywords:
P systems
Active membranes
NP oracles

ABSTRACT

Among the computational features that determine the computing power of polarizationless P systems with active membranes, the depth of the membrane hierarchy is one of the least explored. It is known that this model of P systems can solve **PSPACE**-complete problems when no constraints are given on the depth of the membrane hierarchy, whereas the complexity class $\mathbf{P}_{\|}^{\#\mathbf{P}}$ is characterized by monodirectional *shallow* P systems with minimal cooperation, whose depth is 1. No similar result is currently known for polarizationless systems without cooperation or other additional features. In this paper we show that these P systems, using a membrane hierarchy of depth 2, are able to solve *at least* all decision problems that are in the complexity class $\mathbf{P}_{\|}^{\mathbf{NP}}$, the class of problems solved in polynomial time by deterministic Turing machines that are given the possibility to make a polynomial number of parallel queries to oracles for **NP** problems.

1. Introduction

Membrane systems, also known as *P systems*, have been introduced by Gh. Păun in 1998 as a framework to define parallel models of computation inspired by the functioning of living cells. In the basic model, a tree-like membrane hierarchy divides the Euclidean space into regions. Each membrane can contain other membranes; the outermost membrane is called the *skin*, and divides the P system from the surrounding *environment*. The leaves of the membrane hierarchy correspond to *elementary* membranes, that is, membranes that do not contain any other membrane. Each membrane delimits a *region*, that contains a (possibly empty) finite *multiset* of objects over a finite alphabet. Every membrane label has an associated set of *rewriting rules*, written following an appropriate formal grammar; these rules are usually applied in the so-called *maximally parallel way* to evolve the multiset of objects located in the region. Each of the objects obtained from the application of the rewriting rules can possibly cross the membrane enclosing the region, thus moving up in the membrane hierarchy, or can cross one of the membranes contained into the region in which it has been produced, moving down towards the bottom of the membrane hierarchy.

Since the introduction of this basic model of computation many variants of P systems have been proposed and studied in the literature, giving rise to the research field known as *membrane computing*. This is currently a very active field of research, where three types of models are extensively studied: *cell-like* P systems [1], based on the tree-like hierarchical structure of the membranes within a living cell; *tissue-like* P systems [2], based on the intercommunication between the cells in

* Corresponding author.
E-mail addresses: alberto.leporati@unimib.it (A. Leporati), lmanzoni@units.it (L. Manzoni), giancarlo.mauri@unimib.it (G. Mauri),

biological tissues, and usually modeled as a graph-like interconnection structure of computing elements, each representing a cell; and *spiking neural* P systems [3], inspired by the electrical impulses (called *spikes*) that neurons emit as information, also represented as networks of cells exchanging information in the form of spike symbols. Two nice and clear introductions to P systems can be found in [4,5], while a comprehensive presentation of the state of knowledge in 2010 is given in *The Oxford Handbook of Membrane Computing* [6]. For the latest developments and an extensive bibliography, we refer the reader to the P systems Web page [7].

Since their birth, P systems have stimulated a relevant number of investigations both on the theoretical side – dealing with their computing power and efficiency in solving computationally difficult problems – and from the point of view of applications – studying how P systems can be applied to simulate several kinds of natural phenomena. Focusing on cell-like P systems, one of the models that have attracted most attention, especially from the theoretical point of view, are P systems *with active membranes* [8]. In this kind of P systems the membranes take a more active role during the computations: not only they contain objects, rules, and other membranes, but each membrane may also have an associated electrical charge, which typically can be negative, positive, or neutral. The electrical charge of a membrane affects the applicability of the rules located in the enclosed region, and also the communication between different regions, that is, the ability of objects to cross the membranes. A feature commonly found in P systems with active membranes is the presence of division or separation rules. These rules, inspired by biological mitosis, duplicate an existing membrane, thus modifying the membrane hierarchy, respectively copying or separating the objects that occur in the parent membrane into the two offspring membranes. The membrane structure can also be modified by dissolution rules: when executed, a dissolution rule destroys the membrane in which it is applied, making all the objects and membranes contained in it "fall out" in the surrounding region.

During the two decades following their introduction, P systems with active membranes have been employed to solve classically intractable problems, like **NP**-complete ones [9] or, more recently, problems in the complexity class $\mathbf{P}^{\#\mathbf{P}}$ and in the entire counting hierarchy [10]. To reach these results, the simulation of Turing machines provides an important building block. In particular, the construction of P systems that simulate Turing machines using as few membranes (or cells) as possible and limiting the depth of the system is a "trick" that allows to nest multiple machines in order to solve problems in large complexity classes. For example, nesting of nondeterministic machines – where the nondeterminism was simulated by membrane division – and a counting mechanism allow to characterize $\mathbf{P}^{\#\mathbf{P}}$, the class of all problems solvable by a deterministic Turing machine with access to a #**P** oracle [10,11]. The same ideas can be applied to tissue P systems, where the different communication topology makes even more important to keep the Turing machine simulations compact [12].

Considering the computational features used by P systems with active membranes, we have that uniform families of this kind of systems, with charges and bidirectional communication (that is, objects can cross membranes in both directions), are able to solve $\mathbf{P}^{\#\mathbf{P}}$-complete problems when only one level of membrane nesting (those kinds of P systems are usually called *shallow systems*) is allowed [13,10], and **PSPACE**-complete problems when this restriction is removed [14]. The presence of simple cooperation rules, like the ones provided by antimatter, where two opposite objects can annihilate each other, allows the systems to reach $\mathbf{P}^{\#\mathbf{P}}$ with a shallow membrane structure, also when the systems have no charges [11]. Even when the communication is severely restricted, as in monodirectional systems [15], where send-in is forbidden, uniform families of monodirectional P systems with active membranes *with charges* characterize $\mathbf{P}^{\mathbf{NP}}$ or, if shallow, the class $\mathbf{P}_{\parallel}^{\mathbf{NP}}$, as shown in [15]. It is interesting to see that this is not the case for monodirectional systems with antimatter: the additional cooperation provided by object annihilation makes it possible to perform a counting operation once (in a destructive way), thus providing families of this kind of systems the ability to solve all problems in $\mathbf{P}^{\#\mathbf{P}}$ where only one oracle query suffices, even with only one level of nesting [11]. Considering monodirectional P systems with minimal cooperation [16–18], in [19] it has been proved that these systems, working in polynomial time, characterize the class $\mathbf{P}_{\parallel}^{\#\mathbf{P}}$ of all decision problems solvable in polynomial time by a deterministic Turing machine with access to a single query to a #**P** oracle. This result shows that charges are not essential for monodirectional shallow P systems, thus helping to understand the roles of depth and the direction of communication for reaching computing power beyond **NP** [20].

In this paper we continue the investigation about the influence of the membrane hierarchy depth on the computational power of P systems with active membranes. In particular, we prove that depth-2 polarizationless P systems with active membranes, using non-cooperative evolution rules, bidirectional communication rules, dissolution rules and weak division rules for non-elementary membranes, working in polynomial time, can solve at least all decision problems in the class $\mathbf{P}_{\parallel}^{\mathbf{NP}}$. Compared with the result obtained in [15], here the P systems are bidirectional but without charges; under these conditions, reaching $\mathbf{P}_{\parallel}^{\mathbf{NP}}$ is obtained by increasing the membrane hierarchy depth from 1 to 2. In general, we will allow to use a polynomial number of different **NP** oracles; however, we will also discuss the cases in which one oracle or even one query suffices. For simplicity, we will prove our result using semi-uniform families of P systems; albeit it is possible to extend it to uniform families, we will not delve into details on how to do it, as this transformation is mostly annoying from a technical point of view and not really interesting from a scientific point of view.

It is important to notice that the results presented here are specific to the case of *polarizationless* P systems. In fact, when polarization is present, even without cooperation all problems in $\mathbf{P}^{\#\mathbf{P}}$ can be solved by shallow systems [13], whereas depth-2 systems can solve at least the problems in $\mathbf{P}^{\#\mathbf{P}^{\#\mathbf{P}}}$, i.e., those residing in the second level of the counting hierarchy [10]. In fact, the result presented here is, as far as the authors know, the first one showing that polarizationless P system of constant depth can go beyond **P**.

The rest of this paper is organized as follows: Section 2 recalls some basic notions and notations which are used in the following. In Section 3 we provide a high-level description on how polynomial-time deterministic Turing machines with **NP**

oracles can be simulated using our model of depth-2 P systems with active membranes. Section 4 shows in particular how a single **NP** query can be simulated, thus allowing to use P systems as oracles, and in Section 5 the main result is stated. Section 6 contains the conclusions, and proposes some directions for future research.

2. Basic notions

In this paper we consider semi-uniform and uniform families of P systems with active membranes without charges, working in polynomial time and using non-cooperative evolution rules, communication (send-in and send-out) rules, dissolution rules and elementary and weak non-elementary division rules. A detailed introduction to this model of computation can be found in *The Oxford Handbook of Membrane Computing* [6], that also contains a more general introduction to P systems, how they operate and the related notions of formal language theory and multiset processing.

Definition 1. A *polarizationless P system with active membranes*, of initial degree $d \geq 1$, is a tuple

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

where:

- Γ is an alphabet, i.e., a finite non-empty set of symbols, usually called *objects*;
- Λ is a finite set of labels;
- μ is a membrane structure (i.e., a rooted *unordered* tree, usually represented by nested brackets) consisting of d membranes labeled 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 multisets (finite sets with multiplicity) of objects in Γ, describing the initial contents of each of the d regions of μ;
- R is a finite set of rules.

The rules in R considered in this paper are of the following types:

(a) *Object evolution rules*, of the form $[a \to w]_h$.
They can be applied inside a membrane labeled by h and containing the object a; this object is rewritten into the multiset w (i.e., the object a is removed from the multiset in h and replaced by the objects in w). Since the left-hand side of the rule contains only one object, rules of this kind are usually called *non-cooperative* evolution rules.

(b) *Send-in communication rules*, of the form $a\,[\,]_h \to [b]_h$.
They can be applied to a membrane labeled by h and such that its parent region, i.e., the one containing it, 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 labeled by h and containing an occurrence of the object a; the object a is sent out from h to the parent region, becoming b.

(d) *Dissolution rules*, of the form $[a]_h \to b$.
They can be applied to any membrane (except the outermost one) labeled by h and containing an occurrence of the object a; the object a is sent out from h to the parent region becoming b, the membrane h ceases to exist and all the other objects it contains are sent into the parent region.

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

(f) *Weak non-elementary division rules*, of the form $[a]_h \to [b]_h\,[c]_h$.
They can be applied to a membrane labeled by h containing an occurrence of the object a, and possibly further membranes; the membrane is divided into two membranes having label h; the object a is replaced in the two offspring membranes, respectively, by the single objects b and c, while the rest of the contents (including whole membrane substructures) is replicated in both.

In what follows we will consider P systems having small constant depth, especially depths 1 and 2. The *depth* of a P system is defined as the depth of its membrane hierarchy, represented as a tree. Thus, P systems consisting of a single membrane have depth 0, P systems whose skin contains only elementary membranes have depth 1, and so on. Also, given a P system Π, for simplicity we will refer to a membrane and all its content (including the whole membrane substructures) as a *sub-system* of Π; the depth of such a sub-system is defined in the obvious way, considering the sub-system as if it were itself a P system.

The *configuration* of a P system at a given time step is described by listing the multisets of objects contained in each membrane. A *computation step* changes the current configuration of the system according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules: inside each membrane, several evolution rules can be applied simultaneously.
- The application of rules is *maximally parallel* [6]. Each object appearing on the left-hand side of *an applicable* evolution, communication, dissolution or division rule must be subject to exactly one copy of them: an object cannot remain idle if there are rules applicable to it. Analogously, each membrane can only be subject to one communication, dissolution, or division rule (types (c)–(f)) per computation step; for this reason, these rules will be called *blocking rules* in the rest of the paper. As a result, the only objects and membranes that do not evolve are those associated with no rule.
- 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 from a certain configuration.
- In each computation step, all the chosen rules are applied simultaneously in an atomic way. However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps as follows. First, all evolution rules are applied inside the elementary membranes, followed by all communication, dissolution and division rules involving the membranes themselves; this process is then repeated on the membranes containing them, and so on towards the outermost membrane, still respecting the restriction that each object can be subject to at most one rule for each computation step. This means, in particular, that each membrane evolves only after its internal configuration has been updated. For instance, before a membrane division occurs, all chosen 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 or dissolved, and any object 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 $\mathcal{C}_k$.

P systems can be used as language *recognizers* by employing two distinguished objects yes and no: we assume that all computations are halting, and that either one copy of object yes or one copy 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 if and only if an accepting computation exists. All P systems in this paper are assumed to be confluent.

In order to solve decision problems (or, equivalently, decide languages), we use *families* of recognizer 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 [21]; usually, one of the two following *uniformity conditions* is imposed.

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

Definition 3. 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 the *input membrane*.

Any explicit encoding of Π_x is allowed as output of the construction, as long as it is at most polynomially shorter than the one where the rules are listed one by one, the membrane structure is represented in such a way that all membranes are listed one by one and their content is encoded in unary. 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 [21] for further details on the encoding of P systems.

3. Structure of the simulation

As stated in the Introduction, the aim of this paper is to prove that semi-uniform families of depth-2 P systems of the kind defined above can simulate the computations of deterministic Turing machines working in polynomial time, and having access to one (or even to a polynomial number) of **NP** oracles. Further, the queries made to the oracle(s) are independent of each other; this means that they can all be prepared in advance, and they can be performed in parallel. Stated shortly, what we are going to prove is that the above defined P systems are able to solve at least all decision problems in the complexity class $\mathbf{P}_{\|}^{\mathbf{NP}}$. Albeit the construction we will present uses semi-uniform families of P systems, in Section 5 we will give some indications on how to extend it to uniform families.

Before going into the details of the proof, let us provide a high-level view on how the simulation works, as a road map to follow so as not to get lost.

Let $x \in \Sigma^\star$ be an input string of length n, and let M be a deterministic Turing machine working in polynomial time $p(n)$ with access to an oracle $\mathcal{O}$ for a problem in **NP**. The task of M is determining whether x is a positive instance of a decision problem in $\mathbf{P}_\parallel^{\mathbf{NP}}$; equivalently, this amounts to determining whether $x \in L$ for a given language $L \in \mathbf{P}_\parallel^{\mathbf{NP}}$, thus looking at the complexity class as a set of languages. To simplify the simulation made by the P system, we start assuming that M can ask at most one query to the oracle $\mathcal{O}$; we will later show how to extend the proof to a polynomial number of queries, either asked to the same oracle or to different oracles. Further, the simulation of M will start just from the moment in which the query is executed: indeed, all the previous computation steps performed by M can be simulated by the deterministic Turing machine that builds the P system, either in the semi-uniform or in the uniform setting.

Since the problem solved by the oracle is in **NP**, there exists a nondeterministic Turing machine N working in polynomial time $t(n)$ (such a number of computation steps is polynomial with respect to the length of the query string and, hence, with respect to n) that has an accepting computation starting with the query string as input if and only if the query has a positive answer, that is, if and only if the query string is in the **NP** language accepted by the oracle.

Without loss of generality, we can also assume that both M and N have a binary alphabet, $\Sigma = \{0, 1\}$. Further, since both machines work in polynomial time, and at most one new tape cell can be visited at each computation step, we can consider only $p(n)$ tape cells for M and $t(n)$ tape cells for N. So, let $x_0, \ldots, x_{p(n)-1}$ be the content of the tape of M at the moment in which the query is asked to $\mathcal{O}$, and let $y_0, \ldots, y_{t(n)-1}$ be the (possibly padded) query string. Moreover, let r be the current state of M, j the position of the tape head of M, and q the initial state of N. Also, assume that the tape head of N is located in position 0, and that the answer to the query will be written in position i on the tape of M.

Since we are working under the polynomial semi-uniformity condition, the deterministic Turing machine that builds the P system Π_x can first simulate M up to the moment of the execution of the oracle query, and then output the description of the P system, whose initial configuration will be the one at the moment in which the oracle query is going to be performed. So doing, the initial membrane structure of Π_x will contain the following membranes:

- The skin membrane, labeled M, where the simulation of the deterministic Turing machine M will take place.
- A collection of membranes labeled (k, ℓ) and $(k, \ell)'$, for $0 \leq k < p(n)$ and $0 \leq \ell \leq p(n)$, contained in M. These membranes will be used to store the content of the tape of the simulated deterministic Turing machine M, at each time step; precisely, membrane (k, ℓ) will represent the content of tape cell k at time step ℓ. Membranes $(k, \ell)'$ will serve the same purpose, and will be used as an auxiliary work space during the simulation of a computation step of the Turing machine M.
- A membrane labeled N, contained in M. This is where the simulation of the nondeterministic Turing machine N, on its own simulating the oracle query, will occur.
- A collection of membranes labeled (k, ℓ) and $(k, \ell)'$ for $0 \leq k < t(n)$ and $0 \leq \ell \leq t(n)$, contained in N. Similarly to what happens for membrane M, these membranes will be used to store the content of the tape of the simulated nondeterministic Turing machine N, at each time step; precisely, membrane (k, ℓ) will represent the content of tape cell k at time step ℓ. Membranes $(k, \ell)'$ will serve the same purpose, and will be used as an auxiliary work space during the simulation of a computation step of machine N.
- $t(n)$ membranes labeled $F_0, \ldots, F_{t(n)-1}$, directly contained in membrane M. These membranes will generate an exponential number of copies of themselves, and will be used at the end of the simulation of the oracle query to reduce the number of copies of yes_N objects – if any – to just one copy. In fact, during the simulation of the oracle query the nondeterministic Turing machine N is simulated, and one copy of the yes_N object is produced for each accepting computation path of the machine; as the presence of multiple copies of this object may disturb the subsequent computation steps, a reduction to a single copy is needed.
- A membrane A_N directly contained in M. This membrane will be used at the end of the execution of the query, to "read" its answer (either yes or no). To be precise, it will be used to check for the presence of one copy of the yes_N object, denoting a positive answer produced by the membrane N simulating the oracle query; if no yes_N object occurs then it will send to membrane M one copy of the object no_N, denoting a negative answer to the query.

The initial content of the membranes will be the following:

- The membrane M contains the object $\bar{r}_{j,0,12t(n)+5}$. This denotes that the simulated Turing machine M is in state r, its tape head is located on the j-th cell of the tape, and these are the values at time instant 0. The subscript $12t(n) + 5$ will be used as a timer, to let the simulation of machine M start as soon as the query answer has been produced. When the timer reaches 0, the bar over r is removed and the simulation of M starts.
- Each of the membranes $(k, 0)$ in M, for $0 \leq k < p(n)$, contains one copy of the object $\bar{x}_{k,12t(n)+5}$. As stated above, $x_0, \ldots, x_{p(n)-1}$ is the content of the tape of the Turing machine M at the moment of the query to the oracle, and membranes $(k, 0)$ represent the content of the tape of M at the time step 0, that is, at the first computation step of M. Just like above, the subscript $12t(n) + 5$ is used as a timer to synchronize the start of the simulation of machine M with the production of the query answer. When this timer reaches 0, the bar over x is removed.

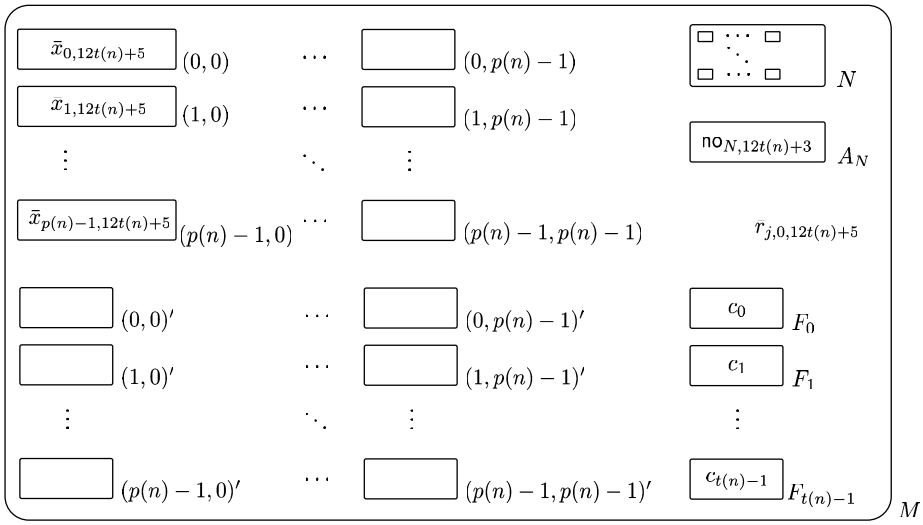

Fig. 1. Structure and initial content of membrane M.

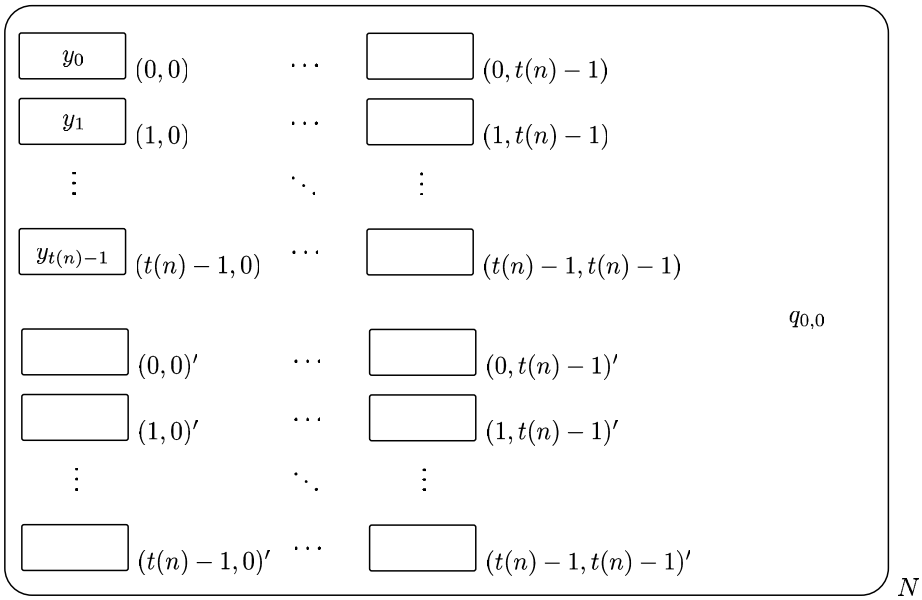

Fig. 2. Structure and initial content of membrane N.

- Each of the membranes F_k contains the object c_k, for $0 \leq k < t(n)$. Such an object will drive the production of an exponential number of copies of membranes F_k, as described above.
- Membrane A_N contains the object $\mathsf{no}_{N,12t(n)+3}$, that represents a negative answer to the query, as described above. The second subscript is used as a counter, to produce the answer no_N to the query (only) in case there are no copies of object yes_N.
- Membrane N contains the object $q_{0,0}$, where q is the initial state of the nondeterministic Turing machine N that simulates the oracle query, and the two subscripts denote respectively the current position of the tape head and the current time step.
- Each of the membranes $(k,0)$ in N, for $0 \leq k < t(n)$, contains one copy of the object y_k. As stated above, $y_0, \ldots, y_{t(n)-1}$ is the (possibly padded) query string, which is located on the tape of the nondeterministic Turing machine N. Hence the objects in membranes $(k,0)$, inside membrane N, represent the content of the tape of machine N at the first time step.
- All the other membranes are empty.

Figs. 1 and 2 show the structure and initial content of membranes M and N, respectively.

At the beginning of the computation, the P system Π_x simulates the execution of the query to oracle $\mathcal{O}$, as will be described in Section 4. This is performed inside membrane N, that simulates the nondeterministic Turing machine N, that on its own simulates the oracle $\mathcal{O}$. The query string is located on the tape of machine N, whose content in the initial configuration is represented by the objects contained in membranes $(k, 0)$, inside membrane N, for $0 \leq k < t(n)$.

Each computation step of the Turing machine N is simulated by Π_x in nine steps. First of all, Π_x has to "read" the symbol (either 0 or 1) contained in the i-th cell of the tape of N. As stated above, the content of the tape of N at time t is represented by the objects in membranes (k, t). The "read" operation is performed in three steps.

First step In the first step, the object $q_{i,t}$ – denoting the fact that the machine N is currently (at time step t) in state q, and the tape head is located on the i-th cell – enters into the membrane (i, t), and stores in its subscript the value 0 as a guess about the symbol contained in the cell.

Second step In the second step, all membranes (k, t) that contain a 0 are dissolved; if membrane (i, t) is among them, then the state/head position object q falls out in membrane N having correctly stored in its subscript the symbol of the i-th cell of the tape of Turing machine N; if instead membrane (i, t) is not dissolved, then the subscript of q is modified to indicate that the symbol contained in the i-th cell of the tape of N is 1.

Third step In the third step, all membranes (k, t) that contain a 1 are dissolved.

Fourth and fifth steps Now that the "read" operation has been performed, in the two subsequent steps all the objects that were inside the membranes (k, t) – both those representing the tape symbols, and the state/head position object – are moved into the correct $(k, t)'$ auxiliary membranes. Also, in the fifth step, all the objects that represent the tape symbols in membranes $(k, t)'$ store in their subscript both their position k on the tape and the time value $t + 1$, thus preparing themselves for the configuration that will represent the next state of the Turing machine N.

Sixth step In the sixth step, the state/head position object dissolves the membrane that contains it, and the object representing the symbol in the i-th cell of the tape is deleted; this is done because the current symbol in the i-th cell of the tape has to be overwritten with a new symbol, as indicated by the transition function of the Turing machine N.

Seventh step In the seventh time step the transition function is applied, and a single object incorporating the result (new state, new symbol to be written in the i-th cell of the tape, and new position of the tape head) is produced. In case the simulated computation step of the Turing machine N is nondeterministic, then a weak non-elementary division of membrane N occurs, and each of the two offspring membranes receives a composite object that corresponds to one of the triples that constitute the result of the transition function. Cases in which the transition function produces more than two triples are not considered, as they can be easily handled through an appropriate sequence of membrane divisions.

Eighth step In the eighth step all membranes $(i, t)'$ are dissolved, and the objects representing the result of the transition function are "unpacked".

Ninth step Finally, in the ninth step all the objects representing the content of the tape of the Turing machine enter into the corresponding membranes $(k, t + 1)$, reflecting the content of the tape at time $t + 1$; the state/head position object is rewritten in a form that allows to start the simulation of the next computation step of the Turing machine N.

The simulation of a computation path of the Turing machine N terminates when a final state is reached. If this is an accepting state, one copy of the object yes_N is produced and sent out to membrane M. As discussed in Section 5, without loss of generality we may assume that all computation paths of N terminate simultaneously. When this happens, membrane M contains one copy of object yes_N for each accepting computation (if any) of machine N. By using a "funnel" mechanism, that involves the creation of an exponential number of copies of membranes F_k, these objects yes_N are transformed to a single copy of object $\mathsf{yes}_{N,-1}$. Such an object interacts with the membrane A_N to finally produce one object that represents the answer to the oracle query, that is, 1_i in case of positive answer, and 0_i in case of negative answer. This object enters the membrane $(i, 0)$ contained into membrane M, simulating the fact that the answer to the query is written in the i-th cell of the tape of the Turing machine M. Now the simulation of the deterministic Turing machine M can start, and it is performed just like the simulation of the nondeterministic Turing machine N, with the difference that the transition function of M never makes nondeterministic choices. The simulation terminates when a final state is reached, at which point a single object (either yes or no) representing the result of the computation is emitted to the environment, during the last computation step.

4. Simulating an NP oracle query

We will now focus on how to perform the simulation of the nondeterministic Turing machine N, on its own simulating the oracle query, inside membrane N. The same construction will also be employed to simulate the deterministic Turing machine M, by neglecting the rules dealing with nondeterministic transitions.

4.1. Simulation of a nondeterministic Turing machine computation step

Let N be a nondeterministic Turing machine, having set of states Q, tape alphabet $\Sigma = \{0, 1\}$, and transition function $\delta : Q \times \Sigma \to 2^{Q \times \Sigma \times \{-1,1\}}$. Without loss of generality we assume that in the simulated machine N each nondeterministic step consists of at most two choices, that is, $|\delta(q, a)| \leq 2$ for all $q \in Q$ and all $a \in \Sigma$.

P system Π_x can simulate in 9 of its steps one computation step of the Turing machine N, as follows. Assume that at time step t the machine N is in state q, with the tape head on the i-th cell; this is encoded by the presence of object $q_{i,t}$ in membrane N. The content of the tape $x_0, x_1, \ldots, x_{t(n)-1} \in \Sigma^{t(n)}$ is encoded by having the object x_i inside the membrane (i, t), for $0 \leq i < t(n)$.

First step In the first step, the object $q_{i,t}$ enters the membrane (i, t) to "read" the symbol in position i on the tape, while the objects representing the content of the tape start to count to (eventually) dissolve the membrane where they are contained:

$$
\begin{aligned}
&q_{i,t}\,[\,]_{(i,t)} \to [q_{i,t}]_{(i,t)} && \text{for } q \in Q, 0 \leq i < t(n), \text{ and } 0 \leq t \leq t(n)\\
&[0 \to 0_0]_{(i,t)} && \text{for } 0 \leq i < t(n) \text{ and } 0 \leq t \leq t(n)\\
&[1 \to 1_1]_{(i,t)} && \text{for } 0 \leq i < t(n) \text{ and } 0 \leq t \leq t(n)
\end{aligned}
$$

Notice that objects 1 take 1 as a subscript, instead of 0. This is because at the second time step each object 0_0 will dissolve the membrane containing it, while at the third step the remaining membranes (i, t) will be dissolved by their objects 1_0. This mechanism allows the P system to know what kind of object has dissolved the membrane, thus simulating the read operation of the symbol contained in the i-th cell of the tape of the Turing machine N.

Second step At the second step of the simulation, the object $q_{i,t}$ representing the state and the tape head position of the Turing machine acquires a new subscript 0 as a guess to indicate that the tape cell i contains the symbol 0. If such a guess is correct, then membrane (i, t) contains the object 0_0; the occurrence of this object dissolves the membrane, thus completing the "read" operation of symbol 0 in the i-th cell. Overall, every membrane (i, t) that contains the object 0_0 is dissolved, and the objects 0_0 that trigger the dissolution save in their subscript the current time step t of machine N and their position on the tape. On the other hand, the dissolution of all membranes (i, t) containing the object 1_1 must be postponed to the next step, as discussed above, hence objects 1_1 are rewritten to 1_0 to wait one step:

$$
\begin{aligned}
&[q_{i,t} \to q_{i,t,0}]_{(i,t)} && \text{for } q \in Q, 0 \leq i < t(n), \text{ and } 0 \leq t \leq t(n)\\
&[0_0]_{(i,t)} \to 0_{i,t} && \text{for } 0 \leq i < t(n) \text{ and } 0 \leq t \leq t(n)\\
&[1_1 \to 1_0]_{(i,t)} && \text{for } 0 \leq i < t(n) \text{ and } 0 \leq t \leq t(n)
\end{aligned}
$$

Third step At the third step of the simulation, all the membranes (i, t) that contain the object 1_0 are dissolved. Also in this case, the objects 1_0 that trigger the dissolution save in their subscript the current time step t of machine N and their position on the tape. Notice that, before membrane (i, t) is dissolved, the object $q_{i,t}$ updates its guess on the content of the i-th cell of the tape, changing to 1 the third component of its subscript. If, instead, membrane (i, t) was dissolved in the previous step, because the i-th cell of the tape of Turing machine N contains the symbol 0, then the object $q_{i,t,0}$ that has fallen out in membrane N is primed to wait one time step. The same treatment goes to all the objects $0_{i,t}$ that have fallen out in membrane N during the previous time step:

$$
\begin{aligned}
&[q_{i,t,0} \to q_{i,t,1}]_{(i,t)} && \text{for } q \in Q, 0 \leq i < t(n), \text{ and } 0 \leq t \leq t(n)\\
&[q_{i,t,0} \to q'_{i,t,0}]_N && \text{for } q \in Q, 0 \leq i < t(n), \text{ and } 0 \leq t \leq t(n)\\
&[0_{i,t} \to 0'_{i,t}]_N && \text{for } 0 \leq i < t(n) \text{ and } 0 \leq t \leq t(n)\\
&[1_0]_{(i,t)} \to 1_{i,t} && \text{for } 0 \leq i < t(n) \text{ and } 0 \leq t \leq t(n)
\end{aligned}
$$

Fourth step At the fourth time step, all the objects representing the tape symbols, that are now in membrane N with a subscript indicating their position on the tape, enter into the auxiliary membranes $(i, t)'$, losing their subscripts. At this point, the collection of membranes $(i, t)'$ represents the content of the tape of machine N, just like the collection of membranes (i, t) represented it before and during the first time step. However, the difference is that now the state/tape head object q has stored in its subscript what is the symbol contained in the i-th cell of the tape. During this time step this object is primed, in order to wait:

$$
\begin{aligned}
&[q_{i,t,1} \to q'_{i,t,1}]_N && \text{for } q \in Q, 0 \leq i < t(n), \text{ and } 0 \leq t \leq t(n)\\
&[q'_{i,t,0} \to q''_{i,t,0}]_N && \text{for } q \in Q, 0 \leq i < t(n), \text{ and } 0 \leq t \leq t(n)
\end{aligned}
$$

$$0'_{i,t}\,[\,]_{(i,t)'} \to [0]_{(i,t)'} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t \le t(n)$$
$$1_{i,t}\,[\,]_{(i,t)'} \to [1]_{(i,t)'} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t \le t(n)$$

Fifth step At the fifth time step, also the state/head position object q enters into the membrane $(i,t)'$ corresponding to the i-th cell of the tape. In the meanwhile, all the objects that entered into membranes $(i,t)'$ during the previous time step prepare themselves to go into membranes $(i,t+1)$, that will represent the content of the tape at the beginning of the next computation step of the Turing machine N:

$$q'_{i,t,1}\,[\,]_{(i,t)'} \to [q_{i,t,1}]_{(i,t)'} \qquad \text{for } q \in Q,\ 0 \le i < t(n), \text{ and } 0 \le t \le t(n)$$
$$q''_{i,t,0}\,[\,]_{(i,t)'} \to [q_{i,t,0}]_{(i,t)'} \qquad \text{for } q \in Q,\ 0 \le i < t(n), \text{ and } 0 \le t \le t(n)$$
$$[0 \to 0_{i,t+1}]_{(i,t)'} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$
$$[1 \to 1_{i,t+1}]_{(i,t)'} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$

Sixth step At the sixth time step, all the objects representing the contents of the tape cells are primed, to wait. Then, the state/head position object q dissolves the membrane $(i,t)'$ that contains it. The other object (either $0'_{i,t+1}$ or $1'_{i,t+1}$) that was contained in the dissolved membrane falls out to membrane N, and will be deleted during the next time step:

$$[q_{i,t,0}]_{(i,t)'} \to \tilde{q}_{i,t,0} \qquad \text{for } q \in Q,\ 0 \le i < t(n), \text{ and } 0 \le t \le t(n)$$
$$[q_{i,t,1}]_{(i,t)'} \to \tilde{q}_{i,t,1} \qquad \text{for } q \in Q,\ 0 \le i < t(n), \text{ and } 0 \le t \le t(n)$$
$$[0_{i,t+1} \to 0'_{i,t+1}]_{(i,t)'} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$
$$[1_{i,t+1} \to 1'_{i,t+1}]_{(i,t)'} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$

Seventh step At the seventh time step, the P system is ready to simulate the application of the transition function δ of the Turing machine N. Recall that we assumed $|\delta(q,a)| \le 2$ for all $q \in Q$ and $a \in \Sigma$; we examine the cases $|\delta(q,a)| = 1$ and $|\delta(q,a)| = 2$ separately. So, suppose that $\delta(q,a) = \{(r,b,d)\}$, where q is the current state, a is the symbol read on the tape, r is the new state, b is the new symbol to be written on the tape, and $d \in \{-1,1\}$ indicates the movement of the tape head. At this time step, the object $\tilde{q}_{i,t,a}$ is rewritten to represent the result of the application of the transition function; notice that the evolution rule produces a *single* object, that will be "unpacked" at the next time step. In the meanwhile, the 0 and 1 objects in membranes $(i,t)'$ continue to wait, while the object (0 or 1) in membrane N is rewritten to the "junk" object #, thus making it inactive. This is made because the content of the i-th cell of the tape has to be replaced with the new symbol b indicated in the output triple of the transition function; the object $b_{i,t+1}$ that represents this new symbol will be produced at the next time step:

$$[\tilde{q}_{i,t,a} \to \langle r_{i+d,t+1}\, b_{i,t+1}\rangle]_N \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$
$$[0'_{i,t+1} \to 0''_{i,t+1}]_{(i,t)'} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$
$$[1'_{i,t+1} \to 1''_{i,t+1}]_{(i,t)'} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$
$$[0'_{i,t+1} \to \#]_N \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$
$$[1'_{i,t+1} \to \#]_N \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$

If the transition function is instead non-deterministic, i.e., of the form $\delta(q,a) = \{(r,b,d_r),(s,c,d_s)\}$, then the same rules as above are applied, but instead of rewriting $\tilde{q}_{i,t,a}$ via an evolution rule, a weak non-elementary division rule is applied on membrane N:

$$[\tilde{q}_{i,t,a}]_N \to [\langle r_{i+d_r,t+1}\, b_{i,t+1}\rangle]_N\,[\langle s_{i+d_s,t+1}\, c_{i,t+1}\rangle]_N \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$

Eighth step Irrespective from the fact that membrane N has been divided or not during the seventh step, at the eighth time step all membranes $(i,t)'$ are dissolved, and the objects representing the result of the transition function δ are "unpacked":

$$[\langle r_{i+d,t+1}\, b_{i,t+1}\rangle \to \hat{r}_{i+d,t+1}\,\hat{b}_{i,t+1}]_N \qquad \text{for } r \in Q,\ b \in \{0,1\},\ d \in \{-1,1\},\ 0 \le i < t(n), \text{ and } 0 \le t < t(n)$$
$$[0''_{i,t+1}]_{(i,t)'} \to \hat{0}_{i,t+1} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$
$$[1''_{i,t+1}]_{(i,t)'} \to \hat{1}_{i,t+1} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$

Ninth step At the ninth step it is only necessary to send all the objects representing the tape content into the corresponding membranes. Simultaneously, the object representing the new state, that in the previous step was rewritten to an auxiliary object for synchronization purposes, is now rewritten to a usable form:

$$[\hat{r}_{i+d,t+1} \to r_{i+d,t+1}]_N \qquad \text{for } r \in Q, d \in \{-1, 1\}, \\ 0 \le i < t(n), \text{ and } 0 \le t < t(n)$$

$$\hat{0}_{i,t+1}\,[\,]_{(i,t+1)} \to [0]_{(i,t+1)} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$

$$\hat{1}_{i,t+1}\,[\,]_{(i,t+1)} \to [1]_{(i,t+1)} \qquad \text{for } 0 \le i < t(n) \text{ and } 0 \le t < t(n)$$

The configuration of the P system now correctly represents the configuration of the nondeterministic Turing machine N after executing its computation step, and it can be used to start the simulation of the next computation step.

When the simulation of the Turing machine N reaches one of the states $q_{i,t(n)}$, for some i, this is a final state by the assumptions we imposed on N. If such a state is accepting then the P system executes the following rule to send out a "yes" answer to the query:

$$[q_{i,t(n)}]_N \to [\,]_N\, \mathsf{yes}_N \qquad \text{for } 0 \le i < t(n) \text{ and } q \text{ accepting}$$

Since the P system Π_x simulates one step of the Turing machine N in nine of its time steps, the slowdown induced by the simulation is only linear: $t(n)$ computation steps of N require $9t(n) + 1$ steps of Π_x. At the end of the simulation, membrane M contains as many copies of the object yes_N as the number of accepting computations of N.

4.2. Reading the query answer

Once the yes_N objects – one for each accepting computation of the nondeterministic Turing machine N – have been sent out in the skin membrane M, it is necessary to actually "read" the answer to the query and communicate it to the Turing machine M. To do so, without loss of generality, we will write either 1 (for a positive answer) or 0 (for a negative one) in the i-th cell of the tape of machine M. This means, in particular, that if multiple yes_N objects are present then exactly one copy of the object $1_{(i,0)}$ must be sent into the membrane with label $(i, 0)$ contained into membrane M. On the other hand, if no object yes_N is present then exactly one copy of object $0_{(i,0)}$ must be sent into the same membrane.

This procedure will be performed with the help of a "funnel", that reduces the amount of objects of type yes_N from a possibly exponential number (at most $2^{t(n)}$, if each step of N was nondeterministic and all computations accepted) to either 1 or 0. This funnel is implemented using 2^k membranes labeled F_k, for $0 \le k < t(n)$, contained in membrane M. While this is an exponential number of membranes, the P system can generate them "at run time" by applying a sequence of division rules to the membranes with labels $F_0, \ldots, F_{t(n)-1}$ that occur in membrane M in the initial configuration of Π_x. Recall that every membrane F_k, for $0 \le k < t(n)$, initially contains one copy of the object c_k. The correct number of membranes F_k is then generated through the following set of division rules for elementary membranes:

$$[c_j]_{F_k} \to [c_{j-1}]_{F_k}\,[c_{j-1}]_{F_k} \qquad \text{for all } 0 < j \le k$$

After at most $t(n) - 1$ time steps, all the membranes with labels of the form F_k have been generated. Since the simulation of the nondeterministic Turing machine N requires $9t(n)+1$ time steps of the P system, the generation of these membranes can be performed in parallel with the simulation of the oracle query.

The main idea behind the operation of the funnel is that if there are at most 2^{k+1} objects of type $\mathsf{yes}_{N,k}$ in membrane M, then at most 2^k of them will be able to enter membranes F_k at a given step, while the remaining objects (at most 2^k) will enter during the next time step. Once an object of type $\mathsf{yes}_{N,k}$ enters a membrane labeled F_k, it counts for one time step before dissolving the membrane and rewriting itself as $\mathsf{yes}_{N,k-1}$. Note that the dissolution is never in conflict with any other blocking rule, since by the time it can be applied all objects of type $\mathsf{yes}_{N,k}$ have been processed by their associated send-in rules. The objects falling out when F_k is dissolved, that were not responsible for the dissolution, are rewritten into junk objects. Since at most 2^k objects were responsible for dissolving the membranes F_k, there are now at most 2^k objects of type $\mathsf{yes}_{N,k-1}$; these allow to perform the same operations as above with the 2^{k-1} membranes labeled F_{k-1}. At the end of the entire process only one object will remain, making it possible to determine the answer to the query.

Going into the details, the objects $\mathsf{yes}_{N,k}$ enter into the corresponding membranes F_k by the following rules:

$$\mathsf{yes}_N\,[\,]_{F_{t(n)-1}} \to [\mathsf{yes}'_{N,t(n)-1}]_{F_{t(n)-1}}$$

$$\mathsf{yes}_{N,k}\,[\,]_{F_k} \to [\mathsf{yes}'_{N,k}]_{F_k} \qquad \text{for } 0 \le k < t(n) - 1$$

Notice that the first rule is only due to the fact that the object yes_N is not of the form $\mathsf{yes}_{N,k}$.

When an object $\mathsf{yes}_{N,k}$ is inside a membrane labeled F_k, it waits for one step before dissolving the membrane:

$$[\mathsf{yes}'_{N,k} \to \mathsf{yes}''_{N,k}]_{F_k} \qquad \text{for } 0 \le k < t(n)$$

$$[\mathsf{yes}''_{N,k}]_{F_k} \to \mathsf{yes}_{N,k-1} \qquad \text{for } 0 \le k < t(n)$$

Any object falling out without dissolving the membrane is then rewritten into a "junk" object:

$$[\mathsf{yes}''_{N,k} \to \#]_M \qquad \text{for } 0 \le k < t(n)$$

At the end of the process, if at least one object of type yes_N was present in the skin membrane, then after $3t(n)$ time steps this membrane will contain exactly one copy of the object $\mathsf{yes}_{N,-1}$.

Since the simulation of the Turing machine N requires $9t(n)+1$ steps before sending out the objects yes_N (one for each accepting computation) and the generation of $\mathsf{yes}_{N,-1}$ requires $3t(n)$ additional steps, after $12t(n)+1$ time steps since the start of the computation there is exactly one object $\mathsf{yes}_{N,-1}$ in membrane M if and only if the answer to the oracle query computed by the nondeterministic Turing machine N is positive.

The P system can now use the membrane labeled A_N to "read" the answer to the query. This membrane contains one copy of the object $\mathsf{no}_{N,12t(n)+3}$, which is rewritten $12t(n)+3$ times before being sent out as 0_i:

$$[\mathsf{no}_{N,t} \to \mathsf{no}_{N,t-1}]_{A_N} \qquad \text{for } 0 \le t < 12t(n)+3$$
$$[\mathsf{no}_{N,0}]_{A_N} \to [\,]_{A_N}\, 0_i$$

In the meantime, if the object $\mathsf{yes}_{N,-1}$ exists, it enters the membrane A_N and dissolves it, waits one step and then it is rewritten as 1_i. The no object fallen out from the dissolved membrane becomes the junk symbol:

$$\mathsf{yes}_{N,-1}\,[\,]_{A_N} \to [\mathsf{yes}_N]_{A_N}$$
$$[\mathsf{yes}_N]_{A_N} \to \mathsf{yes}'_N$$
$$[\mathsf{yes}'_N \to 1_i]_M$$
$$[\mathsf{no}_{N,0} \to \#]_M$$

In this way, at time step $12t(n)+4$ exactly one among the objects 0_i and 1_i occurs in the skin membrane. At the next time step this object can be sent into the membrane $(i,0)$ representing the i-th cell of the tape of the Turing machine M at the beginning of its simulation:

$$0_i\,[\,]_{(i,0)} \to [0]_{(i,0)} \qquad \text{for } 0 \le i < p(n)$$
$$1_i\,[\,]_{(i,0)} \to [1]_{(i,0)} \qquad \text{for } 0 \le i < p(n)$$

Since the simulation of the oracle Turing machine N is based on the careful synchronization of the dissolution steps, the simulation of the Turing machine M must start at time $12t(n)+6$, that is, one step after the answer to the query computed by N has been written on the tape of M. This can be accomplished by adding a timer to the object representing the initial state r and tape head position j of Turing machine M, as well as to the objects representing the content of those cells of the tape of M where there is no answer to the query (or queries, in case of multiple ones, as described below). The membranes $(i,0)$ that will contain the answers to the queries are left unchanged, since their contents will be determined at the end of the simulation of the oracle Turing machines. The corresponding set of rules is as follows:

$$[\bar{r}_{j,0,\ell} \to \bar{r}_{j,0,\ell-1}]_M \qquad \text{for } 0 < \ell \le 12t(n)+5$$
$$[\bar{r}_{j,0,0} \to r_{j,0}]_M$$
$$[\bar{0}_\ell \to \bar{0}_{\ell-1}]_{(i,0)} \qquad \text{for } 0 < \ell \le 12t(n)+5,$$
$$\text{tape cell } i \text{ not containing a query answer}$$
$$[\bar{0}_0 \to 0]_{(i,0)} \qquad \text{for tape cells } i \text{ not containing a query answer}$$
$$[\bar{1}_\ell \to \bar{1}_{\ell-1}]_{(i,0)} \qquad \text{for } 0 < \ell \le 12t(n)+5,$$
$$\text{tape cell } i \text{ not containing a query answer}$$
$$[\bar{1}_0 \to 1]_{(i,0)} \qquad \text{for tape cells } i \text{ not containing a query answer}$$

The simulation of the Turing machine M starts at time step $12t(n)+6$ and is performed just like we have described above for the simulation of the nondeterministic machine N, with the difference that this time there will be no nondeterministic choices, that is, it will be $|\delta(q,a)| = 1$ for all $q \in Q$ and $a \in \Sigma$. The simulation will halt when a final state of machine M is obtained, at which point the acceptance object yes or the rejection object no will be sent-out to the environment, in the last computation step.

5. Solving problems in $\mathbf{P}_{\|}^{\mathbf{NP}}$

The above construction shows how to perform the simulation of a deterministic Turing machine M asking a *single* query to an oracle, in turn simulated by a nondeterministic Turing machine N. It is however possible to extend the construction to allow a polynomial number $q(n)$ of oracle queries, all performed in parallel. To be as general as possible, we can assume that each query is asked to a different **NP** oracle $\mathcal{O}_i$, simulated by a corresponding nondeterministic Turing machine N_i, for $1 \leq i \leq q(n)$. In turn, each Turing machine N_i will be simulated by an associated sub-system N_i located in the skin membrane of Π_x. As a special case there may be a single oracle $\mathcal{O}$, simulated by a nondeterministic Turing machine N, queried $q(n)$ times. However, also in this case the skin membrane of Π_x will contain $q(n)$ copies of the sub-system N, each simulating the execution of one query; this duplication is necessary, as each sub-system dissolves some of its membranes during the computation and thus cannot be reused after the execution of a query.

Another special case stems from the following question: Can the number of oracle queries be reduced to just 1, as it happens with the characterization of the complexity class $\mathbf{P}_{\|}^{\#\mathbf{P}}$ provided in [19]? In particular, Proposition 1 of [19] shows that a polynomial number of parallel #**P** queries can be simulated by a single #**P** query in polynomial time. The idea behind the proof is that the nondeterministic Turing machine that simulates the single #**P** query takes all the query strings as input, concatenated and separated with a new symbol \$, and outputs as a result a number that encodes all the query answers. Unfortunately, the same trick cannot be used when dealing with the complexity class $\mathbf{P}_{\|}^{\mathbf{NP}}$. Indeed, let $\mathcal{O}_1, \ldots, \mathcal{O}_{q(n)}$ be the **NP** oracles, and let $N_1, \ldots, N_{q(n)}$ be the nondeterministic Turing machines simulating them. We can define a nondeterministic Turing machine N that takes as input the string $x_1\$x_2\$\cdots\$x_{q(n)}$, where x_i is a query string for oracle $\mathcal{O}_i$ and, for every $1 \leq i \leq q(n)$, simulates one after the other each nondeterministic Turing machine N_i on input x_i, recording the answers $a_1, \ldots, a_{q(n)}$ on its tape. However, being the simulator of an **NP** oracle, the Turing machine N can only return one bit of information (either yes or no) as its answer, hence it cannot return any string-based or number-based representation of the entire sequence of query answers $a_1, \ldots, a_{q(n)}$. The most general answer that can be returned by the Turing machine N is the result of a Boolean function having the values $a_1, \ldots, a_{q(n)}$ as input. In this scenario, a single sub-system N contained in the skin membrane of Π_x can simulate the nondeterministic Turing machine N, computing the query answers $a_1, \ldots, a_{q(n)}$, and finally compute the overall result as the prescribed Boolean function over the input values $a_1, \ldots, a_{q(n)}$. Hence, in this case, only one sub-system N inside membrane M suffices.

Going back to the scenario in which the skin membrane of Π_x contains $q(n)$ sub-systems N_i, for $1 \leq i \leq q(n)$, the fact that in the simulation described in Section 4 all operations are precisely synchronized, produces some technical annoyance when more than one query has to be simulated. Fortunately, all computations of the oracle nondeterministic Turing machines N_i can be easily padded to the same length $t(n)$ with no asymptotic loss of efficiency [22, Proposition 7.1]. In this way, all sub-systems N_i of Π_x will produce the answer to their query simultaneously. To distinguish the processing of each query, every membrane N contained in the skin membrane of Π_x will have a different label. The same must be done with the membranes A_N, and $F_0, \ldots, F_{t(n)-1}$, that are used to process the yes and no objects that represent the answers to the queries, and a new subscript must be added to the objects themselves in order not to mix them. So doing, all queries can be performed independently, they complete at the same time, and the simulation of M can start with the answer to all queries written on the tape on different cells.

All this said, we can now state the main result of this paper:

Theorem 1. *Semi-uniform families of depth-2 polarizationless P systems with active membranes, with rules for the evolution and communication of objects, and rules for the dissolution and division of elementary and (weak) non-elementary membranes, are able to solve all problems in* $\mathbf{P}_{\|}^{\mathbf{NP}}$ *in polynomial time.*

As a last observation, Theorem 1 can be extended to *uniform* families of the same kind of P systems. Recall from Definition 3 that, for each n, in the uniform setting a deterministic Turing machine F builds in polynomial time a common structure Π_n for all P systems dealing with inputs of length n. Such a structure has a distinguished input membrane, where the multiset w_x encoding the specific input x is placed, thus obtaining the P system Π_x. In order to extend our result to the uniform setting, we have thus to deal with two technical problems: (1) the structure (and hence the rules) of Π_n is fixed for all inputs of length n, and (2) all input objects are initially put in one membrane. Problem (1) can be easily solved by using one (or even a polynomial number of) *universal* nondeterministic Turing machine N to simulate the oracle(s), and a *universal* deterministic Turing machine M to simulate (with a polynomial slowdown) the computation of the Turing machine that invokes the oracle. The input to each of the universal machines will consist of a multiset of objects, that encodes both the input string and the description of the actual (deterministic or nondeterministic) Turing machine to be simulated. The two multisets of objects – one for machine N, and one for machine M – will be put in the input membrane of Π_x. To solve Problem (2), each object will have a subscript that indicates where the object should be placed in order to correctly represent the initial configuration of the two Turing machines. The simulation described in this paper is thus preceded by a preparation phase, executed using an appropriate set of communication rules, that simply places the input objects in the correct place. When this phase is completed, the simulation of the oracle nondeterministic Turing machine N can start, followed by the simulation of the deterministic Turing machine M.

6. Conclusions and directions for future research

In this paper we have proved that polarizationless P systems with active membranes, having a membrane hierarchy of depth 2 and using non-cooperative evolution rules, bidirectional communication rules, dissolution rules and weak division rules for non-elementary (and for elementary) membranes, working in polynomial time, can solve at least all decision problems in the complexity class $\mathbf{P}_{\|}^{\mathbf{NP}}$, the class of problems solved in polynomial time by deterministic Turing machines that are given the possibility to make a polynomial number of parallel queries to oracles for **NP** problems.

For simplicity we have proved our result in the semi-uniform setting, and limiting to one the number of oracle queries. However, we have given indications on how to extend this result to the uniform setting, and/or to the case in which a polynomial number of queries is performed, also possibly to different oracles. We have also discussed some restricted cases, in which either the number of nondeterministic Turing machines that simulate the oracles or even the number of the membranes in the P system that simulate such Turing machines, can be reduced to one.

Concerning future research, in our opinion the most interesting open problem is whether it is possible to obtain a *characterization* of the complexity class $\mathbf{P}_{\|}^{\mathbf{NP}}$ by means of the model of P systems used in this paper; in particular, determining what are the combinations of different computational features that allow to obtain or not such a characterization is certainly a challenging but intriguing avenue for new research. Maybe not so interesting from a theoretical point of view, but certainly useful when trying to apply the techniques described in this paper to real simulations, is to make explicit the construction just sketched above of the uniform construction. As the reader can imagine, this task is certainly possible but tedious, given the amount of technical details to be dealt with.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Gh. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (1) (2000) 108–143, https://doi.org/10.1006/jcss.1999.1693.
[2] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón, Tissue P systems, Theor. Comput. Sci. 296 (2) (2003) 295–326, https://doi.org/10.1016/S0304-3975(02)00659-X.
[3] M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fundam. Inform. 71 (2–3) (2006) 279–308, https://content.iospress.com/articles/fundamenta-informaticae/fi71-2-3-08.
[4] Gh. Păun, G. Rozenberg, A guide to membrane computing, Theor. Comput. Sci. 287 (1) (2002) 73–100, https://doi.org/10.1016/S0304-3975(02)00136-6.
[5] Gh. Păun, Membrane Computing: An Introduction, Springer, 2002.
[6] Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, 2010.
[7] The P systems web page, http://ppage.psystems.eu/.
[8] Gh. Păun, P systems with active membranes: attacking NP-complete problems, J. Autom. Lang. Comb. 6 (1) (2001) 75–90.
[9] 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, in: Proceedings of the Second International Conference, Springer, 2001, pp. 289–301.
[10] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Membrane division, oracles, and the counting hierarchy, Fundam. Inform. 138 (1–2) (2015) 97–111, https://doi.org/10.3233/FI-2015-1201.
[11] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, The counting power of P systems with antimatter, Theor. Comput. Sci. 701 (2017) 161–173, https://doi.org/10.1016/j.tcs.2017.03.045.
[12] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Characterising the complexity of tissue P systems with fission rules, J. Comput. Syst. Sci. 90 (2017) 115–128, https://doi.org/10.1016/j.jcss.2017.06.008.
[13] 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, in: Lecture Notes in Computer Science, vol. 8961, Springer, 2014, pp. 284–299.
[14] P. Sosík, A. Rodríguez-Patón, Membrane computing and complexity theory: a characterization of PSPACE, J. Comput. Syst. Sci. 73 (1) (2007) 137–152, https://doi.org/10.1016/j.jcss.2006.10.001.
[15] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Monodirectional P systems, Nat. Comput. 15 (4) (2016) 551–564, https://doi.org/10.1007/s11047-016-9565-2.
[16] L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez, Polarizationless P systems with active membranes: computational complexity aspects, J. Autom. Lang. Comb. 21 (1–2) (2016) 107–123, https://doi.org/10.25596/jalc-2016-107.
[17] D. Orellana-Martín, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez, A path to computational efficiency through membrane computing, Theor. Comput. Sci. 777 (2019) 443–453, https://doi.org/10.1016/j.tcs.2018.12.024.
[18] D. Orellana-Martín, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez, Minimal cooperation as a way to achieve the efficiency in cell-like membrane systems, J. Membr. Comput. 1 (2) (2019) 85–92, https://doi.org/10.1007/s41965-018-00004-9.
[19] A. Leporati, L. Manzoni, G. Mauri, A.E. Porreca, C. Zandron, Simulating counting oracles with cooperation, J. Membr. Comput. 2 (4) (2020) 303–310, https://doi.org/10.1007/s41965-020-00052-0.
[20] P. Sosík, P systems attacking hard problems beyond NP: a survey, J. Membr. Comput. 1 (2019) 198–208, https://doi.org/10.1007/s41965-019-00017-y.
[21] N. Murphy, D. Woods, The computational power of membrane systems under tight uniformity conditions, Nat. Comput. 10 (1) (2011) 613–632, https://doi.org/10.1007/s11047-010-9244-7.
[22] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1993.