Monodirectional P Systems*

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

Summary. We investigate the influence that the flow of information in membrane systems has on their computational complexity. In particular, we analyse the behaviour of P systems with active membranes where communication only happens from a membrane towards its parent, and never in the opposite direction. We prove that these "monodirectional P systems" are, when working in polynomial time and under standard complexity-theoretic assumptions, much less powerful than unrestricted ones: indeed, they characterise classes of problems defined by polynomial-time Turing machines with **NP** oracles, rather than the whole class **PSPACE** of problems solvable in polynomial space.

1 Introduction

P systems with active membranes working in polynomial time are known to be able to solve all **PSPACE**-complete problems [1]; this exploits membrane structures of polynomial depth and a bidirectional flow of information (in terms of moving objects or changing charges), both from a parent membrane to its children, and the in opposite direction.

When restricting the depth of the membrane structures of a family of P systems to a constant amount, it is still possible to solve problems in the counting hierarchy **CH**, defined in terms of polynomial-time Turing machines with oracles for counting problems [4]. In the proof of this result, it has been noticed that send-in communication rules of the form $a\,[\,]_h^\alpha \to [b]_h^\beta$ allow us to check whether the amount of objects located in a membrane exceeds a (possibly exponential) threshold in polynomial time.

It is then natural to ask whether that feature is actually necessary in order to obtain the power of counting in polynomial time. In this paper we prove (under the standard complexity-theoretic assumption that $\mathbf{P}^{\mathbf{NP}} \neq \mathbf{P}^{\#\mathbf{P}}$) that this is actually

* This work was partially supported by Università degli Studi di Milano-Bicocca, FA 2013: "Complessità computazionale in modelli di calcolo bioispirati: Sistemi a membrane e sistemi di reazioni".

the case: P systems with monodirectional communication, where the information flows only towards the outermost membrane, are limited to $\mathbf{P}^{\mathbf{NP}}$, the class of problems efficiently solved by Turing machines with **NP** oracles. This happens even when allowing polynomially deep membrane structures, a weak form of non-elementary membrane division, or dissolution (which, in this case, turns out to be as powerful as weak non-elementary division). The $\mathbf{P}^{\mathbf{NP}}$ upper bound is actually reached when dissolution or weak non-elementary division are allowed; if neither is available, then the computation power decreases to $\mathbf{P}_{\parallel}^{\mathbf{NP}}$, where the queries must all be fixed in advance, rather than asked adaptively. Chapter 17 of Papadimitriou's book [7] provides more details on complexity classes defined in terms of Turing machines with **NP** oracles.

For an introduction to P systems with active membranes ($\mathcal{AM}$), we refer the reader to the original paper by Gh. Păun [8], supplemented by the definitions of complexity classes $\mathbf{PMC}_{\mathcal{AM}}$ (resp., $\mathbf{PMC}^{\star}_{\mathcal{AM}}$) of problems solved by uniform (resp., semi-uniform) families of confluent P systems in polynomial time [5]. Define $\mathcal{M} = \mathcal{AM}(-\mathrm{i}, -\mathrm{n}, +\mathrm{wn})$ to be the class of *monodirectional P systems with active membranes*, without send-in rules; we also remove the usual ("strong") non-elementary division rules, of the form

$$\left[\left[\,\right]^{+}_{h_1} \cdots \left[\,\right]^{+}_{h_m} \left[\,\right]^{-}_{h_{m+1}} \cdots \left[\,\right]^{-}_{h_n} \right]^{\alpha}_{h} \rightarrow \left[\left[\,\right]^{\delta}_{h_1} \cdots \left[\,\right]^{\delta}_{h_m} \right]^{\beta}_{h} \left[\left[\,\right]^{\zeta}_{h_{m+1}} \cdots \left[\,\right]^{\zeta}_{h_n} \right]^{\gamma}_{h}$$

since they also provide a way for membrane h to share information with its children by changing their charge. We replace these rules by "weak" non-elementary division rules [11] of the form $[a]^{\alpha}_{h} \rightarrow [b]^{\beta}_{h}\, [c]^{\gamma}_{h}$, which allow the creation of complex membrane structures (such as complete binary trees) without exchanging information with the children membranes.

Let $\mathcal{M}(-\mathrm{d})$, $\mathcal{M}(-\mathrm{wn})$, and $\mathcal{M}(-\mathrm{d}, -\mathrm{wn})$ denote monodirectional P systems without dissolution, without weak non-elementary division, and without both kinds of rules, respectively. For each class $\mathcal{D}$ of P systems, let $\mathbf{PMC}_{\mathcal{D}}$ and $\mathbf{PMC}^{\star}_{\mathcal{D}}$ be the classes of problems solvable by uniform and semi-uniform families of P systems of class $\mathcal{D}$. Then, the main results of this paper can be summarised as follows:

- The whole class $\mathbf{PMC}^{[\star]}_{\mathcal{M}}$, as well as $\mathbf{PMC}^{[\star]}_{\mathcal{M}(-\mathrm{d})}$ and $\mathbf{PMC}^{[\star]}_{\mathcal{M}(-\mathrm{wn})}$, are equivalent to $\mathbf{P}^{\mathbf{NP}}$. Here $[\star]$ denotes optional semi-uniformity.
- The class $\mathbf{PMC}^{[\star]}_{\mathcal{M}(-\mathrm{d},-\mathrm{wn})}$ is equivalent to $\mathbf{P}_{\parallel}^{\mathbf{NP}}$.

The rest of the paper is structured as follows: in Section 2 we prove some basic limitations of monodirectional P systems; in Section 3 we exploit these results to prove upper bounds to the complexity classes for monodirectional P systems; in Section 4 we provide the corresponding lower bounds by simulating Turing machines with **NP** oracles; in Section 5 some results of the preceding sections are improved; finally, in Section 6 we present some open problems and directions for future research.

2 Properties of monodirectional P systems

We begin by proving some properties of monodirectional P systems that show how the lack of inbound communication substantially restricts the range of behaviours exhibited during the computations.

Definition 1. *Let Π be a P system, and let $\mathcal{C}$ and $\mathcal{D}$ be configurations of Π. We say that $\mathcal{C}$ is a* restriction *of $\mathcal{D}$, in symbols $\mathcal{C} \sqsubseteq \mathcal{D}$, if the membrane structures of the two configurations are identical (i.e., they have the same shape, labelling, and charges) and each multiset of objects of $\mathcal{C}$ is a submultiset of that located in the corresponding region of $\mathcal{D}$.*

The following proposition shows that, while a recogniser P system working in time t might create exponentially many objects per region during its computation, only a polynomial amount (with respect to t) of them in each region does actually play a useful role if the system is monodirectional: indeed, the final result of the computation can be identified by just keeping track of a number of objects per region equal to the number of steps yet to be carried out.

Lemma 1. *Let Π be a monodirectional recogniser P system, and let $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_t)$, with $t \geq 1$, be a halting computation of Π. Then, there exists a sequence of configurations $(\mathcal{D}_0, \ldots, \mathcal{D}_t)$ such that*

- *(i) we have $\mathcal{D}_i \sqsubseteq \mathcal{C}_i$ for $0 \leq i \leq t$, and each multiset of $\mathcal{D}_i$ has at most $t - i$ objects;*
- *(ii) for all $i < t$ there exists a configuration $\mathcal{E}_{i+1}$ such that $\mathcal{E}_{i+1}$ is reachable in one step from $\mathcal{D}_i$ ($\mathcal{D}_i \to \mathcal{E}_{i+1}$ for brevity) and $\mathcal{D}_{i+1} \sqsubseteq \mathcal{E}_{i+1}$;*
- *(iii) a send-out rule of the form $[a]_h^\alpha \to [\,]_h^\beta$ yes (resp., $[a]_h^\alpha \to [\,]_h^\beta$ no) is applied to the outermost membrane during the transition step $\mathcal{D}_{t-1} \to \mathcal{E}_t$ if and only if $\boldsymbol{\mathcal{C}}$ is an accepting (resp., rejecting) computation.*

Proof. By induction on t. If $t = 1$, then the environment of $\mathcal{C}_1$ contains *yes* or *no*, which have been sent out during the computation step $\mathcal{C}_0 \to \mathcal{C}_1$ by a rule $[a]_h^\alpha \to [\,]_h^\beta$ *yes* or $[a]_h^\alpha \to [\,]_h^\beta$ *no*. Let $\mathcal{D}_0 \sqsubseteq \mathcal{C}_0$ be obtained by keeping only the objects on the left-hand side of send-out, dissolution, and division rules applied during $\mathcal{C}_0 \to \mathcal{C}_1$ (we call these rules "blocking", since at most one of them can be applied inside each membrane at each step). At most one object per region is kept, given the lack of send-in rules. Let $\mathcal{D}_1 \sqsubseteq \mathcal{C}_1$ be obtained by deleting all objects. Then:

- *(i)* we have $\mathcal{D}_0 \sqsubseteq \mathcal{C}_0$ and $\mathcal{D}_1 \sqsubseteq \mathcal{C}_1$ by construction, and all multisets of $\mathcal{D}_0$ and $\mathcal{D}_1$ have at most 1 and exactly 0 objects, respectively;
- *(ii)* let the transition $\mathcal{D}_0 \to \mathcal{E}_1$ be computed by applying all blocking rules applied during the step $\mathcal{C}_0 \to \mathcal{C}_1$, which are all enabled by construction; then $\mathcal{E}_1 \sqsubseteq \mathcal{C}_1$ and, since $\mathcal{D}_1 \sqsubseteq \mathcal{C}_1$ and $\mathcal{D}_1$ contains no objects, necessarily $\mathcal{D}_1 \sqsubseteq \mathcal{E}_1$;
- *(iii)* the computation $\boldsymbol{\mathcal{C}}$ is accepting if and only if the rule $[a]_h^\alpha \to [\,]_h^\beta$ *yes* is applied from $\mathcal{C}_0$, and the latter is equivalent by construction to that rule being applicable from $\mathcal{D}_0$ (the reasoning is similar if $\boldsymbol{\mathcal{C}}$ is rejecting).

This proves the base case. Now let $\boldsymbol{\mathcal{C}} = (\mathcal{C}_{-1}, \mathcal{C}_0, \ldots, \mathcal{C}_t)$ be a halting computation of length $t+1$. The sub-computation $(\mathcal{C}_0, \ldots, \mathcal{C}_t)$ is also halting, and by induction hypothesis there exists a sequence of configurations $(\mathcal{D}_0, \ldots, \mathcal{D}_t)$ satisfying (i)–(iii). Construct the configuration $\mathcal{D}_{-1}$ as follows: first of all, keep all objects from $\mathcal{C}_{-1}$ that appear on the left-hand side of blocking rules applied during the computation step $\mathcal{C}_{-1} \to \mathcal{C}_0$; this requires at most one object per region, and guarantees that the membrane structure's shape and charges can be updated correctly (i.e., the same as $\mathcal{C}_0$ and $\mathcal{D}_0$).

We must also ensure that all objects of $\mathcal{D}_0$ can be generated from $\mathcal{D}_{-1}$ during the transition $\mathcal{D}_{-1} \to \mathcal{E}_0$. Once the blocking rules to be applied have been chosen, any object a located inside a membrane of $\mathcal{D}_0$ can be traced back to a single object in $\mathcal{D}_{-1}$. Either a appears on the right-hand side of one of those blocking rules, or it appears on the right-hand side of an object evolution rule applied in the step $\mathcal{C}_{-1} \to \mathcal{C}_0$, or it does not appear explicitly in any rule applied in that step; in the latter case, it is either carried on unchanged from $\mathcal{D}_{-1}$ (possibly from another region, if membrane dissolution occurred), or is created by duplicating the content of a membrane by applying a division rule (triggered by a different object). As a consequence, at most t objects per region of $\mathcal{D}_{-1}$, possibly in conjunction with a single object per region involved in blocking rules, suffice in order to generate the t objects per region of $\mathcal{D}_0$. As a consequence,

- (i) we have $\mathcal{D}_{-1} \sqsubseteq \mathcal{C}_{-1}$ by construction, and $\mathcal{D}_{-1}$ contains at most $t+1$ objects per region;
- (ii) by applying all blocking rules and as many evolution rules as possible from the computation step $\mathcal{C}_{-1} \to \mathcal{C}_0$ in $\mathcal{D}_{-1}$, we obtain a configuration $\mathcal{E}_0$ with the same membrane structure as $\mathcal{D}_0$ and, as mentioned above, containing all objects from $\mathcal{D}_0$ (and possibly other objects generated by evolution rules).

Since (iii) holds by induction hypothesis, this completes the proof. □

Notice that this lemma does not give us an efficient algorithm for choosing *which* objects are important for each step of the computation; it only proves that a small (i.e., polynomial-sized) multiset per region exists. However, it is easy to find such an algorithm by slightly relaxing the conditions: instead of limiting the cardinality of the multisets to $t - i$, we limit the number of occurrences *of each symbol* to that value, and simply delete the occurrences in excess separately for each symbol. This gives us the larger cardinality bound $|\Gamma| \times (t - i)$ per region, which is polynomial whenever the number of computation steps of the system is, and still allows us to simulate the overall behaviour of the P system.

Lemma 1 fails for P systems with send-in rules because some configurations where each multiset is small nonetheless require a previous configuration with a region containing exponentially many objects. This is the case, for instance, for P systems solving counting problems, where the number of assignments satisfying a Boolean formula is checked against a threshold by means of send-in rules [4]. Those assignments are represented in the P system by a potentially exponential number of objects located in the same region, which are sent into exponentially many children

membranes in parallel (i.e., at most one object enters each child membrane), and cannot always be reduced to a polynomial amount without changing the accepting behaviour of the P system.

Another property of monodirectional P systems is the existence of computations where membranes having the same labels always have children (and, recursively, all the descendents) with the same configuration. This property will be useful when simulating confluent recogniser monodirectional P systems in Section 3.

Lemma 2. *Let Π be a monodirectional P system. Then there exists a computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_t)$ of Π where, in each configuration $\mathcal{C}_i$, the following holds: any two subconfigurations[2] of $\mathcal{C}_i$ having membranes with the same label as roots are identical, except possibly for the multiset and charge of the root membranes themselves.*

Proof. By induction on i. The statement trivially holds for the initial configuration of Π, since the membranes are injectively labelled.

When a division rule is applied to a membrane h, two subconfigurations with root h are created; this is the only way to generate multiple membranes sharing the same label. The two resulting subconfigurations may only differ with respect to the contents and charges of the root membranes, since the internal membranes have evolved before the division of h occurs (recall that the rules are applied, from a logical standpoint, in a bottom-up way [8]).

On the other hand, if two subconfigurations with identically labelled root membranes already exist in a configuration $\mathcal{C}_i$, then we can assume that the property holds by induction hypothesis. We can then nondeterministically choose which rules to apply in the subconfiguration having the first membrane as root, excluding the root itself; since the other subconfiguration is identical (except possibly for the root), the same multiset of rules can also be applied to it, thus preserving the property in the next configuration of the system. □

While Lemma 2 somehow "compresses" each level of the configuration of monodirectional P systems, it does not, however, reduce the number of distinct membranes per level to a polynomial number. Indeed, the standard membrane computing technique of generating all (exponentially many) possible assignments to a set of variables does not require send-in rules [10], and can be carried out in parallel on all levels of the membrane structure.

Lemma 2 also fails for P systems with send-in rules. The reason is that two identical subconfigurations can be made different by having a single object located immediately outside, and nondeterministically sending it into one of the root membranes of the two subtrees; the evolution of the two branches of the system might then diverge completely.

[2] We define a subconfiguration of $\mathcal{C}_i$ as a subtree (a root node together with all its descendents) of the membrane structure of $\mathcal{C}_i$, including labels, multisets, and charges of the membranes.

3 Simulation of monodirectional P systems

It is a well-known result in membrane computing that P systems with active membranes can be simulated in polynomial time by deterministic Turing machines if no membrane division rules are allowed [10]. More specifically, the portion of the system that is not subject to membrane division can be simulated deterministically with a polynomial slowdown, while the output of the dividing membranes can be obtained by querying an appropriate oracle. It was recently proved that, for standard (bidirectional) P systems where only elementary membranes can divide, an oracle for a **#P** function is necessary and sufficient [5].

In what follows we prove that an **NP**-oracle is sufficient for the simulation of monodirectional P systems. In particular, the oracle will solve the following problem.

Lemma 3. *Given the initial configuration of an elementary membrane with label h of a monodirectional P system, an object type $a \in \Gamma$, and two integers $k, t \in \mathbb{N}$* in unary notation*, it is* **NP***-complete to decide whether the set of membranes with label h existing at time t emits (via send-out or dissolution rules) at least k copies of object a at that time step.*

Proof. The problem is **NP**-hard, since one can simulate an arbitrary polynomial-time, nondeterministic Turing machine M by using a single membrane with elementary division (without using send-in rules) and obtain the same result as M by checking if the resulting membranes send out at least one ($k = 1$) "acceptance object" at a specific time step [4].

Conversely, the problem can be solved by a nondeterministic, polynomial-time Turing machine M as follows. Simulate t computation steps of the membrane explicitly, by keeping track of its charge and multiset, as in any standard simulation [10]. If the membrane divides, then M keeps track of all the resulting membranes, *until the number exceeds k*. If that happens, then k copies of the membrane are chosen nondeterministically among those being simulated (which are at most $2k$ after any simulated step, if all membranes divide), and the remaining ones are discarded. Since there is no incoming communication, any instance of the membrane can be simulated correctly, as its behaviour does not depend on the behaviour of its siblings. If one of the simulated membranes dissolves before t steps, one of the k "slots" is released and can be reused in case of a further membrane division.

After having simulated t steps as described, the machine M accepts if and only if at least k copies of a are emitted (sent out, or released by dissolution) in the last step by the membranes being simulated. At most k membranes need to be simulated in order to check whether at least k copies of the object are emitted and, by exploiting nondeterminism, we are guaranteed that the correct subset of membranes is chosen by at least one computation of M. Since k and t are polynomial with respect to the size of the input, the result follows. □

The values of t and k are given in unary since, otherwise, the number of steps or the number of membranes to simulate could be exponential with respect to the size of the input, and the problem would not be solvable in polynomial time.

As a consequence of Lemma 3, monodirectional P systems without non-elementary division can be simulated in polynomial time with access to an **NP** oracle.

Theorem 1. $\mathbf{PMC}^{\star}_{\mathcal{M}(-\mathrm{wn})} \subseteq \mathbf{P}^{\mathbf{NP}}$.

Proof. The rules applied to non-elementary membranes can be simulated directly in deterministic polynomial time by a Turing machine M [5]; this includes the outermost membrane, which ultimately sends out the result object. In order to update the configurations of the non-elementary membranes correctly, the objects emitted from elementary membranes (which potentially divide) have to be added to their multisets.

Suppose the P systems of the family being simulated work in polynomial time $p(n)$. By Lemma 1, the final result of the computation can be correctly determined by keeping track of at most $p(n)$ copies of each object per region. Hence, we can update the configurations by using an oracle for the problem of Lemma 3. At time step t, we make multiple queries for each label h of an elementary membrane and for each object type $a \in \Gamma$: by performing a binary search on k over the range $[0, p(n)]$, we can find the exact number of copies of a emitted by membranes with label h at time t, or discover that this number is at least $p(n)$ (and, in that case, we only add $p(n)$ objects to the multiset). This completes the proof. □

Monodirectional P systems without non-elementary division become weaker if dissolution is also disallowed: now a membrane cannot *become* elementary during the computation, and thus the evolution of each dividing membrane is always independent of the rest of the system. This allows us to perform all queries in parallel, rather than sequentially (in an adaptive way).

Theorem 2. $\mathbf{PMC}^{\star}_{\mathcal{M}(-\mathrm{d},-\mathrm{wn})} \subseteq \mathbf{P}^{\mathbf{NP}}_{\parallel}$.

Proof. If dissolution rules are not allowed, being elementary is a static property of the membranes, i.e., a membrane is elementary for the whole computation if and only if it is elementary in the initial configuration. By observing that each query is completely independent of the others (i.e., each query involves a different membrane, time step and object) and also independent of the configurations of the non-dividing membranes (due to the lack of send-in rules), we can perform them in parallel even before starting to simulate the P system. This proves the inclusion in $\mathbf{P}^{\mathbf{NP}}_{\parallel}$. □

Now consider monodirectional P systems with non-elementary membrane division. For this kind of systems, the behaviour of a dividing membrane is, of course, dependent on the behaviour of its children and, recursively, of all its descendants.

In order to simulate the behaviour of the children by using oracles, we define a more general query problem, where we assume that the behaviour of the descendents of the membrane mentioned in the query has already been established.

First of all, notice that the lack of send-in rules allows us to extend the notion of transition step $\mathcal{C} \to \mathcal{D}$ between configurations to labelled subforests[3] $\mathcal{E}$ of $\mathcal{C}$ and $\mathcal{F}$ of $\mathcal{D}$ as $\mathcal{E} \to \mathcal{F}$; the only differences from the standard definition are that $\mathcal{E}$ is not necessarily a single tree, and that its outermost membranes may divide and dissolve.

Definition 2. *Let Π be a monodirectional P system, let $\mathcal{C}$ be a configuration of Π, and let $h \in \Lambda$ be a membrane label. A subforest $\mathcal{S}$ of $\mathcal{C}$ is called a* label-subforest induced by h, *or* h-subforest *for brevity, if one of the following conditions hold:*

- *$\mathcal{C}$ is the initial configuration of Π, and $\mathcal{S}$ consists of a single tree rooted in the (unique) membrane h,*
- *$\mathcal{C}$ is a possible configuration of Π at time $t+1$ with $\mathcal{C}' \to \mathcal{C}$, and there exists an h-subforest $\mathcal{S}'$ in $\mathcal{C}'$ such that $\mathcal{S}' \to \mathcal{S}$.*

The notion of h-subforest can be viewed as a generalisation of the equivalence classes of membranes in P systems without charges defined by Murphy and Woods [6].

Lemma 4. *Let Π be a monodirectional P system. Then there exists a computation of Π where, at each time step and for each membrane label $h \in \Lambda$, all h-subforests are identical.*

Proof. Multiple h-subforests can only be created by division of an ancestor of h; but then, by Lemma 2, there exists a computation of Π where the resulting h-subforests are identical. □

Example 1. Figure 1 shows the evolution of the membrane structure of a monodirectional P system and its label-subforests. The label-subforests in the initial configuration $\mathcal{C}_0$ coincide with all downward-closed subtrees. In the computation step $\mathcal{C}_0 \to \mathcal{C}_1$ both h_2 and h_3 divide; the division of the latter causes the duplication of the h_3- and h_4-subforests (and, indirectly, of the h_5-subforest); the division of an ancestor membrane is the only way to have more than one label-subforest. By Lemma 4, we can always assume that multiple label-subforests induced by the same label are identical. In the computation step $\mathcal{C}_1 \to \mathcal{C}_2$, the rightmost membrane having label h_2 and both instances of h_4 dissolve. Notice that this does *not* cause the disappearance of the two h_4-subforests: in the general case, the membranes h_4 might contain label-subforests induced by different labels, and we still need to refer to them as a single entity (the h_4-subforest), without the need to describe the internal structure, even when h_4 ceases to exist.

As can be observed from Figure 1, a subforest can be identified as an h-subforest by checking whether it can be generated from the downward-closed subtree rooted in h in the initial configuration.

[3] We define a subforest F' of a forest F to be any subgraph such that, whenever F' includes a vertex v, it also includes all the descendents of v.

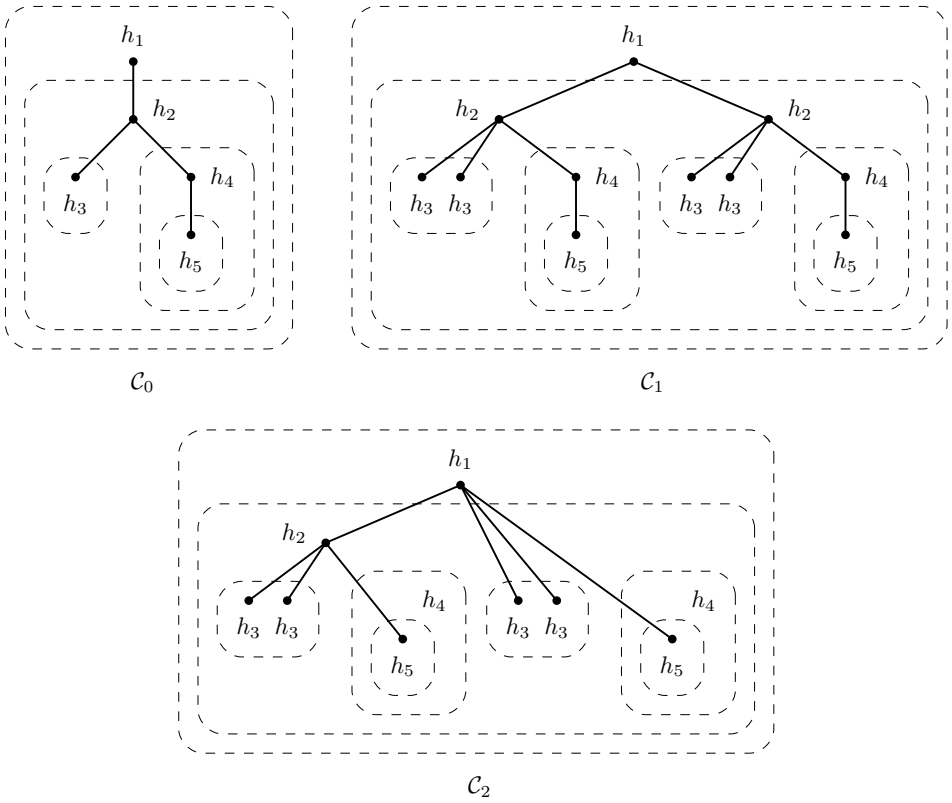

Fig. 1. Evolution of a membrane structure and its label-subforests, which are enclosed by dashed rectangles.

A computation that ensures that all h-subforests are identical for all $h \in \Lambda$ can be obtained by imposing a total ordering (a priority) on the set of rules of the P system, and applying inside each membrane the rules with higher priority whenever possible. In the following, we assume that a priority order (e.g., the lexicographic order) has been fixed; there is no loss of generality in doing that, since we only focus on *confluent* P systems in this paper. We define the multiset of objects emitted by a label-subforest as the union of the multisets emitted by its outermost membranes.

Lemma 5. *Given the initial configuration of a membrane with label h of a monodirectional P system, an object $a \in \Gamma$, two integers $k, t \in \mathbb{N}$* in unary notation*, and a table T of the objects emitted during computation steps $1, \ldots, t$ by the label-subforests immediately contained in h, it is* **NP***-complete to decide whether each h-subforest emits at least k copies of object a at time t.*

Proof. The problem is **NP**-hard, since the set of elementary membranes with label h of Lemma 3 is an example of h-subforest; that problem is thus a special case (limited to label-subforests of height 0) of the current one.

To prove membership in **NP** we also use an algorithm similar to the proof of Lemma 3: simulate up to k instances of membrane h, nondeterministically choosing which ones to keep when a membrane division occurs. However, besides simulating the rules directly involving the membranes with label h, we need to update their configuration by adding, at each computation step, the objects emitted by the label-subforests they contain. This is trivial, since the required data is supplied as the input table T. Here we exploit Lemma 4, and simulate a computation where all label-subforests contained in multiple instances of h are identical, and always emit the same objects.

The other main difference from the proof of Lemma 3 is that we do not release one of the k slots when one instance of membrane h dissolves, since its children may still emit objects, and those count in determining the output of the h-subforest. Rather, if an instance of h currently being simulated dissolved during steps $1, \ldots, t$, then we add the outputs at time t of the label-subforests immediately contained in h to the result of the computation; those outputs are obtained from table T.

The statement of this lemma then follows from an argument completely analogous to that presented in the proof of Lemma 3: there exists a sequence of nondeterministic choices leading to the simulation of k instances of h sending out at least k objects if and only if at least k objects are actually sent out by the P system being simulated. □

We can finally show that monodirectional P systems using non-elementary division (and dissolution) also do not exceed the upper bound $\mathbf{P}^{\mathbf{NP}}$.

Theorem 3. $\mathrm{PMC}^{\star}_{\mathcal{M}} \subseteq \mathbf{P}^{\mathbf{NP}}$.

Proof. We use an algorithm similar to the one described in the proof of Theorem 1. However, instead of using the oracle to compute the output of the elementary membranes, we use it to compute the output of the label-subforests. This requires first asking all queries for the label-subforests of height 0 (with an empty table T), then using the results as the table T for the queries involving label-subforests of height 1, and so on, until reaching the non-divisible membranes; these can be simulated directly by using the results of the queries involving the label-subforests immediately contained in them. Notice that the queries involving label-subforests of a given height can always be asked in parallel (across all values of a, k, t); the queries must be asked sequentially only when involving different heights. □

4 Simulation of $\mathbf{P}^{\mathbf{NP}}$ machines

In order to prove the converse inclusions between complexity classes, we describe a simulation of any Turing machine M with an **NP** oracle by means of monodirectional

P systems (an adaptation of [4]). Let Q be the set of states of M; we assume, without loss of generality, a binary alphabet $\{0,1\}$ for M. Finally, we denote by $\delta\colon Q \times \Sigma \to Q \times \Sigma \times \{\triangleleft, \triangleright\}$ the transition function of M.

Suppose that the configuration of M at a certain time step is the following: the tape contains the string $x = x_1 \cdots x_m$, the state of the machine is q, and the tape head is located on cell i. This configuration is encoded as a multiset located in a single membrane h of the P system, as follows. There is one object 1_{j-i} for each $1 \leq j \leq m$ such that $x_j = 1$; that is, each 1 in the string x is represented as an object indexed by its position in x, shifted by i; the 0s of x are not represented by an object, but rather by the absence of the corresponding 1. The object 1_0 (resp., its absence) represents a 1 (resp., a 0) located under the tape head; the indices will be updated (increased or decreased) when simulating a tape head movement. Finally, the state q of M is encoded as an object q with the same name. Further objects, not part of the encoding of the configuration of M, may also appear for simulation purposes.

A transition step of M is simulated by 7 steps of the P system. We assume that the membrane h containing the encoding of the configuration of M also contains the object $\ominus$.

Step 1. The object $\ominus$ is sent out (as the "junk" object #) in order to change the charge of h to negative:

$$[\ominus]_h^0 \to [\,]_h^- \,\# \tag{1}$$

Step 2. When h is negative, the object 1_0 is sent out, if appearing, in order to change the charge to positive. If 1_0 does not appear, the membrane remains negative.

$$[1_0]_h^- \to [\,]_h^+ \,\# \tag{2}$$

The remaining tape-objects are primed:

$$[1_i \to 1_i']_h^- \qquad\qquad \text{for } i \neq 0 \tag{3}$$

The state-object q is also primed, and produces the object $\odot$:

$$[q \to q' \odot]_h^- \tag{4}$$

Step 3. The system can now observe the charge of h and establish whether 1_0 appeared (i.e., whether the symbol under the tape head was 1) or not (i.e., the symbol was 0); this corresponds to a positive or negative charge, respectively. The object q' is rewritten accordingly:

$$[q' \to (q,1)]_h^+ \qquad\qquad [q' \to (q,0)]_h^- \tag{5}$$

At the same time, the neutral charge of h is restored by $\odot$:

$$[\odot]_h^\alpha \to [\,]_h^0 \,\# \qquad\qquad \text{for } \alpha \in \{+,-\} \tag{6}$$

Step 4. For the sake of example, suppose the transition function of M on state q is defined by $\delta(q,0) = (r,1,\triangleright)$ and $\delta(q,1) = (s,0,\triangleleft)$; the other cases are similar. The object $(q,0)$ or $(q,1)$ is rewritten accordingly:

$$[(q,0) \to (r,1,\triangleright)]_h^0 \qquad [(q,1) \to (s,0,\triangleleft)]_h^0 \tag{7}$$

Simultaneously, the tape-objects are primed again:

$$[1_i' \to 1_i'']_h^0 \qquad \text{for } i \neq 0 \tag{8}$$

Step 5. Now the triple generated in the previous step is "unpacked" into its components, which include an object that will be eventually rewritten into the new state-object, the object $1_0''$ (or nothing), and an object to be used to change the charge according to the direction of the movement of the tape head:

$$[(r,1,\triangleleft) \to \hat{r}\ 1_0''\ \oplus]_h^0 \qquad \text{for } r \in Q \tag{9}$$
$$[(r,1,\triangleright) \to \hat{r}\ 1_0''\ \ominus]_h^0 \qquad \text{for } r \in Q \tag{10}$$
$$[(r,0,\triangleleft) \to \hat{r}\ \oplus]_h^0 \qquad \text{for } r \in Q \tag{11}$$
$$[(r,0,\triangleright) \to \hat{r}\ \ominus]_h^0 \qquad \text{for } r \in Q \tag{12}$$

Step 6. The object $\oplus$, if appearing, changes the charge of the membrane to positive:

$$[\oplus]_h^0 \to [\ ]_h^+ \ \# \tag{13}$$

If $\ominus$ appears, it behaves similarly, according to rule (1). Simultaneously, the object $\hat{r}$ is primed and produces $\odot$:

$$[\hat{r} \to \hat{r}'\ \odot]_h^0 \qquad \text{for } r \in Q \tag{14}$$

Step 7. Now the charge of h is negative if the tape head is moving right, and the indices of the tape-objects have to be decremented, or positive if the tape head is moving left, and the indices must be incremented; the primes are also removed:

$$[1_i'' \to 1_{i-1}]_h^- \qquad [1_i'' \to 1_{i+1}]_h^+ \qquad \text{for } -(m-1) \leq i \leq m-1 \tag{15}$$

The object $\hat{r}'$ is now rewritten into the state-object r, and produces the $\ominus$ object to be used in Step 1 of the simulation of the next step of M:

$$[\hat{r}' \to r\ \ominus]_h^\alpha \qquad \text{for } r \in Q \text{ non final and } \alpha \in \{+,-\} \tag{16}$$

Finally, the neutral charge of h is restored by $\odot$ through rule (6). The configuration of the membrane now encodes the next configuration of M, and the system can begin simulating the next computation step. The process is depicted in Figure 2.

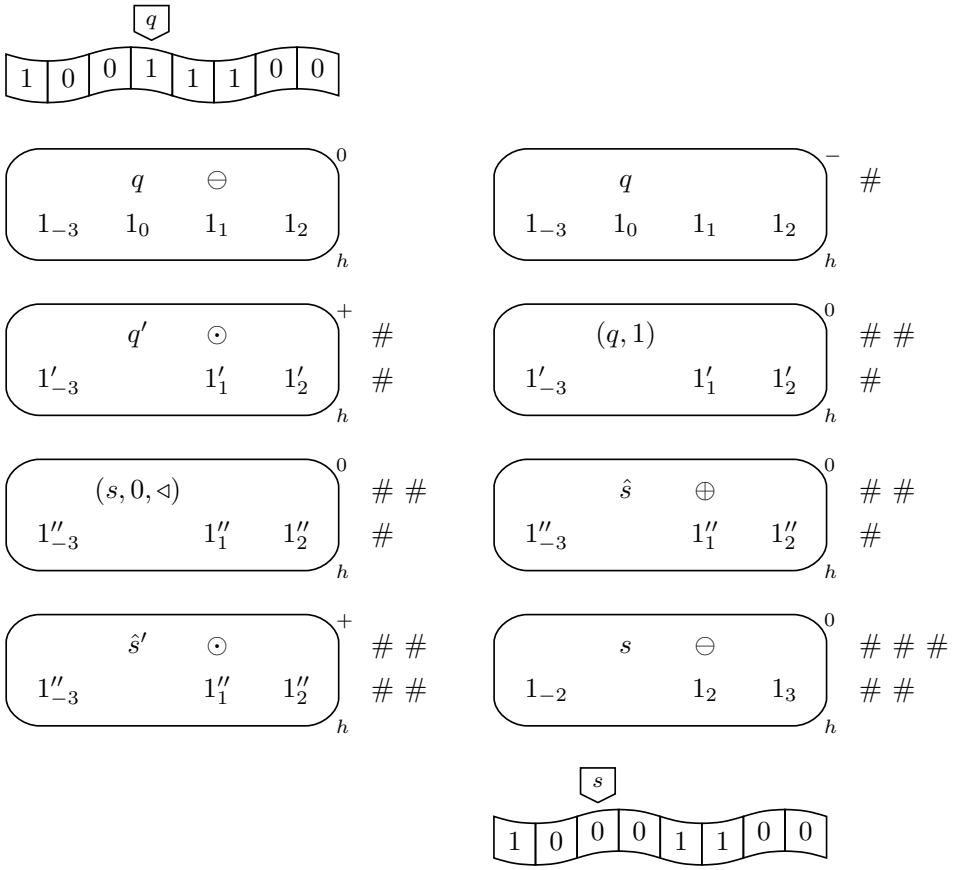

Fig. 2. Two successive Turing machine configurations, and the configurations of the P system simulating the transition step (in left-to-right, top-to-bottom order).

When $r \in Q$ is a final state (accepting or rejecting), instead of applying rule (16) the system rewrites the object $\hat{r}'$ as *yes* or *no*:

$$[\hat{r}' \to yes]_h^\alpha \qquad \text{for } r \in Q \text{ accepting and } \alpha \in \{+, -\} \tag{17}$$

$$[\hat{r}' \to no]_h^\alpha \qquad \text{for } r \in Q \text{ rejecting and } \alpha \in \{+, -\} \tag{18}$$

The object *yes* or *no* is then sent out as the result of the computation of the P system in the next step:

$$[yes]_h^0 \to [\,]_h^0\, yes \qquad\qquad [no]_h^0 \to [\,]_h^0\, no \tag{19}$$

It is easy to see that this simulation provides us with a uniform family of P systems $\boldsymbol{\Pi}_M = \{\Pi_x : x \in \{0,1\}^\star\}$, each consisting of a single membrane h and simulating the deterministic Turing machine M on all possible inputs.

4.1 Simulating oracle queries

If membrane h is not the outermost membrane of the system, then we can use division rules to simulate nondeterminism with parallelism. Suppose, for the sake of example, that the transition function of M describes nondeterministic binary choices such as $\delta(q, 0) = \{(r, 1, \triangleright), (s, 0, \triangleleft)\}$. Then, instead of the rules (7), we define the elementary division rule

$$[(q, 0)]_h^0 \to [(r, 1, \triangleright)]_h^0 \; [(s, 0, \triangleleft)]_h^0 \tag{20}$$

The two resulting copies of membrane h can then evolve in parallel according to the two possible choices.

This construction allows us to simulate polynomial-time deterministic Turing machines M with an **NP** oracle. In this section, we use the following conventions: the machine M simulates a work tape and a query tape with a single tape, by using the odd and even positions, respectively. When making a query, M writes the query string in the even positions of its tape, then enters a query state $q_?$. The oracle answers by erasing the query string (i.e., overwriting it with zeros), except for the first cell, where it writes 0 or 1 according to the result. The machine M then resumes its computation in state $q_!$, with the tape head located on the answer.

The oracle can be simulated by a polynomial-time nondeterministic Turing machine M', having initial state $q_?$ and deciding the oracle language. This machine uses only the even positions of the tape, and ends its computation in the post-query configuration described above. We assume that M' performs a series of nondeterministic choices leading to acceptance, if an accepting computation exists at all.

This combination of M and M' can be simulated by linearly nested membranes of a P system, one membrane for each query to be asked. The computation begins inside the innermost membrane, where we place a multiset encoding the initial configuration of M on its input x; whenever a query is performed, the computation moves one level higher in the membrane structure. In the following description we refer to all nested membranes as h, for brevity; the labels can be made unique, and the rules replicated for each label, with a polynomial-time preprocessing. The P system simulates the computation steps of M as described above, until M enters the query state $q_?$. Now the system pauses the simulation of M. Instead of producing $q_?$ and $\ominus$, as in rule (16), the system produces $q_?$ and $\tilde{q}_{!,t}$, where t is the maximum number of steps required by M' on query strings written by M. This number can be bounded above by considering the polynomial running time of M' on the longest possible query string, which is at most as long as the running time of M on its input x. The object $\tilde{q}_{!,t}$ is sent out from h as $q_{!,t}$, setting its charge to negative as $\ominus$ does, and upon reaching the parent membrane it begins counting down:

$$[\tilde{q}_{!,t}]_h^0 \to [\,]_h^- \; q_{!,t} \tag{21}$$

$$[q_{!,j} \to q_{!,j-1}]_h^0 \qquad \text{for } 1 \le j \le t \tag{22}$$

In the internal membrane, the nondeterministic Turing machine M' is now simulated. Since M' is allowed to make nondeterministic choices, in general there will be a number of membranes simulating M' after the first simulated step. When one of these membranes is simulating the last step of a computation of M', the object $\hat{q}'_!$ is produced by rule (14): then, instead of having a rule of type (16), the object $\hat{q}'_!$ is used to dissolve the membrane and release the tape-objects to the parent membrane:

$$[\hat{q}'_!]^{\alpha}_h \to \# \qquad \text{for } \alpha \in \{+,-\} \tag{23}$$

After t steps, all membranes simulating M' have completed the simulation, and have released their contents to the parent membrane. This membrane now contains:

- the object $q_{!,0}$;
- objects 1_i corresponding to the 1s contained in the odd positions of the tape of M (which are left unchanged by the simulation of M'); each of these objects has a multiplicity equal to the number of computations of M' on the previous query string;
- zero or more occurrences of 1_1, one for each accepting computation of M' on the query string; in particular, there is at least one occurrence of 1_1 if and only if the query string is accepted by the oracle. Notice that this object has index 1 even if it is on the first even position of the tape, since index 0 is reserved to the tape cell under the head (tape cell 1).

Before resuming the simulation of M, the system needs to eliminate any duplicate copies of objects 1_i. First of all, the object $q_{!,0}$ is rewritten into $q_!$, the next state of M:

$$[q_{!,0} \to q_!]^0_h \tag{24}$$

We then change the behaviour of M in such a way that, before continuing its original computation after receiving the answer to the oracle query, it sweeps its entire tape left-to-right and back to the first cell. This behaviour, in conjunction with the following extra rule of the P system:

$$[1_0 \to \epsilon]^+_h \tag{25}$$

erases any duplicate of 1_i for all i. Indeed, if a copy of 1_0 appears when h is positive, then another copy has been sent out in the previous step by rule (2); rule (25) eliminates such duplicates.

When the tape head of M moves back to the leftmost cell, the machine can resume its original behaviour, and the encoding of the configuration of M in the P system is now correct according to the description given at the beginning of this section.

Further queries by M are simulated analogously, by exploiting another level of the membrane structure. Notice that simulating a query actually "consumes" one level of the membrane structure, due to the dissolution rule (23). For this reason, the initial membrane structure of the P system simulating M consists of an outermost membrane, containing as many nested membranes as the number of queries performed by M.

Theorem 4. *A deterministic polynomial-time Turing machine which asks $p(n)$ queries to an* **NP** *oracle on inputs of length n can be simulated by a uniform family of monodirectional P systems of depth $p(n)$ without non-elementary division rules.*

Proof. The family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \{0,1\}^\star\}$ simulating M on input x can be constructed uniformly in polynomial time, since only the initial multiset depends on the actual string x, while the set of rules and the membrane structure only depend on $|x|$. We only need to make sure that the indices of the tape-objects are large enough to ensure that both the tape of M and the tape of M' can be represented at the same time. □

Corollary 1. $\mathbf{P}^{\mathbf{NP}} \subseteq \mathbf{PMC}_{\mathcal{M}(-\mathrm{wn})}$. □

Instead of using membrane dissolution as in rule (23), we can use the object $\hat{q}'_!$ to produce $\oplus$:

$$[\hat{q}'_! \to \oplus]^\alpha_h \qquad \text{for } \alpha \in \{+,-\} \tag{26}$$

which ensures that the charge of h is positive instead of negative two steps later. The tape-objects are then sent out, one at a time, by using the following rules:

$$[1_i]^+_h \to [\,]^+_h\, 1_i \qquad \text{for } -(m-1) \le i \le m-1 \tag{27}$$

The timer t of the object $\tilde{q}_{!,t}$ has to be increased appropriately, in order to take into account the time needed to send out all the tape-objects. However, since the membrane where the simulation of M is non-elementary after the first query, rule (20) is now a weak non-elementary division rule. As a consequence, we have:

Theorem 5. *A deterministic polynomial-time Turing machine which asks $p(n)$ queries to an* **NP** *oracle on inputs of length n can be simulated by a uniform family of monodirectional P systems of depth $p(n)$ without dissolution rules.* □

Corollary 2. $\mathbf{P}^{\mathbf{NP}} \subseteq \mathbf{PMC}_{\mathcal{M}(-\mathrm{d})}$. □

In order to prove the converse of Theorem 2, we introduce an auxiliary complexity class (a variant of the class of optimisation problems **OptP** [3]).

Definition 3. *Define* **OrP** *to be the class of functions $f\colon \{0,1\}^\star \to \{0,1\}^\star$ having a polynomial-time nondeterministic Turing machine M such that, for all $x \in \{0,1\}^\star$, we have $f(x) = \bigvee M(x)$, where $M(x)$ denotes the set of possible output strings of M on input x, and $\bigvee$ denotes bitwise disjunction of strings; here we assume that the bitwise disjunction of strings of different lengths is performed by padding the shortest ones with zeros.*

The purpose of the class **OrP** is to capture a polynomial number of parallel **NP** queries with a single query to a function over binary strings.

Proposition 1. $\mathbf{P}^{\mathbf{NP}}_{\parallel} = \mathbf{P}^{\mathbf{OrP}[1]}$.

Proof. A polynomial number of parallel queries $y_1, \ldots, y_m$ to an oracle for $L \in \mathbf{NP}$ can be replaced by a single query to an oracle for the function $f(y_1, \ldots, y_m) = z_1 \cdots z_m$, where $z_i = 1$ if and only if $y_i \in L$. Let M be an **NP** machine deciding L, and let M' be the following nondeterministic machine: on input $y_1, \ldots, y_m$ simulate M on each y_i and record the corresponding output bit z_i; finally, output $z_1 \cdots z_m$. For all $1 \leq i \leq m$, if y_i is accepted by the oracle, then there exists a computation of M' such that $z_i = 1$: thus, by taking the bitwise disjunction of all possible output strings of M', we obtain the i-th bit of $f(y_1, \ldots, y_m)$; this proves that $f \in \mathbf{OrP}$. Notice that this proof requires the query strings $y_1, \ldots, y_m$ to be fixed in advance, i.e., the queries cannot be performed adaptively.

Vice versa, a single query to an oracle for $f \in \mathbf{OrP}$ with query string y can be replaced by the following polynomial number of parallel queries, one for each $1 \leq i \leq |f(y)|$: "is the i-th bit of $f(y)$ a 1?". These queries are in **NP**, since they can be answered by simulating an **OrP** machine M for f and selecting only its i-th output bit; the answer will be positive if and only if there exists a computation of M having a 1 as the i-th output bit, which (by definition of **OrP**) is equivalent to the i-th bit of $f(y)$ being 1. □

Simulating an **OrP** query by means of a P system is completely analogous to simulating an **NP** query, except that, instead of a single output bit, we have a polynomial number of them. These binary strings are automatically combined by bitwise disjunction when the tape-objects are sent out of the membrane simulating the nondeterministic Turing machine. Furthermore, since a single **OrP** query suffices to capture $\mathbf{P}_{\parallel}^{\mathbf{NP}}$, we obtain the following results:

Theorem 6. *A deterministic polynomial-time Turing machine which asks a polynomial number of* parallel *queries to an* **NP** *oracle on inputs of length n can be simulated by a uniform family of monodirectional P systems of depth 1 without dissolution (and, necessarily, without non-elementary division).* □

Corollary 3. $\mathbf{P}_{\parallel}^{\mathbf{NP}} \subseteq \mathbf{PMC}_{\mathcal{M}(-\mathrm{d},-\mathrm{wn})}$. □

5 Further results

The depth of the P systems of Theorems 4 and 5 can be asymptotically reduced by exploiting the equivalence of a logarithmic number of adaptive queries and a polynomial number of parallel queries [7, Theorem 17.7], formally $\mathbf{P}_{\parallel}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{NP}[\log n]}$. Suppose a deterministic polynomial-time Turing machine performs $p(n)$ sequential **NP** queries, and divide these queries into $\Theta(p(n)/\log n)$ blocks of $\Theta(\log n)$ queries. Each block can then be replaced by a polynomial number of parallel **NP** queries or, by Proposition 1, by a single **OrP** query. Hence, $p(n)$ sequential **NP** queries can be simulated by $\Theta(p(n)/\log n)$ sequential **OrP** queries, and each of the latter can be simulated by one level of depth in a P system:

Corollary 4. *A deterministic polynomial-time Turing machine which asks $p(n)$ queries to an* **NP** *oracle on inputs of length n can be simulated by a uniform family of monodirectional P systems of depth $\Theta(p(n)/\log n)$ without non-elementary division rules (resp., without division rules).* □

Theorem 3 can be sharpened by making the intra-level query parallelism explicit with **OrP** queries:

Corollary 5. *Let $\boldsymbol{\Pi}$ be a family of semi-uniform polynomial-time monodirectional P systems of depth $f(n)$. Then $\boldsymbol{\Pi}$ can be simulated by a polynomial-time deterministic Turing machine with $f(n)$ queries to an* **OrP** *oracle.* □

We can also prove that monodirectional families of P systems of *any* constant depth, even with dissolution and non-elementary division rules (in symbols $\mathcal{M}(\mathrm{O}(1))$), are always equivalent to families of depth *one* without dissolution and without non-elementary division (in symbols $\mathcal{M}(1, -\mathrm{d}, -\mathrm{wn})$), and thus only able to simulate parallel **NP** queries.

Theorem 7. $\mathbf{PMC}^{[\star]}_{\mathcal{M}(\mathrm{O}(1))} = \mathbf{PMC}^{[\star]}_{\mathcal{M}(1,-\mathrm{d},-\mathrm{wn})} = \mathbf{P}^{\mathbf{NP}}_{\parallel}$.

Proof. By Theorem 6, we already know that $\mathbf{P}^{\mathbf{NP}}_{\parallel} \subseteq \mathbf{PMC}_{\mathcal{M}(\mathrm{O}(1))}$, even when limited to depth 1; the inclusion $\mathbf{PMC}_{\mathcal{M}(\mathrm{O}(1))} \subseteq \mathbf{PMC}^{\star}_{\mathcal{M}(\mathrm{O}(1))}$ holds by definition. The inclusion $\mathbf{PMC}^{\star}_{\mathcal{M}(\mathrm{O}(1))} \subseteq \mathbf{P}^{\mathbf{NP}}_{\parallel}$ can be proved as follows. By Theorem 3, a family of P systems of constant depth k can be simulated in polynomial time by asking k sets (one per level) of $p(n)$ parallel queries, for some polynomial p. Each set of $p(n)$ parallel queries can be converted into $\Theta(\log n)$ sequential queries [7, Theorem 17.7], for a total of $k \times \Theta(\log n)$ sequential queries. These can be converted back into a polynomial number of parallel queries. □

Finally, observe that Theorem 3 also trivially holds for monodirectional P systems *without charges*. This implies a better upper bound than previously known [5] for a monodirectional variant of the P conjecture [9, Problem F], which states that P systems without charges and without non-elementary division characterise **P**.

6 Conclusions

In this paper we confirmed the importance of the direction of the information flow in P systems with active membranes with respect to their computing power. Indeed, when working in polynomial time and using only outward-bound communication, the corresponding complexity class decreases from **PSPACE** to $\mathbf{P}^{\mathbf{NP}}$, or from $\mathbf{P}^{\#\mathbf{P}}$ to $\mathbf{P}^{\mathbf{NP}}_{\parallel}$ when non-elementary division and dissolution rules are disallowed. It is interesting to notice that, unlike with other restrictions such as removing membrane division [10] or charges and dissolution [2], the resulting P systems are still more powerful than **P** (unless, of course, $\mathbf{P} = \mathbf{NP}$).

The role of strong non-elementary division (which is replaced in this paper by weak non-elementary division) in the absence of send-in rules is still unclear. Even if it provides a way to convey information from a parent membrane to its children, we do not know whether this is sufficient to altogether replace send-in communication while maintaining a polynomial run-time.

Finally, it would be interesting to investigate monodirectional P systems where the information flow is reversed, i.e., send-out communication and dissolution rules (as well as strong non-elementary division rules) are disallowed. A first issue to overcome is choosing an appropriate acceptance condition for the P systems, to replace sending out *yes* or *no* from the outermost membrane. The acceptance condition most similar "in spirit" to the original one is probably accepting (resp., rejecting) by having at least one *yes* (resp., *no*) object appear, either anywhere in the system, or inside a distinguished (and possibly dividing) membrane, during the last computation step; we also add the restriction that *yes* and *no* can never appear together, since giving the priority to one of them would allow us to solve **NP**-complete (or **coNP**-complete) problems "for free". Such monodirectional P systems appear to be very weak when working in polynomial time; indeed, even though exponentially many membranes can still be created by division, they can never communicate. Is **P** actually an upper bound to the class of problems they can solve?

References

1. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. Fundamenta Informaticae 58(2), 67–77 (2003)
2. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Nuñez, A., Romero-Campero, F.J.: Computational efficiency of dissolution rules in membrane systems. International Journal of Computer Mathematics 83(7), 593–611 (2006)
3. Krentel, M.W.: The complexity of optimization problems. Journal of Computer and System Sciences 36, 490–509 (1988)
4. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Membrane division, oracles, and the counting hierarchy. Fundamenta Informaticae (2015), in press
5. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Simulating elementary active membranes, with an application to the P conjecture. In: Gheorghe, M., Rozenberg, G., Sosík, P., Zandron, C. (eds.) Membrane Computing, 15th International Conference, CMC 2014, Lecture Notes in Computer Science, vol. 8961, pp. 284–299. Springer (2015)
6. Murphy, N., Woods, D.: Active membrane systems without charges and using only symmetric elementary division characterise P. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 8th International Workshop, WMC 2007. Lecture Notes in Computer Science, vol. 4860, pp. 367–384 (2007)
7. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1993)
8. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)

9. Păun, Gh.: Further twenty six open problems in membrane computing. In: Gutíerrez-Naranjo, M.A., Riscos-Nuñez, A., Romero-Campero, F.J., Sburlan, D. (eds.) Proceedings of the Third Brainstorming Week on Membrane Computing. pp. 249–262. Fénix Editora (2005)
10. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, pp. 289–301. Springer (2001)
11. Zandron, C., Leporati, A., Ferretti, C., Mauri, G., Pérez-Jiménez, M.J.: On the computational efficiency of polarizationless recognizer P systems with strong division and dissolution. Fundamenta Informaticae 87, 79–91 (2008)