Characterizing PSPACE with Shallow Non-Confluent P Systems

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

Dipartimento di informatica, Sistemistica e Comunicazione
Universit degli Studi di Milano-Bicocca,
Viale Sarca 336, 20126, Milan, Italy
{leporati, luca.manzoni, mauri, porreca, zandron}@disco.unimib.it

Summary. In P systems with active membranes, the question of understanding the power of non-confluence within a polynomial time bound is still an open problem. It is known that, for shallow P systems, that is, with only one level of nesting, non-confluence allows them to solve conjecturally harder problems than confluent P systems, thus reaching **PSPACE**. Here we show that **PSPACE** is not only a bound, but actually an exact characterization. Therefore, the power endowed by non-confluence to shallow P systems is equal to the power gained by *confluent* P systems when non-elementary membrane division and polynomial depth are allowed, thus suggesting a connection between the roles of non-confluence and nesting depth.

1 Introduction

While families of *confluent* recognizer P systems with active membranes with charges are known to characterize the complexity class **PSPACE** when working in polynomial time [9, 10], their computational power when the nesting level is constrained to one (i.e., only one level of membranes inside the outermost membrane, usually called *shallow* P systems) is reduced to the class $\mathbf{P}^{\#\mathbf{P}}$, which is conjecturally smaller [1]. While confluent P systems can make use of nondeterminism, they are constrained in returning the same result for all computations starting from the same initial configuration. However, by accepting when at least one computation accepts, like nondeterministic Turing Machines (TM) traditionally do, P systems can make use of the entire power of nondeterminism: uniform families of *non-confluent* recognizer P systems with active membranes with charges can solve **PSPACE**-complete problems even in the shallow case and even when send-in rules are disallowed (i.e., for monodirectional systems) [4]. Here we show that, in fact, **PSPACE** is a characterization of this kind of shallow non-confluent P systems when they work in polynomial time. This result shows that the complex relation between computational power, nesting depth, and monodirectionality present for

confluent P systems is absent in the non-confluent case. In particular, in the confluent case, systems with no nesting characterize **P** [11] whereas, additional nesting gives additional power [2] until reaching **PSPACE** when unlimited nesting is allowed [9, 10]. In the monodirectional case even unlimited nesting cannot escape $\mathbf{P}^{\mathbf{NP}}$, which is conjecturally smaller [3]. Non-confluent systems, on the other hand, characterize **NP** when there are no internal membranes [8], and immediately gain the full power of **PSPACE** with only one level of nesting. Furthermore, at least for shallow systems, this provides an exact characterization. It is therefore natural to ask what is the relation between the mechanisms that empower confluent P systems and the full power of non-confluence. Are the former ones only a way to simulate the latter?

2 Basic Notions

For an introduction to membrane computing and the related notions of formal language theory, we refer the reader to *The Oxford Handbook of Membrane Computing* [6]. Here we recall the formal definition of P systems with active membranes using only elementary division rules.

Definition 1. *A* P system with active membranes with elementary division rules *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 for the membranes;*
- μ *is a membrane structure (i.e., a rooted* unordered *tree, usually represented by nested brackets) consisting of* d *membranes labelled by elements of* Λ *in a one-to-one way;*
- $w_{h_1}, \ldots, w_{h_d}$*, with* $h_1, \ldots, h_d \in \Lambda$*, are multisets (finite sets whose elements have a multiplicity) of objects in* Γ*, describing the initial contents of the* d *regions of* μ*;*
- R *is a finite set of rules.*

Each membrane possesses, besides its label and position in μ, another attribute called *electrical charge*, which can be either neutral (0), positive (+) or negative (−) and is always neutral before the beginning of the computation.

The rules in R are of the following types:

(a) *Object evolution rules*, of the form $[a \to w]_h^\alpha$
They can be applied inside a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by the objects in w).

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

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

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

The instantaneous *configuration* of a membrane of label h consists of its charge α and the multiset w of objects it contains at a given time. It is denoted by $[w]_h^\alpha$. The *(full) configuration* $\mathcal{C}$ of a P system Π at a given time is a rooted, unordered tree. The root is a node corresponding to the external environment of Π, and has a single subtree corresponding to the current membrane structure of Π. Furthermore, the root is labelled by the multiset located in the environment, and the remaining nodes by the configurations $[w]_h^\alpha$ of the corresponding membranes. In the *initial configuration* of Π, the configurations of the membranes are $[w_{h_1}]_{h_1}^0, \ldots, [w_{h_d}]_{h_d}^0$.

A P system is *shallow* if it contains at most one level of membranes inside the outermost membrane. This means that all the membranes contained in the outermost membrane are elementary, i.e., they contain no other nested membrane.

A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules: inside each membrane, several evolution rules can be applied simultaneously.
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, or division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). Analogously, each membrane can only be subject to one communication or division rule (types (b)–(e)) per computation step; these rules will be called *blocking rules* in the rest of the paper. In other words, the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.
- In each computation step, all the chosen rules are applied simultaneously (in an atomic way). However, in order to clarify the operational semantics, each com-

putation step is conventionally described as a sequence of micro-steps whereby each membrane evolves only after their internal configuration (including, recursively, the configurations of the membrane substructures it contains) has been updated. In particular, 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, 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 $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ of configurations, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable from $\mathcal{C}_i$ via a single computation step, and no rules of Π are applicable in $\mathcal{C}_k$. A *non-halting* computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted.

P systems can be used as language *recognisers* by employing two distinguished objects yes and no: in this case we assume that all computations are halting, and that either one copy of object yes or one of object no is sent out from the outermost membrane, and only in the last computation step, in order to signal acceptance or rejection, respectively. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. In this paper we deal, however, with *non-confluent* P systems, where multiple computations can have different results and the overall result is established as for nondeterministic TM: it is acceptance iff an accepting computation exists [7].

In order to solve decision problems (or, equivalently, decide languages) over an alphabet Σ, we use *families* of recogniser P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. 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 [5].

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

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

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

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction

of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [5] for further details on the encoding of P systems.

In the following, we denote the class of problems solvable by polynomial-time uniform or semi-uniform families of non-confluent shallow P systems with active membranes with charges by $\mathbf{NPMC}^{[\star]}_{\mathcal{AM}(\text{depth-1},-\text{d},-\text{ne})}$, where $[\star]$ denotes optional semi-uniformity. If no restriction on the depth of the membrane structure is present, but both non-elementary division and dissolution rules are forbidden, then the corresponding class of problems is denoted by $\mathbf{NPMC}^{[\star]}_{\mathcal{AM}(-\text{d},-\text{ne})}$.

3 Nondeterministic Simulation with Oracles

Let $\boldsymbol{\Pi}$ be a semi-uniform family of non-confluent shallow recognizer P systems with active membranes with charges, and let H be the TM of the semi-uniformity condition of $\boldsymbol{\Pi}$. We are going to define a machine M working in polynomial *space* such that on input H and x Turing machine M accepts iff the P system $H(x) = \Pi_x$ of $\boldsymbol{\Pi}$ accepts in polynomial *time*. Notice that a single machine M suffices for all families of P systems. The machine associated with a specific family $\boldsymbol{\Pi}$ of P systems can be obtained by "hard-coding" the input H to M.

First of all, on input H and x, machine M simulates machine H with x as input to obtain a polynomial-size description of Π_x. To simplify the description of the procedure used by machine M to simulate Π_x, we will assume M to work as a nondeterministic polynomial-time TM with access to an oracle for a problem in $\mathbf{NPSPACE} = \mathbf{PSPACE}$. As the following result shows, both this nondeterministic behaviour and the oracle queries can still all be simulated using a polynomial-space deterministic TM.

Proposition 1. $\mathbf{NP}^{\mathbf{NPSPACE}} = \mathbf{PSPACE}$.

Proof. Clearly $\mathbf{NP}^{\mathbf{NPSPACE}} \supseteq \mathbf{PSPACE}$, hence only the opposite inclusion needs to be proved. Let N be a polynomial-time nondeterministic TM with access to an oracle for a language $L \in \mathbf{NPSPACE}$. Let D be a deterministic polynomial space TM built in the following way:

- D simulates N until a query is performed. This simulation, including the nondeterministic choices of N, can be performed in polynomial space by D, since $\mathbf{NP} \subseteq \mathbf{PSPACE}$.
- Since $L \in \mathbf{NPSPACE}$ and $\mathbf{NPSPACE} = \mathbf{PSPACE}$, there exists a deterministic polynomial space TM deciding L that can be simulated by D to answer any query performed by N while still using only a polynomial amount of space. Once a query has been answered, D can resume the simulation of N.

Since D can faithfully simulate N and its oracle queries, D can recognize the same language as N, thus showing that $\mathbf{NP}^{\mathbf{NPSPACE}} \subseteq \mathbf{PSPACE}$, as desired. □

We can now describe how the simulation of Π_x is carried on by M. In the following, we assume that the size of the input x is n, and that each computation of Π_x requires at most T time steps before halting and producing a result. By hypothesis T is polynomial with respect to n.

3.1 Simulation of the Outermost Membrane

The main idea of this construction is to simulate the evolution of the outermost membrane directly by means of a nondeterministic polynomial-time TM. All interactions with the internal membranes are performed via nondeterministic guesses. That is, for each communication rule and for each time step, the number of rules that are applied between the outermost and the inner membranes is guessed in a nondeterministic way. If yes has been sent out by the simulation of the outermost membrane, an oracle query is performed to check whether all performed interactions with the inner membranes were *consistent* with this information, that is, if a computation of the inner membranes able to perform the guessed interactions actually exists. If the query returns a positive answer, then a computation of the entire system actually producing yes exists. In any other case, the simulating machine rejects (since either an invalid simulation of the outermost membrane – and of the P system – was produced, or the simulation itself was correct but the simulated computation was a rejecting one).

To perform this construction we build a table $\mathcal{T}$ indexed by pairs of the form (r, t), where $r \in R$ is either a send-in rule from the outermost membrane to one of the internal membranes or a send-out rule from one of the internal membranes to the outermost membrane, and $t \in \{0, \ldots, T-1\}$ is a time step. The entry $\mathcal{T}(r, t)$ represent the number of times rule r has been applied at the time step t. It is important to notice that table $\mathcal{T}$ can be stored using a polynomial amount of space. In fact, the number of entries is limited by the size of R (which, by uniformity condition, is polynomial in the input size n), and by the number T of time steps needed for the P system to halt. We only need to prove that each entry $\mathcal{T}(r, t)$ can be stored in a polynomial amount of space.

Let $m \in \mathbb{N}$ be number of internal membranes in the initial configuration of Π_x. By the semantics of the rules of P systems, the number of objects sent in to internal membranes or sent out from them after t time steps cannot be greater than $m \times 2^t$, where the second multiplicative factor is the maximum number of membranes per label that can be obtained by membrane division in t time steps. Since this value is exponential in t, it can be represented by a polynomial number of bits with respect to $t \leq T$. Thus, each entry of $\mathcal{T}$ requires at most a polynomial amount of space with respect to n. We denote the maximum value attainable by an entry of $\mathcal{T}$ by K.

Apart from keeping track of the communication rules applied between the outermost and the internal membranes, we also need to assure that all rules are applied in a maximally parallel way. To do so, we define another table $\mathcal{U}$ indexed by pairs of the form (a, t) where $a \in \Gamma$ is an object type and $t \in \{0, \ldots, T-1\}$ is,

as before, a time step. The entry $\mathcal{U}(a,t)$ represents the number of objects of type a in the outermost membrane that had no rule applied to them at time t. Table $\mathcal{U}$ can, too, be stored in a polynomial amount of space.

The simulation procedure of the outermost membrane is detailed as Algorithm 1. There, label h always indicates the outermost membrane and the label k an internal membrane label, while $|w|_a$ denotes the number of instances of the object a inside the multiset w. The *applicability* of a rule refers, in the algorithm, to the fact that the indicated membrane must have the correct charge α and, if the rule is *blocking*, that the membrane has not already been used by another blocking rule in the same time step. For example, the condition on line 14 of Algorithm 1 is never verified once another send-out rule has been simulated in a previous iteration of the loop for the current time step.

Lines 1–3 perform the initialization of the algorithm, setting the initial content and charge of the outermost membrane and declaring the environment initially empty. The main simulation loop is performed in lines 4–29. Since the maximum number of time steps needed for Π_x to produce a result is T, the simulation loop is repeated at most T times. If the loop ends without having produced either yes on no in the environment while simultaneously halting, the simulation performed did not correspond to any actual computation of Π_x, thus a negative answer must be produced (line 30).

Lines 5–7 deal with the send-in rules from the outermost membrane to the inner membranes. Since the number of internal membranes where the rule r can be applied is not known, the number is nondeterministically chosen and is bounded by the maximum number of inner membranes and the number of objects of type a in the outermost membrane (line 6). The guessed number of internal membranes saved in table $\mathcal{T}$ and the effect of the rules on the multiset w is scheduled for application (line 7). Notice that, since the state of the internal membranes is not stored, this amounts to the removal of $\mathcal{T}(r,t)$ instances of objects of type a from w.

Lines 8–10 deal with send-out rules from the internal membranes to the outermost membrane. As before, since the configuration and number of the internal membranes is not known, the number of times this rule is applied is nondeterministically guessed (line 9), saved in table $\mathcal{T}$, and the appearance of the corresponding objects of type b in w is scheduled (line 10).

Lines 11–13 perform the simulation of the evolution rules inside the outermost membrane. Since the simulated system is non-confluent, the actual number of applications of each rule is guessed (line 12) before the actual effect of the rule applications are scheduled (line 13).

Lines 14–19 deal with the application of send-out rules from the outermost membrane to the environment. First of all, a nondeterministic guess is performed to decide whether the rule is actually applied (line 15). If so, then the actual effects of the rules are scheduled for application (lines 16–19).

The table $\mathcal{U}$ is then updated to memorize the number of objects that were not subjected to any rule (lines 20–21). This will be used during the query process

```
1  w ← initial multiset of the outermost membrane;
2  env ← ∅;
3  charge ← 0;
4  for t ← 0 to T − 1 do
5  |  for all applicable r = a [ ]_k^α → [b]_k^β do
6  |  |  T(r, t) ← guess (0, min(|w|_a, K));
7  |  |  mark T(r, t) instances of a for removal from w;
8  |  for r = [a]_k^α → [ ]_k^β b do
9  |  |  T(r, t) ← guess (0, K);
10 |  |  mark T(r, t) instances of b for insertion in w;
11 |  for all applicable r = [a → u]_h^α do
12 |  |  m ← guess (0, |w|_a);
13 |  |  mark m copies of u for addition to w and m copies of a for removal;
14 |  for all applicable r = [a]_h^α → [ ]_h^β b do
15 |  |  m ← guess (0, 1);
16 |  |  if m = 1 then
17 |  |  |  mark one copy of a for removal from w;
18 |  |  |  mark one copy of b for addition in env;
19 |  |  |  mark charge to be changed from α to β;
20 |  for a ∈ Γ do
21 |  |  U(a, t) ← number of instances of a in w not marked;
22 |  Apply modifications to w, env, and charge according to the markings;
23 |  if rule application was not maximally parallel then
24 |  |  reject;
25 |  if yes or no has been sent out in the environment then
26 |  |  if query (T, U, t) answer is positive and no further rules are applicable
   |  |    in the next time step then
27 |  |  |  accept or reject accordingly;
28 |  |  else
29 |  |  |  reject;
30 reject;
```

Algorithm 1: The nondeterministic algorithm that performs the simulation of the outermost membrane of Π_x.

to ensure that the send-in rules from the outermost membrane to the internal membranes were actually applied in a maximally parallel way.

All the scheduled modifications to the content and charge of the outermost membrane and to the environment are now executed (line 22). If there are irreconcilable problems in the maximally parallel application of the rules then a rejection is performed (lines 23–24). This happens when there were objects in the outermost membrane that were not selected to be sent-in into the internal membranes (this can be checked by looking at table $\mathcal{U}$), nor were they subject to applicable send-out or evolution rules.

Finally, if either yes or no appears in the environment (lines 25–29) then it is necessary to check whenever the guesses performed for the interaction with the

internal membranes were accurate and no further rules are applicable in the next time step in the outermost membrane (lines 26–29). If the answer to the query is positive and no further rules were actually applicable, then the simulation can either accept or reject accordingly (line 27). Otherwise, the simulation performed did not correspond to any actual computation of Π_x and we must reject (line 29).

Algorithm 1 can be executed in polynomial time by a nondeterministic TM with access to an oracle to perform the query procedure. In fact, both the outer loop and the inner loops are executed only a polynomial amount of times (either bounded by the time needed for Π_x to halt or by the number or rules in the system). All other operations, including checking the applicability of rules, can be performed in polynomial time given an efficient description of the configuration of the outermost membrane (in which the number of objects is stored in binary). Furthermore, all nondeterministic guesses are of a polynomial amount of bits.

3.2 Simulation of the Oracle

The query that is simulated by means of a nondeterministic machine working in polynomial space is the following one:

> Is there an halting computation of length t of the internal membranes consistent with the rule applications guessed?

To be able to answer this query in nondeterministic polynomial space the main idea is to simulate each membrane sequentially and keep track of the communication rules that are applied while comparing them with the ones guessed by the simulation of the outermost membrane. If division is applied then only the simulation of one of the dividing membranes is immediately carried out (as performing them all at the same time might require exponential – instead of polynomial – space) while the other membrane is pushed into a stack, thus performing a *depth-first simulation* of the membrane hierarchy. This ensures that a polynomial amount of space suffices: it the space needed to simulate one membrane, plus a stack in which the number of elements is at most T, one for each time step. This algorithm is similar to the deterministic one presented in [10], although with an explicit stack instead of a recursive definition, and the further difference that their algorithm was able to work for unbounded-depth system. The actual algorithm implemented to answer the query is presented in Algorithm 2.

Lines 1–3 perform the initial set-up, where a new stack S is filled with the configuration of all internal membranes at the initial time step, i.e., $t = 0$. In particular, for each membrane the multiset of objects contained, label, charge, and time step of the simulation are all pushed as an single record into S.

In the main loop of lines 4–32 the simulation of all internal membranes is performed one at a time. This loop is executed until the stack of membranes to be simulated is not empty, which might require an exponential amount of time.

Once a new membrane to be simulated *starting at time* t_{push} has been extracted (line 5) the simulation of the membrane proceeds up to time step t, which is given

```
1  S ← ∅ ;
2  for all internal membrane [w]_k^α in the initial configuration do
3  |  push_S (w, k, α, 0) ;
4  while S is not empty do
5  |  (w, k, charge, t_push) ← pop S ;
6  |  for t' ← t_push to t do
7  |  |  for r = [a]_k^α → [b]_k^β [c]_k^γ applicable do
8  |  |  |  m ← guess (0, 1);
9  |  |  |  if m = 1 then
10 |  |  |  |  mark a copy of a for removal, a copy of b for addition to w;
11 |  |  |  |  mark charge to be changed to β;
12 |  |  for r = a [ ]_k^α → [b]_k^β applicable do
13 |  |  |  m ← guess (0, 1);
14 |  |  |  if m = 1 then
15 |  |  |  |  T(r, t') ← T(r, t') − 1;
16 |  |  |  |  mark a copy of b for addition to w;
17 |  |  |  |  mark charge to be changed to β;
18 |  |  for r = [a]_k^α → [ ]_k^β b applicable do
19 |  |  |  m ← guess (0, 1);
20 |  |  |  if m = 1 then
21 |  |  |  |  T(r, t') ← T(r, t') − 1;
22 |  |  |  |  mark a copy of a for removal from w;
23 |  |  |  |  mark charge to be changed to β;
24 |  |  for r = [a → u]_k^α applicable do
25 |  |  |  m ← guess (0, |w|_a);
26 |  |  |  mark m copies of a for removal, m copies of u for addition to w;
27 |  |  apply marked modifications to w and charge;
28 |  |  if rule application was not maximally parallel then
29 |  |  |  reject;
30 |  |  if division was applied, push_S (w − {b} + {c}, k, γ, t');
31 |  if the current membrane has further applicable rules then
32 |  |  reject;
33 if each entry of T is 0 then
34 |  accept;
35 else
36 |  reject;
```

Algorithm 2: The nondeterministic polynomial space algorithm simulating the inner membranes of Π_x.

in input as part the query (loop of lines 6–30) and represents the time at which the simulation of the outermost membrane has suspended in order to perform the query.

In lines 7–11, for each applicable division rule, i.e., the correct object and charge are present and the membrane has not already been used by a blocking rule in this time step, a nondeterministic choice is performed (line 8) to decide if the rule is actually applied. If so (lines 7–11), then the modifications described by the first half of the right-hand-side of the rule are performed, while the other membrane resulting from the division will be pushed on the stack S at the end of the simulation of the current time step (line 30). This cannot be performed earlier since the rewriting rules are applied, by the semantics of rule application in P systems, before the division actually takes place.

The simulation of both send-in and send-out rules (lines 12–17 and lines 18–23, respectively) is performed similarly. Since we are working in a situation of non-confluence, even if a rule is applicable, in order to actually decide whether to apply it, a nondeterministic guess is performed (line 13 and line 19, respectively). In both cases the modifications to be performed to the membrane configuration are scheduled for later execution (lines 16–17 and lines 22–23, respectively). Since send-in and send-out are communication rules between the outermost membrane and the internal membranes, each time one of them is applied the value of $\mathcal{T}(r, t')$ is decremented. If, at the end of the simulation, the number of guessed applications and the real number of applications of the communication rules coincides, all entries $\mathcal{T}(r, t')$ should be 0 (at line 15 and line 21, respectively).

The application of evolution rules (lines 24–26), their effect being limited to the internal state of the membrane, is simpler. As usual, which rules are actually applied is determined by a nondeterministic choice (line 25).

Once all rule applications have been decided, the actual modifications to the state of the membrane are applied (line 27) and, if the rule application was not maximally parallel then the computation rejects (lines 28–29). This can be verified by checking if there still exist objects inside the membrane with applicable rules but no rule was applied to them, or if $\mathcal{U}(a, t')$ is positive for some $a \in \Gamma$ with an applicable send-in rule to the currently simulated membrane. Since $\mathcal{U}(a, t')$ indicates the number of objects that were available for the application of send-in from the outermost membrane but no internal membrane was available, such an inconsistency would denote that the simulation of the internal membranes had no correspondence to the already performed simulation of the outermost membrane.

If a division rule was applied, then the configuration of the second membrane resulting from division is pushed to the stack S (line 30). Here, an instance of the object b has been replaced by an instance of object c and the charge has been changed from β to γ to obtain from the current membrane a copy corresponding to the other one obtained by division.

Before proceeding with the simulation of another membrane, we check that after t steps the computation in this membrane has actually halted (lines 31–32). Otherwise the current computation must reject (line 32).

After the simulation of all internal membranes is finished, i.e., the stack was emptied, a check on the entries of $\mathcal{T}$ is performed. If all and every communication rule application guessed during the simulation of the outermost membrane was actually executed then all entries of $\mathcal{T}$ should be 0. A positive (resp., negative) value for $\mathcal{T}(r,t)$ denotes that less (resp., more) applications of rule r at time t were performed than the number that was guessed.

If at least one accepting computation of the machine simulating the oracle query exists then the answer to the query is positive. Furthermore, if there is a way to "glue" the simulation of the outermost membrane and of the internal membranes, then the result produced by Algorithm 1 was correct. Combining this simulation with the inverse simulation presented in [4], we can then state the main result of the paper.

Theorem 1. $\mathbf{PSPACE} = \mathbf{NPMC}^{[\star]}_{\mathcal{AM}(\text{depth-1},-\text{d},-\text{ne})}$. □

As long as no dissolution is allowed, the property of being elementary is a static one and, if no non-elementary division is present, the simulation of the outermost membrane can be extended to include all non-elementary membranes, allowing us to state the following result.

Corollary 1. $\mathbf{PSPACE} = \mathbf{NPMC}^{[\star]}_{\mathcal{AM}(-\text{d},-\text{ne})}$. □

4 Conclusions

We have shown that, differently from confluent P systems, monodirectionality and a restriction on the depth of the system to 1 (or, equivalently, the absence of both dissolution and non-elementary division) do not prevent non-confluent P systems from reaching **PSPACE** in polynomial time. It remains open to establish if this upper bound can be extended to membrane structures of higher (non-constant) depth where non-elementary division is allowed. Since both monodirectionality and nesting depth have a huge influence in the computational power of confluent systems, it would be worthwhile to understand why they do not provide an analogous increase to non-confluent systems. These features are usually employed by algorithms designed for confluent P systems to simulate the power of nondeterminism, so the question is: are they always useless when non-confluence is already present?

References

1. 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 (2014)

2. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Membrane division, oracles, and the counting hierarchy. Fundamenta Informaticae 138(1–2), 97–111 (2015)
3. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Monodirectional P systems. Natural Computing 15(4), 551–564 (2016)
4. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Shallow non-confluent P systems. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, 17th International Conference, CMC 2016. Lecture Notes in Computer Science, vol. 10105, pp. 307–316 (2017)
5. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
6. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
7. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–284 (2003)
8. Porreca, A.E., Mauri, G., Zandron, C.: Non-confluence in divisionless P systems with active membranes. Theoretical Computer Science 411(6), 878–887 (2010)
9. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
10. Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences 73(1), 137–152 (2007)
11. 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)