

International Journal of Foundations of Computer Science
 © World Scientific Publishing Company

P SYSTEMS WITH ACTIVE MEMBRANES WORKING IN POLYNOMIAL SPACE

ANTONIO E. PORRECA, ALBERTO LEPORATI,
 GIANCARLO MAURI, CLAUDIO ZANDRON

*Dipartimento di Informatica, Sistemistica e Comunicazione
 Università degli Studi di Milano-Bicocca
 Viale Sarca 336/14, 20126 Milano, Italy
 {porreca, leporati, mauri, zandron}@disco.unimib.it*

Received (Day Month Year)
 Accepted (Day Month Year)
 Communicated by (xxxxxxxxxx)

We prove that recognizer P systems with active membranes using polynomial space characterize the complexity class **PSPACE**. This result holds for both confluent and nonconfluent systems, and independently of the use of membrane division rules.

Keywords: Membrane computing; complexity theory; space complexity.

1991 Mathematics Subject Classification: 68Q10, 68Q15

1. Introduction

P systems with active membranes [2] are a variant of P systems where the membranes affect the applicability of rules during the computation. Furthermore, the membranes can also grow exponentially in number, via division rules, allowing us to solve computationally hard problems in polynomial time.

In this paper we prove that P systems with active membranes working in polynomial *space* solve exactly the problems in **PSPACE**, even when strong features such as division rules and nonconfluence are used. The proof technique is based on a variant of two previously published simulation algorithms [8, 9].

2. Preliminaries

We assume that the reader is familiar with basic terminology and results concerning P systems with active membranes (see [3], chapters 11–12 for a survey).

Definition 1. A P system with active membranes of the initial degree $d \geq 1$ is a tuple $\Pi = (\Gamma, \Lambda, \mu, w_1, \dots, w_d, R)$, where:

- Γ is a finite alphabet of symbols, also called objects;
- Λ is a finite set of labels for the membranes;

2 A.E. Porreca, A. Leporati, G. Mauri & C. Zandron

- μ is a membrane structure (i.e., a rooted unordered tree) consisting of d membranes enumerated by $1, \dots, d$; furthermore, each membrane is labeled by an element of Λ , not necessarily in a one-to-one way;
- w_1, \dots, w_d are strings over Γ , describing the initial multisets of objects placed in the d regions of μ ;
- R is a finite set of rules, whose types are described below.

Each membrane possesses a further attribute, named *polarization* or *electrical charge*, which is either neutral (represented by 0), positive (+) or negative (−) and it is assumed to be initially neutral.

The rules are applied in a *maximally parallel* way, and the result is computed bottom-up, i.e., before dividing or dissolving a membrane, the result of all the rules inside it must have already been computed. By applying the rules this way, we move from a configuration to the next in a series of computation steps (see [6]).

We can use families of P systems with active membranes as language recognizers, thus allowing us to solve decision problems, as follows. A *recognizer P system with active membranes* Π has an alphabet containing two distinguished objects YES and NO, used to signal acceptance and rejection respectively; every computation of Π is halting and exactly one object among YES, NO is sent out from the skin membrane during each computation. In order to decide a language $L \subseteq \Sigma^*$, we associate with each input string $x \in \Sigma^*$ a recognizer P system Π_x deciding the membership of x in L [4]; the mapping $x \mapsto \Pi_x$ is computed by a polynomial-time deterministic Turing machine, and the family $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^*\}$ is said to be *semi-uniform*. Instead, a family of P systems is *uniform* iff all strings x of the same length are mapped into a single recognizer P system $\Pi_{|x|}$, which then receives an input multiset encoding the specific string x . A recognizer P system Π_x is said to be *confluent* if all of its computations agree on the result, and *nonconfluent* if this is not necessarily true; in the latter case, we say that Π_x accepts iff an accepting computation exists.

A family $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^*\}$ of P systems is said to operate within space bound $f: \mathbb{N} \rightarrow \mathbb{N}$ iff, for each $x \in \Sigma^*$, the sum of the number of membranes and objects in each configuration of Π_x does not exceed $f(|x|)$. The polynomial space complexity classes for P systems of type \mathcal{D} (e.g., a restriction of P systems with active membranes) are denoted by $[\mathbf{N}]\mathbf{PMCSpace}_{\mathcal{D}}^{[\star]}$, where the optional \mathbf{N} denotes nonconfluence, and the optional \star semi-uniformity instead of uniformity. From the definitions, the inclusions $\mathbf{PMCSpace}_{\mathcal{D}} \subseteq \mathbf{NPMCSpace}_{\mathcal{D}}$ and $\mathbf{PMCSpace}_{\mathcal{D}}^{\star} \subseteq \mathbf{NPMCSpace}_{\mathcal{D}}^{\star}$ follow immediately [5].

For further technical details, we refer the reader to [4, 5, 6].

3. Simulation algorithm

In this section we describe a simulation algorithm for arbitrary nonconfluent recognizer P systems Π with active membranes, running on a nondeterministic random access machine N . We adopt Papadimitriou's model of RAM [1], that is, we assume that the arithmetic operations of addition and subtraction are carried out in con-

stant time, while multiplication requires linear time with respect to the number of bits. This model of RAM can be simulated by a nondeterministic Turing machine with a polynomial slowdown and linear increase in space complexity.

The description of the simulated P system Π is given as input to the algorithm; any reasonable encoding is admissible (e.g., with the membrane structure represented as a string of brackets), as long as the multisets are represented in *unary*.

The current configuration \mathcal{C} of Π is stored explicitly by N : the membrane structure is represented by a rooted tree, and each node of the tree represents a membrane. Each node of the tree contains the following information: the label $h \in \Lambda$ and the charge $\alpha \in \{-, 0, +\}$ of the membrane, a description of the multiset of objects contained in the region, and a list of children nodes, representing the membranes immediately inside it.

As the representation of the multiset, we use a k -tuple of integers encoded in binary, where in the i -th entry we store the multiplicity of the i -th object of the alphabet (this requires fixing an arbitrary total ordering of the alphabet in advance).

The set of rules of Π can be represented by a list of records, each one containing all the information required to apply the corresponding rule [8]; for instance, the record for a communication rule $[a]_h^\alpha \rightarrow []_h^\beta b$ contains the label h of the membrane, the initial and final charges α and β , and the names of the objects a and b .

The following is a high-level description of the simulation algorithm.

- A** For each rule in R , assign to it a nondeterministically chosen set of membranes and objects, to which the rule should be applied.
- B** Check if the assignment of membranes and objects to rules is indeed maximally parallel; if this is not the case, abort the simulation by rejecting.
- C** Apply the rules selected in step **A**, starting from the elementary membranes and going up towards the skin membrane.
- D** If either YES or NO were sent out from the skin membrane in step **C**, then halt and accept or reject accordingly; otherwise, jump to step **A**.

Step **A** can be further split into sub-steps. The idea is to consider the rules of Π one by one, and nondeterministically assign objects and membranes to it, but without immediately applying them. The sub-steps of **A** can be described as follows.

- A₁** Let R' be the set of currently unused rules; set $R' \leftarrow R$.
- A₂** If $R' = \emptyset$ then go to step **B**. Otherwise, pick a rule $r \in R'$.
- A₃** If $r = [a \rightarrow w]_h^\alpha$ then, for each membrane of the form $[]_h^\alpha$, nondeterministically choose an amount k of copies of object a to be rewritten into w ; this amount can be anywhere from 0 to the multiplicity of a in that particular membrane. Subtract k from the number of available copies of a in h .
- A₄** If $r = a []_h^\alpha \rightarrow [b]_h^\beta$ then, for each available membrane of the form $[]_h^\alpha$ having an available instance of a in the region immediately outside, nondeterministically choose whether to apply r and, in that case, assign those particular instances of a and h to r making them unavailable.

4 *A.E. Porreca, A. Leporati, G. Mauri & C. Zandron*

- A₅** If $r = [a]_h^\alpha \rightarrow []_h^\beta b$ or $r = [a]_h^\alpha \rightarrow b$ or $r = [a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$ then, for each available membrane of the form $[]_h^\alpha$ containing an available instance of a , nondeterministically choose whether to apply r and, in that case, assign those particular instances of a and h to r making them unavailable.
- A₆** If $r = [[]_{h_1}^+ \cdots []_{h_k}^+ []_{h_{k+1}}^- \cdots []_{h_n}^-]_h^\alpha \rightarrow [[]_{h_1}^\delta \cdots []_{h_k}^\delta]_h^\beta [[]_{h_{k+1}}^\varepsilon \cdots []_{h_n}^\varepsilon]_h^\gamma$ then, for each available membrane of the form $[]_h^\alpha$ containing the available membranes $[]_{h_1}^+, \dots, []_{h_k}^+, []_{h_{k+1}}^-, \dots, []_{h_n}^-$ and possibly some neutral available membranes, nondeterministically choose whether to apply r or not; if so, assign all the involved membranes to r making them unavailable.
- A₇** Set $R' \leftarrow R' - \{r\}$ and go back to step **A₂**.

In step **B** we check whether the nondeterministic assignment of objects and membranes to rules is maximally parallel. We simply check, for each membrane and object in the current configuration, if there exists a rule in R which is applicable to available objects; if this is the case, we can deduce that parallelism is not maximal, and abort the computation by rejecting (notice that adding rejecting computations to a nondeterministic device does not change its accepting behavior).

In Step **C** we actually apply the chosen rules. We traverse the membrane structure in depth-first order and, while visiting each membrane, we perform the following steps for each rule r associated with it in step **A**.

- C₁** For each $r = [a \rightarrow w]_h^\alpha$, remove k instances of a , where k is the multiplicity chosen for r in step **A₃**, and add k times the objects in w .
- C₂** For $r = a []_h^\alpha \rightarrow [b]_h^\beta$ remove an instance of a from the external region, add an instance of b to the internal one, and change the charge to β .
- C₃** For $r = [a]_h^\alpha \rightarrow []_h^\beta b$ remove an instance of a from the internal region, add an instance of b to the external region, and change the charge to β .
- C₄** For $r = [a]_h^\alpha \rightarrow b$ move all the objects from the internal region to the external one, replacing an instance of a with b , then, remove the membrane h from the current configuration; the children of h are adopted by its parent.
- C₅** For $r = [a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$ duplicate membrane h and its contents, replacing an instance of a by one of b on one side, and by an instance of c on the other; set the charges of the new membranes to β and γ respectively.
- C₆** For $r = [[]_{h_1}^+ \cdots []_{h_k}^+ []_{h_{k+1}}^- \cdots []_{h_n}^-]_h^\alpha \rightarrow [[]_{h_1}^\delta \cdots []_{h_k}^\delta]_h^\beta [[]_{h_{k+1}}^\varepsilon \cdots []_{h_n}^\varepsilon]_h^\gamma$ create a new instance of h , placing inside it all negative membranes h_{k+1}, \dots, h_n from the current membrane, and a *deep copy* of all neutral membranes inside h (i.e., including the substructure having them as the root, including its contents); finally, update the charges according to r .

Finally, step **D** can be easily implemented by keeping a Boolean flag, to be set when a communication rule sending out YES or NO from the skin is applied in step **C**.

The correctness of this simulation algorithm follows immediately from its description, since it is a straightforward implementation of the usual semantics of P systems with active membranes. The only point worth highlighting is that the

rejecting computations of step **B**, due to having chosen a non-maximal combination of rules, do not influence the overall result of the simulating machine, which accepts if and only if the simulated P system accepts.

Before analyzing the time and space required by the simulation algorithm, we prove two auxiliary lemmata.

Lemma 2. *Let Π be a nonconfluent P system with active membranes having a description of length m . Then, the number of membranes in any configuration of Π after t computation steps is bounded by $2^{tm+m \log m}$.*

Proof. Since the number of membranes that Π can generate depends on the shape of the initial membrane structure, we begin by choosing a “worst-case membrane structure” [7]. The initial number of membranes of Π is at most m ; clearly, each membrane structure of degree m , when viewed as a tree, is a subtree of the complete m -ary tree T_m of depth $m - 1$ (i.e., with m levels). Since this tree is *uniform*, that is, the number of children of each node only depends on its depth, it can be represented by a m -tuple of integers $T(k_0, \dots, k_{m-1})$ where k_i denotes the number of nodes on level i , and k_0 is always 1, since the root is unique. Hence, the initial membrane structure is contained in $T_m = T(1, m, m^2, \dots, m^{m-2}, m^{m-1})$.

Now suppose that, during each computation step, every possible membrane divides; for the sake of simplicity, also assume that nonelementary division rules cause the duplication of all the children (instead of separating positive and negative ones, and only duplicating the neutral ones). We consider this scenario only for the sake of finding an upper bound, since it cannot really occur in practice.

The result of the application of rules are computed in a bottom-up way: first the elementary membranes divide, and the resulting “intermediate” membrane structure is $T(1, m, m^2, \dots, m^{m-2}, 2m^{m-1})$. Then, the membranes of depth $m - 2$ divide; this also causes another doubling of those having depth $m - 1$, giving $T(1, m, m^2, \dots, 2m^{m-2}, 2^2m^{m-1})$. By repeating this process all the way to the level below the root (which does not divide) we obtain a structure which is a subtree of $T(1, 2m, 2^2m^2, \dots, 2^{m-2}m^{m-2}, 2^{m-1}m^{m-1})$, that is

$$T((2m)^0, (2m)^1, (2m)^2, \dots, (2m)^{m-2}, (2m)^{m-1})$$

Notice how this tree can be obtained by replacing m with $2m$ in the initial tree T_m : analogously, after another computation step we obtain the tree T_{2^2m} and, in general, after t steps we obtain the tree $T_{2^t m}$. Since this tree has m levels, the number of nodes is bounded by m times the number of leaves, i.e.,

$$(2^t m)^{m-1} \cdot m \leq (2^t m)^{m-1} \cdot 2^t m = (2^t m)^m = 2^{tm+m \log m}. \quad \square$$

We also prove an upper bound on the number of objects.

Lemma 3. *Let Π be a nonconfluent P system with active membranes having a description of length m . Then, the number of objects in any configuration of Π after t computation steps is bounded by $2^{O(t^2 m \log m)}$.*

6 *A.E. Porreca, A. Leporati, G. Mauri & C. Zandron*

Proof. The initial number of objects of Π , and in particular the initial number contained inside each membrane, is bounded by the length m of the whole encoding. The only way to increase this number, besides using membrane division, is to use evolution rules. The right-hand side of each evolution rule $[a \rightarrow w]_h^\alpha$ contains at most m objects (once again, because of the length of the encoding), hence the number of objects after step $i+1$ is at most m times the amount of step i . If no communication or dissolution rules are ever used, after t steps each membrane contains at most $m^{t+1} = 2^{(t+1)\log m}$ objects, for a total of $2^{tm+m\log m} \cdot 2^{(t+1)\log m} = 2^{O(t^2 m \log m)}$ objects in the whole configuration (using Lemma 2). This upper bound also holds when communication or dissolution rules are used, because these kinds of rules only move the objects around without increasing their number. \square

We are now able to prove that the simulation is at most exponentially slower.

Theorem 4. *Let Π be a nonconfluent P system with active membranes, running in time T and having a description of length m . Then, the simulation algorithm computes the same result as Π in time $2^{O(T^2 m \log m)}$.*

Proof. We analyze the complexity of each step of the simulation algorithm, applied during the simulation of step t of Π , beginning with the sub-steps of **A**.

- In step **A**₁ we copy the set of rules in time $O(m)$.
- Step **A**₂ consists of checking if R' is empty and going to step **B**, which can be done in constant time; a nondeterministic choice of a rule involves scanning the set R' , which requires $O(m)$ time.
- In **A**₃ we traverse the whole membrane structure to find the membranes of the form $[]_h^\alpha$; for each of those (assume they all have this form for the sake of argument) we choose a number of objects to be rewritten: this requires a linear number of steps with respect to the number of bits used to store the multiplicities of the objects, that is $O(t^2 m \log m)$ by Lemma 3. Since by Lemma 2 the number of membranes is at most $2^{tm+m\log m}$, this step requires a total time bounded by $2^{tm+m\log m} \cdot O(t^2 m \log m)$, which is $2^{O(tm+m\log m)}$.
- Steps **A**₄ and **A**₅ also require traversing the membrane structure, but checking whether a rule is applicable and choosing whether to actually apply it only requires constant time. Hence these steps require time proportional to the number of membranes, which is also bounded by $2^{O(tm+m\log m)}$.
- While traversing the membrane structure in **A**₆, we need to inspect all the children of the current membrane in order to establish whether the nonelementary division rule can be applied. The number of children is bounded by $2^{tm+m\log m}$; thus, this step also requires $2^{O(tm+m\log m)}$ time.
- Finally, removing the selected rule from R' and going back to **A**₂ in step **A**₇ requires at most $O(m)$ time.

The loop consisting of steps \mathbf{A}_2 – \mathbf{A}_7 is executed once for each of the $O(m)$ rules; hence, the total time required for executing step \mathbf{A} (when simulating step t of Π) is $O(m) \cdot 2^{O(tm+m \log m)}$, which is still $2^{O(tm+m \log m)}$.

The analysis for step \mathbf{B} is very similar; the only difference is that we need to iterate through the different kinds of object in order to establish whether an evolution rule could be applied. However, this only adds a smaller term to the exponent, and the time remains $2^{O(tm+m \log m)}$.

The cost of the sub-steps of step \mathbf{C} are computed as follows.

- In \mathbf{C}_1 we perform a number of multiplications bounded by the size $O(m)$ of the alphabet of Π , and the size of the operands is bounded by $O(t^2 m \log m)$; the total time is thus $O(t^2 m^2 \log m)$.
- Steps \mathbf{C}_2 and \mathbf{C}_3 only require constant time.
- In \mathbf{C}_4 we need to move all the objects of the dissolving membrane to its parent: this requires $O(m)$ constant-time additions. We also need to move all its children, which are at most $2^{O(tm+m \log m)}$.
- Step \mathbf{C}_5 requires us to create a new membrane and duplicate the contents of the dividing one; duplicating the multiset requires $O(m)$ time.
- Finally, in \mathbf{C}_6 we duplicate a whole substructure of membranes with its contents. By Lemmata 2 and 3, the size of this structure is bounded by $2^{tm+m \log m} \cdot 2^{O(t^2 m \log m)}$, which is $2^{O(t^2 m \log m)}$.

One step among \mathbf{C}_1 – \mathbf{C}_6 has to be executed for each membrane and each rule; since the most expensive one is \mathbf{C}_6 , we can bound the total time required by step \mathbf{C} by

$$O(m) \cdot 2^{tm+m \log m} \cdot 2^{O(t^2 m \log m)} \cdot 2^{O(t^2 m \log m)} = 2^{O(t^2 m \log m)}$$

The last step of the simulation algorithm is \mathbf{D} , and it requires only constant time.

This proves that executing steps \mathbf{A} – \mathbf{D} in order to simulate step t of Π requires $2^{O(t^2 m \log m)}$ time, due to step \mathbf{D} . Simulating all T steps of Π requires summing this amount for $1 \leq t \leq T$; this sum is clearly bounded by T times the cost of the last step, that is, by $2^{O(T^2 m \log m)}$ time. \square

Nevertheless, the space required by the simulation is only polynomially larger.

Theorem 5. *Let Π be a nonconfluent P system with active membranes, running in space S and having a description of length m . Then, the simulation algorithm computes the same result as Π using space $O(S \log m)$.*

Proof. Explicitly storing the current configuration of Π as described above requires at most the same space as Π (and much less in some cases, since we store the multiplicity of objects in binary instead of unary). However, our simulation also requires storing the labels of the membranes, which do not occupy space in Π . Since Π initially contains at most m membranes, $\log m$ bits are sufficient to represent the labels. Hence, the configuration of Π can be stored in space $O(S \log m)$.

The auxiliary data structures needed by the simulation algorithm do not exceed this amount of space: the largest one is a stack required to perform a depth-first search of the membrane structure, and the number of items on the stack is bounded by the depth of the structure. \square

The analysis of the simulation algorithm allows us to prove our main result.

Theorem 6. *Let \mathcal{D} be a class of P systems with active membranes using at least communication rules. Then $[\mathbf{N}]\mathbf{PMCSpace}_{\mathcal{D}}^{[*]} = \mathbf{PSPACE}$, where $[\mathbf{N}]$ denotes optional nonconfluence, and $[*]$ optional semi-uniformity.*

Proof. The inclusion $\mathbf{PSPACE} \subseteq \mathbf{PMCSpace}_{\mathcal{D}}$ is proved in [6]. The inclusion $\mathbf{NPMCSpace}_{\mathcal{D}}^* \subseteq \mathbf{PSPACE}$ is proved by first performing the polynomial-time construction $x \mapsto \Pi_x$, then using the simulation algorithm on Π_x : this requires polynomial nondeterministic space by Theorem 5, which can be reduced to polynomial deterministic space by using Savitch's theorem [1]. The remaining classes mentioned above are located between $\mathbf{PMCSpace}_{\mathcal{D}}$ and $\mathbf{NPMCSpace}_{\mathcal{D}}^*$. \square

4. Conclusions

We proved that P systems with active membranes can be simulated by Turing machines with only a polynomial increase in space complexity. By combining this result with the ability of P systems to solve \mathbf{PSPACE} -complete problems in polynomial space, we obtained a characterization of \mathbf{PSPACE} in terms of membrane systems.

An interesting research direction involves searching for analogous results for P systems with active membranes working in logarithmic or exponential space.

References

- [1] C. H. Papadimitriou, *Computational Complexity* (Addison-Wesley, 1993).
- [2] G. Păun, P systems with active membranes: Attacking NP-complete problems, *Journal of Automata, Languages and Combinatorics*, **6**(1) (2001) 75–90.
- [3] G. Păun, G. Rozenberg and A. Salomaa (eds.), *The Oxford Handbook of Membrane Computing* (Oxford University Press, 2010).
- [4] M. J. Pérez-Jiménez, A. Romero-iménez and F. Sancho-Caparrini, Complexity classes in models of cellular computing with membranes, *Natural Comp.* **2**(3) (2003) 265–285.
- [5] A. E. Porreca, A. Leporati, G. Mauri and C. Zandron, Introducing a space complexity measure for P systems, *International Journal of Computers, Communications & Control* **4**(3) (2009) 301–310.
- [6] A. E. Porreca, A. Leporati, G. Mauri and C. Zandron, P systems with active membranes: Trading time for space. *Natural Computing*, to appear.
- [7] A. E. Porreca, A. Leporati and C. Zandron, On a powerful class of non-universal P systems with active membranes, *Proceedings of DLT 2010, LNCS*, to appear.
- [8] A. E. Porreca, G. Mauri and C. Zandron, Complexity classes for membrane systems, *RAIRO Theoretical Informatics and Applications* **40**(2) (2006) 141–162.
- [9] A. E. Porreca, G. Mauri and C. Zandron, Non-confluence in divisionless P systems with active membranes, *Theoretical Computer Science* **411**(6) (2010) 878–887.