Journal of Automata, Languages and Combinatorics **u** (v) w, x–y

A GAP IN THE SPACE HIERARCHY OF P SYSTEMS WITH ACTIVE MEMBRANES

Alberto Leporati, 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
e-mail: {leporati,mauri,porreca,zandron}@disco.unimib.it

ABSTRACT

We prove that uniform families of P systems with active membranes using logarithmic workspace, in addition to a polynomial number of read-only input objects, can efficiently simulate polynomial-space Turing machines, and thus characterise the complexity class **PSPACE**. Since **PSPACE** was already known to be characterised by P systems working in *polynomial* space, this theorem implies, perhaps counter-intuitively, that the latter systems are exponentially wasteful, and a large part of the computation can be carried out by P systems just by moving the objects between their regions, without the need to rewrite them. This result also provides the first scenario where P systems are not equivalent to Turing machines with respect to their space complexity.

Keywords: membrane computing, P systems with active membranes, computational complexity, space complexity

1. Introduction

P systems with active membranes [4], a computing model abstracting the operation of biological cells, have inspired a rich literature investigating their computational properties from a complexity-theoretic standpoint [5]. Their ability to solve **NP**-hard problems in polynomial time by generating exponentially many membranes via division, which then evolve in parallel, provides an interesting trade-off between time and space complexity.

Although P systems seem to be exponentially faster than Turing machines when solving **NP** or even **PSPACE**-complete problems [2], the two devices have instead been proved to be equivalent up to a polynomial from the space complexity point of view, assuming the available space is at least polynomial [1]. Here the space requirements of P systems are measured as the sum of the number of membranes and objects occurring in the largest configuration reached during computations [6].

It is also possible to analyse P systems working in sublinear space by requiring the input objects to be operated upon in a read-only fashion, as in multitape Turing machines, by only measuring the space due to the membrane structure and the

non-input objects. Under these assumptions, logarithmic-space P systems have been proved to be able to simulate Turing machines working in logarithmic space [8].

However, this result felt somewhat unsatisfactory, since logarithmic-space Turing machines can only generate polynomially many configurations, whereas P systems working in logarithmic space have *exponentially* many potential ones. Indeed, n distinct input objects can, in principle, be arbitrarily partitioned in $\log n$ different regions of the P system. This can be trivially performed by applying the rules nondeterministically; the open question was establishing whether those exponentially many configurations can actually be reached in a *confluent* way. In this paper we prove that this is indeed the case, and that the resulting control mechanism can be used to simulate polynomial-space Turing machines by using only logarithmic auxiliary space, thus characterising **PSPACE**. This is the first case where the space complexity of P systems and that of Turing machines differ by an exponential amount. Since **PSPACE** had already been proved to be characterised by *polynomial*-space P systems, these results also highlight a gap in the hierarchy of space complexity classes for P systems: super-polynomial space is required in order to exceed the computational power of logarithmic space.

The paper is organised as follows: in Section 2 we recall the definitions of P systems with active membranes and their complexity classes. In Section 3 we illustrate how Turing machine configurations can be encoded as configurations of P systems, whereas in Section 4 we show how P systems can simulate the transition functions of Turing machines. In Section 5 we discuss our result and we describe some open issues.

2. Definitions

Here we recall the basic definition of P systems with active membranes, while at the same time introducing an input alphabet with specific restrictions.

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

- Γ is an alphabet, i.e., a finite non-empty set of symbols, called *objects*;
- Δ is another alphabet, disjoint from Γ, called the *input alphabet*;
- 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 enumerated by $1, \ldots, d$; furthermore, each membrane is labeled by an element of Λ in a one-to-one way;
- $w_1, \ldots, w_d$ are strings over Γ, describing the initial multisets of objects placed in the d regions of μ;
- R is a finite set of rules over $\Gamma \cup \Delta$.

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.

A description of the available kinds of rules follows. This description differs from the original definition [4] only in that new input objects may not be created during the computation.

- *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 of objects occurring in h and replaced by the objects in w). At most one input object $b \in \Delta$ may appear in w, and only if it also appears on the left-hand side of the rule (i.e., if $b = a$). An object a can be deleted by letting $w = \epsilon$, denoting the empty multiset.
- *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. If $b \in \Delta$ then $a = b$ must hold.
- *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 is changed to β. If $b \in \Delta$ then $a = b$ must hold.

The original definition [4] also includes *dissolution* and *division rules*; however, in this paper we will not need them.

Each instantaneous configuration of a P system with active membranes is described by the current membrane structure, including the electrical charges, together with the multisets of objects located in the corresponding regions. 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. This does not apply to evolution rules, as inside each membrane several of them can be applied simultaneously.
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution or communication rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). The same principle applies to each membrane that can be involved in communication rules. 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 as the result of 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 computation step is conventionally described as a sequence of micro-steps as

follows. First, all evolution rules are applied inside the elementary membranes, followed by all communication rules involving the membranes themselves; this process is then repeated to the membranes containing them, and so on towards the root (outermost membrane). In other words, the membranes evolve only after their internal configuration has been updated.

- Any object sent out from the outermost membrane cannot re-enter the system.

A *halting computation* of the P system Π is a finite sequence of configurations $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, 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 can be applied anymore in $\mathcal{C}_k$. A *non-halting* computation $\vec{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted.

P systems can be used as *recognisers* by employing a distinguished input membrane and two distinguished objects *yes* and *no*; exactly one of these must be sent out from the outermost membrane during each computation, in order to signal acceptance or rejection, respectively; we also assume that all computations are halting. All P systems we will consider in this paper are *confluent*, that is, all computations starting from the same initial configuration are accepting, or all are rejecting.

In order to solve decision problems (i.e., decide languages over an alphabet Σ), we use *families* of recogniser P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for each input length; in this paper we use a logarithmic space uniformity condition [3].

Definition 2 A family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be $(\mathbf{L}, \mathbf{L})$-*uniform* if the mapping $x \mapsto \Pi_x$ can be computed by two deterministic logarithmic-space Turing machines F (for "family") and E (for "encoding") as follows:

- The machine F computes the mapping $1^n \mapsto \Pi_n$, where n is the length of the input x and Π_n is a common P system for all inputs of length n, containing a distinguished input membrane.
- The machine E computes the mapping $x \mapsto w_x$, where w_x is a multiset encoding the specific input x.
- Finally, Π_x is simply Π_n with w_x added to the multiset placed inside its input membrane.

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 [3] for further details on the encoding of P systems.

Finally, we describe how space complexity for families of recogniser P systems is measured, and the related complexity classes. The following definition differs from

the standard one [6] in one aspect: the input objects do not contribute to the size of the configuration of a P system. This way, only the actual working space of the P system is measured, and P systems working in sublinear space may be analysed.

Definition 3 Let $\mathcal{C}$ be a configuration of a recognizer P system Π. The *size* $|\mathcal{C}|$ *of* $\mathcal{C}$ is defined as the sum of the number of membranes in the membrane structure and *the total number of objects in* Γ *(i.e., the non-input objects) they contain.* If $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a computation of Π, then the *space required by* $\vec{\mathcal{C}}$ is defined as $|\vec{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\}$. The *space required by* Π itself is then defined as $|\Pi| = \sup\{|\vec{\mathcal{C}}| : \vec{\mathcal{C}} \text{ is a computation of } \Pi\}$. Finally, let $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recogniser P systems, and let $f\colon \mathbb{N} \to \mathbb{N}$. We say that $\mathbf{\Pi}$ *operates within space bound* f iff $|\Pi_x| \leq f(|x|)$ for each $x \in \Sigma^\star$.

By $(\mathbf{L}, \mathbf{L})$-$\mathbf{LMCSPACE}_{\mathcal{AM}}$ (resp., $(\mathbf{L}, \mathbf{L})$-$\mathbf{PMCSPACE}_{\mathcal{AM}}$) we denote the class of languages which can be decided by $(\mathbf{L}, \mathbf{L})$-uniform families of confluent recognizer P systems with active membranes working in logarithmic (resp., polynomial) space. We may also define the complexity classes $(\mathbf{L}, \mathbf{L})$-$\mathbf{LMCSPACE}_{\mathcal{D}}$ and $(\mathbf{L}, \mathbf{L})$-$\mathbf{PMCSPACE}_{\mathcal{D}}$ for restricted subclasses $\mathcal{D} \subseteq \mathcal{AM}$, for instance P systems with active membranes using only object evolution and communication rules.

3. P Systems Simulating Logarithmic-Space Turing Machines

Let M be a single-tape, deterministic Turing machine working in polynomial space $s(n)$. Let Q be the set of states of M, including the initial state s; we denote by $\Sigma' = \Sigma \cup \{\sqcup\}$ the tape alphabet, which includes the blank symbol $\sqcup \notin \Sigma$. Finally, let $\delta\colon Q \times \Sigma' \rightharpoonup Q \times \Sigma' \times \{-1, 0, +1\}$ be the (partial) transition function of M, which we assume to be undefined on (q, σ) if and only if q is a final state. We describe a uniform family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ simulating M in logarithmic space.

Let $x \in \Sigma^n$ be an input string, and let $m = \lceil \log s(n) \rceil$ be the minimum number of bits needed in order to represent the tape cell indices $0, \ldots, s(n)-1$ in binary notation. The P system Π_n, associated with the input length n and computed as $F(1^n)$, has a membrane structure consisting of an external membrane h and, for each symbol σ of the tape alphabet Σ' of M, the following set of membranes, linearly nested and listed from the outside in:

- a *symbol-membrane* labelled by σ;
- a *query-membrane* labelled by $?_\sigma$;
- for each $j \in \{0, \ldots, m-1\}$, a membrane labelled by j_σ.

Furthermore, the object z_0 is located inside the outermost region h.

The encoding of x, computed as $E(x)$, consists of a multiset w_x — actually, in this particular case, a set — of objects describing the tape of M in its initial configuration on input x. These objects are the symbols of x subscripted by their position $0, \ldots, n-1$ in x, together with the $s(n) - n$ blank objects $\sqcup_n, \ldots, \sqcup_{s(n)-1}$. In what follows, these objects will be referred to as the *input objects*, and they will never be rewritten. The

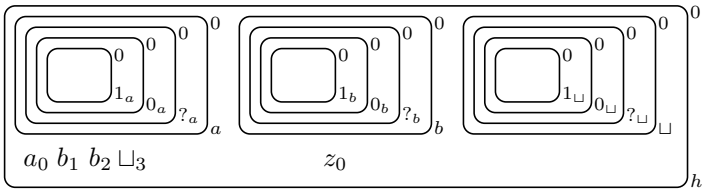

Figure 1: Initial configuration of the P system Π_3, simulating a Turing machine M, having tape alphabet $\Sigma' = \{a, b, \sqcup\}$, working in space $s(n) = n + 1 = 4$ on the input *abb*.

multiset w_x is placed inside the outermost membrane h, which is thus considered the input membrane of Π_n (and also of Π_x). Figure 1 depicts an example.

During the first computation steps of Π_x, each input object σ_i is moved to the corresponding innermost membrane $(m-1)_\sigma$ via the following communication rules:

$$\sigma_i \, [\,]^0_\sigma \to [\sigma_i]^0_\sigma \qquad \text{for } \sigma \in \Sigma',\ 0 \le i < s(n) \tag{1}$$

$$\tau_i \, [\,]^0_{?_\sigma} \to [\tau_i]^0_{?_\sigma} \qquad \text{for } \sigma, \tau \in \Sigma',\ 0 \le i < s(n) \tag{2}$$

$$\tau_i \, [\,]^0_{j_\sigma} \to [\tau_i]^0_{j_\sigma} \qquad \text{for } \sigma, \tau \in \Sigma',\ 0 \le i < s(n),\ 0 \le j < m \tag{3}$$

Note that rules (2) and (3) send into the innermost membrane $(m-1)_\sigma$ *any* input object τ_i which has entered membrane σ. This behavior will be exploited later, when simulating a computation step of M, where for each input object τ_i we will consider only its subscript i, disregarding the value of $\tau \in \Sigma'$.

Since only one communication rule per membrane may be applied during each computation step of Π_x, in the worst case (when all input objects are of the form σ_i for the same $\sigma \in \Sigma'$) all input objects will have reached the innermost membranes after at most $\ell = (2 + m) + (s(n) - 1)$ computation steps. Although this is not the minimal number of steps needed to let the input symbols reach the innermost membranes, here and in what follows we prefer to use coarse upper bounds in order to keep the various phases of computation well separated and simplify their exposition as much as possible.

While the above rules are applied, the object z_0 "waits" by incrementing its subscript up to ℓ, thus realizing a *counter*:

$$[z_t \to z_{t+1}]^0_h \qquad \text{for } 0 \le t < \ell \tag{4}$$

The object z_ℓ then produces $|\Sigma'|$ copies of the object $+$, while rewriting itself into the auxiliary object z':

$$[z_\ell \to z' \underbrace{+ \cdots +}_{|\Sigma'| \text{ copies}}]^0_h \tag{5}$$

The objects $+$ set the charges of the membranes corresponding to symbols $\sigma \in \Sigma'$ to positive by parallel communication rules. While doing so they are rewritten as

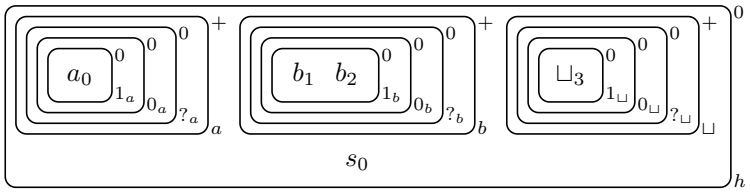

Figure 2: Configuration of Π_x (from Figure 1) encoding the initial configuration of M on input $x = aab$ and using $s(|x|) = 4$ tape cells.

the "junk" objects #, which are subsequently deleted by means of evolution rules (7), rewriting them to the empty multiset ϵ:

$$+\ [\ ]^0_\sigma \to [\#]^+_\sigma \qquad \text{for } \sigma \in \Sigma' \tag{6}$$

$$[\# \to \epsilon]^\alpha_\sigma \qquad \text{for } \sigma \in \Sigma',\ \alpha \in \{-, 0, +\} \tag{7}$$

In the mean time, the object z' is first rewritten into another auxiliary object z'' and finally into s_0, where s is the initial state of M and its subscript 0 indicates the initial position of the tape head:

$$[z' \to z'']^0_h \tag{8}$$

$$[z'' \to s_0]^0_h \tag{9}$$

The configuration reached by Π_x now encodes the initial configuration of M, as shown in Figure 2.

In general, an arbitrary configuration of M on input x is encoded by a configuration of Π_x as follows (see Figure 3):

- the outermost membrane h contains the *state-object* q_i, where q is the current state of M and $i \in \{0, \ldots, s(n) - 1\}$ is the current position of the tape head;
- for each $\sigma \in \Sigma'$, membrane $(m-1)_\sigma$ contains the input object τ_i for some $\tau \in \Sigma'$ if and only if the i-th tape cell of M, counting from 0, contains the symbol σ. Notice that, as stated above, when representing a generic configuration of M we disregard the value of $\tau \in \Sigma'$ in the input object τ_i. The only meaningful information are its subscript i, denoting a position on the tape of M, and the substructure σ that contains τ_i, denoting the symbol currently present in the i-th tape cell. As an example, in Figure 3 the symbol a under the tape head (position 2) is represented as the occurrence of object b_2 into membrane 1_a. We adopted this encoding because, according to Definition 2, the input objects must be all located in the input membrane in the initial configuration of Π_x (hence they must each encode both a symbol and a position on the tape), and they are never rewritten (since otherwise we would have to count them when measuring the space used during the computation);
- all the other membranes are empty;

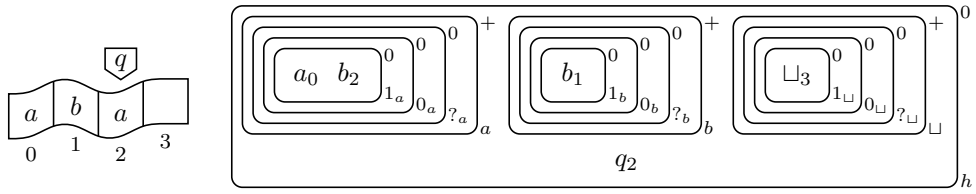

Figure 3: A configuration of M (the same machine as in Figure 1) and the corresponding configuration of the P system Π_x simulating it. The presence of b_2 inside membrane 1_a indicates that tape cell 2 of M contains the symbol a.

- all membranes are neutrally charged, except those labelled by $\sigma \in \Sigma'$, which are positively charged.

4. Simulating a Computation Step

The simulation of a computation step of M by Π_x, starting from a configuration corresponding to one of M, is directed by the state-object q_i. As stated above, this encodes only the current state of M and the position of the head on the tape; hence, it is necessary to identify the actual symbol occurring in that tape position. We have thus to query each membrane substructure, by encoding in binary the tape position i on the electrical charges of membranes $0_\sigma, \ldots, (m-1)_\sigma$. Such membranes then operate as filters for the objects contained in the substructures, letting out only the symbol whose subscript is i.

In order to do so, the object q_i is first rewritten into $|\Sigma'|$ copies of the auxiliary symbol q'_i, one for each membrane substructure of Π_x:

$$[q_i \to \underbrace{q'_i \cdots q'_i}_{|\Sigma'| \text{ copies}}]^0_h \qquad \text{for } q \in Q,\ 0 \le i < s(n) \tag{10}$$

The objects q'_i enter the symbol-membranes in parallel and then the corresponding query-membranes, setting the charges of the latter membranes to positive (meaning that a query is being performed):

$$q'_i\ [\ ]^+_\sigma \to [q'_i]^+_\sigma \qquad \text{for } \sigma \in \Sigma',\ q \in Q,\ 0 \le i < s(n) \tag{11}$$

$$q'_i\ [\ ]^0_{?_\sigma} \to [q'_i]^+_{?_\sigma} \qquad \text{for } \sigma \in \Sigma',\ q \in Q,\ 0 \le i < s(n) \tag{12}$$

The objects q'_i then traverse the membranes $0_\sigma, \ldots, (m-1)_\sigma$ while changing their charges so that they represent the bits of i, from the least to the most significant one, where a positive charge is interpreted as 1, and a negative charge as 0. For instance, the charges of $[\,[\,[\,]^-_{2_\sigma}]^+_{1_\sigma}]^+_{0_\sigma}$ encode the binary number 011_2 (that is, decimal 3). This is accomplished by the following rules:

$$q'_i\ [\ ]^0_{j_\sigma} \to [q'_i]^\alpha_{j_\sigma} \qquad \text{for } \sigma \in \Sigma',\ q \in Q,\ 0 \le i < s(n),\ 0 \le j < m \tag{13}$$

where α is $-$ if the j-th bit of i is 0, and α is $+$ if it is 1.

The membranes j_σ now behave as "filters" for the input objects τ_k occurring in the $(m-1)_\sigma$ membranes: these are sent out from each membrane j_σ if and only if the j-th bit of k corresponds to the charge of j_σ. By applying the following rules, exactly one object τ_k will reach a query-membrane, namely, the object with $k = i$:

$$[\tau_k]^\alpha_{j_\sigma} \to [\ ]^\alpha_{j_\sigma}\ \tau_k \qquad \text{for } \sigma, \tau \in \Sigma',\ 0 \le k < s(n),\ 0 \le j < m \tag{14}$$

where α is $-$ if the j-th bit of k is 0, and α is $+$ if it is 1.

When the object τ_i reaches a query-membrane, it is sent out while changing the charge to negative, in order to signal that the object sought was located in that particular membrane substructure:

$$[\tau_i]^+_{?_\sigma} \to [\ ]^-_{?_\sigma}\ \tau_i \qquad \text{for } \sigma, \tau \in \Sigma',\ 0 \le i < s(n) \tag{15}$$

Finally, the object τ_i is sent out from the symbol-membrane:

$$[\tau_i]^+_\sigma \to [\ ]^+_\sigma\ \tau_i \qquad \text{for } \sigma, \tau \in \Sigma',\ 0 \le i < s(n) \tag{16}$$

Since there are $s(n)$ input objects, each traversing at most $m+2$ membranes, the object τ_i reaches the outermost membrane h after at most $s(n)+m+2$ steps[1].

While the input objects are "filtered out", the state-objects q'_i wait for $s(n)+m+2$ steps by using the following rules, implementing a counter in the second subscript:

$$[q'_i \to q'_{i,1}]^\alpha_{(m-1)_\sigma} \qquad \text{for } \sigma \in \Sigma', q \in Q, 0 \le i < s(n), \alpha \in \{-,+\} \tag{17}$$

$$[q'_{i,t} \to q'_{i,t+1}]^\alpha_{(m-1)_\sigma} \qquad \text{for } \sigma \in \Sigma', q \in Q, 0 \le i < s(n), \alpha \in \{-,+\},\ 1 \le t < s(n)+m+2 \tag{18}$$

$$[q'_{i,s(n)+m+2} \to q''_i]^\alpha_{(m-1)_\sigma} \qquad \text{for } \sigma \in \Sigma', q \in Q, 0 \le i < s(n), \alpha \in \{-,+\} \tag{19}$$

The auxiliary objects q''_i produced at the end of counting are then sent out, while resetting the charges of the membranes j_σ to neutral:

$$[q''_i]^\alpha_{j_\sigma} \to [\ ]^0_{j_\sigma}\ q''_i \qquad \text{for } \sigma \in \Sigma',\ q \in Q,\ 0 \le i < s(n),\ 0 \le j < m,\ \alpha \in \{-,+\} \tag{20}$$

This allows the input objects to move back to the innermost membranes by using the rules of type (3).

When the $|\Sigma'|$ copies of q''_i reach the query-membranes, only one of these (corresponding to the membrane substructure that contained τ_i) is negatively charged, thus allowing that copy of q''_i to identify the symbol σ in cell i of M, and acquire it as its second subscript. The other copies of q''_i are sent out as the junk symbol # and deleted by rules of type (7):

$$[q''_i]^-_{?_\sigma} \to [\ ]^0_{?_\sigma}\ q_{i,\sigma} \qquad \text{for } \sigma \in \Sigma',\ q \in Q,\ 0 \le i < s(n) \tag{21}$$

$$[q''_i]^+_{?_\sigma} \to [\ ]^0_{?_\sigma}\ \# \qquad \text{for } \sigma \in \Sigma',\ q \in Q,\ 0 \le i < s(n) \tag{22}$$

[1] A better upper bound may be found by noticing that at most half the input objects (those having the correct bit in position j of their subscripts) can be sent out from a membrane j_σ; however, we keep the previous upper bound for simplicity's sake, as they are both $\Theta(s(n))$.

The state-object $q_{i,\sigma}$ is then sent out to h, where it waits for $s(n) + m$ steps by implementing a counter in the third subscript; $s(n) + m$ is an upper bound to the number of steps needed for all the input objects to reach the innermost membranes:

$$[q_{i,\sigma}]^+_\sigma \to [\ ]^+_\sigma\ q_{i,\sigma,1} \qquad \text{for } \sigma \in \Sigma',\ q \in Q,\ 0 \le i < s(n) \tag{23}$$

$$[q_{i,\sigma,t} \to q_{i,\sigma,t+1}]^0_h \qquad \text{for } \sigma \in \Sigma',\ q \in Q,\ 0 \le i < s(n),\ 1 \le t < s(n) + m \tag{24}$$

$$[q_{i,\sigma,s(n)+m} \to q'_{i,\sigma}]^0_h \qquad \text{for } \sigma \in \Sigma',\ q \in Q,\ 0 \le i < s(n) \tag{25}$$

The state-object $q'_{i,\sigma}$ produced at the end of counting now contains all the information needed to compute the transition function δ of M. Suppose $\delta(q, \sigma) = (r, \upsilon, d)$ for some $d \in \{-1, 0, +1\}$. Then $q'_{i,\sigma}$ sets the charge of membrane υ to negative (rules (26) and (27)) and waits for $m + 2$ steps by rules (29) and (30), thus allowing τ_i to move to $(m-1)_\upsilon$ by first entering υ (see rule (28)), the only negative membrane, and then by using the rules of type (2)–(3):

$$q'_{i,\sigma}\ [\ ]^+_\upsilon \to [q'_{i,\sigma}]^+_\upsilon \qquad \text{for } 0 \le i < s(n) \tag{26}$$

$$[q'_{i,\sigma}]^+_\upsilon \to [\ ]^-_\upsilon\ q''_{i,\sigma,1} \qquad \text{for } 0 \le i < s(n) \tag{27}$$

$$\tau_i\ [\ ]^-_\upsilon \to [\tau_i]^-_\upsilon \qquad \text{for } \tau \in \Sigma',\ 0 \le i < s(n) \tag{28}$$

$$[q''_{i,\sigma,t} \to q''_{i,\sigma,t+1}]^0_h \qquad \text{for } 0 \le i < s(n),\ 1 \le t < m + 2 \tag{29}$$

$$[q''_{i,\sigma,m+2} \to q'''_{i,\sigma}]^0_h \qquad \text{for } 0 \le i < s(n) \tag{30}$$

Finally, the state-object $q'''_{i,\sigma}$ produced by rule (30) resets the charge of membrane υ to positive, and is rewritten to reflect the change of state and head position, thus producing a configuration of Π_x corresponding to the new configuration of M, as described in Section 3:

$$q'''_{i,\sigma}\ [\ ]^-_\upsilon \to [q'''_{i,\sigma}]^-_\upsilon \qquad \text{for } 0 \le i < s(n) \tag{31}$$

$$[q'''_{i,\sigma}]^-_\upsilon \to [\ ]^+_\upsilon\ r_{i+d} \qquad \text{for } 0 \le i < s(n) \tag{32}$$

The P system Π_x is now ready to simulate the next step of M.

If $q \in Q$ was a final state of M, then recall that we have assumed that $\delta(q, \sigma)$ is undefined for all $\sigma \in \Sigma'$. We introduce the following rules, which halt the P system with the same result (acceptance or rejection) as M:

$$[q_i]^0_h \to [\ ]^0_h\ \mathit{yes} \qquad \text{for } 0 \le i < s(n), \text{ if } q \text{ is an accepting state} \tag{33}$$

$$[q_i]^0_h \to [\ ]^0_h\ \mathit{no} \qquad \text{for } 0 \le i < s(n), \text{ if } q \text{ is a rejecting state} \tag{34}$$

The simulation directly leads to the following result.

Theorem 4 *Let M be a single-tape deterministic Turing machine working in polynomial space $s(n)$ and time $t(n)$. Then, there exists an $(\mathbf{L}, \mathbf{L})$-uniform family $\mathbf{\Pi}$ of P systems with active membranes using object evolution and communication rules that simulates M in space $O(\log n)$ and time $O(t(n)s(n))$.*

Proof. For each $x \in \Sigma^n$, the P system Π_n can be built from 1^n in logarithmic space, since constructing the membrane structure and the set of rules only requires iterating over constant sets of symbols or states, or over sequences of integers of length $O(s(n))$, thus requiring only $O(\log n)$ bits. Analogously, the encoding $E(x)$ can also be computed in logarithmic space by simply subscripting the input symbols. Hence, the family $\mathbf{\Pi}$ described above is $(\mathbf{L}, \mathbf{L})$-uniform.

Each P system Π_x uses only a logarithmic number of membranes and a constant number of objects per configuration, besides the input objects, which are never rewritten; thus, Π_x works in space $O(\log n)$. Simulating one of the $t(n)$ steps of M requires $O(s(n))$ time, as this is an upper bound to the subscripts of objects used to introduce delays during the simulation. The total time is thus $O(t(n)s(n))$. □

Notice that the uniformity condition on $\mathbf{\Pi}$ is strictly weaker than **PSPACE**; hence, "cheating" by hiding the simulation into the construction of $\mathbf{\Pi}$ is impossible: the P systems themselves play a necessary role. The following equivalences follow:

Theorem 5 *For each class $\mathcal{D} \subseteq \mathcal{AM}$ of P systems with active membranes using object evolution and communication among their rules we have*

$$(\mathbf{L}, \mathbf{L})\text{-}\mathbf{LMCSPACE}_{\mathcal{D}} = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}} = \mathbf{PSPACE}.$$

Proof. The inclusion $\mathbf{PSPACE} \subseteq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{LMCSPACE}_{\mathcal{D}}$ follows immediately from Theorem 4. The class $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{LMCSPACE}_{\mathcal{D}}$ is included in $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}}$ by definition. Finally, the inclusion of $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}}$ in **PSPACE** can be proved by simulating P systems by means of Turing machines, which can be carried out with just a polynomial space overhead, as shown in [7, 1]. □

5. Conclusions

We proved that polynomial space Turing machines may be simulated by P systems with active membranes using logarithmic space and a polynomial number of read-only input objects. The equivalence of logarithmic and polynomial space for P systems, and their equivalence to polynomial space for Turing machines, follows.

This result shows that the previously known simulations of polynomial-space Turing machines by means of P systems [1] were far from optimal, and that any problem solvable by P systems in polynomial space is also solvable in logarithmic space. This also suggests that most of the computational power of polynomial-space P systems is due to communication rules moving the read-only input objects between regions. Indeed, the simulation we presented is based on two key ideas. First, input objects are never rewritten; after an initialization phase, used to build a representation of the initial configuration of the Turing machine M, for each input object τ_i we disregard the value of τ: the symbol σ written on the i-th tape cell of M can be inferred from the label of the substructure that contains τ_i. The second idea is applied when querying the symbol under the tape head: the position i of the head is written in binary in the electrical charges of membranes $0_\sigma, \ldots, (m-1)_\sigma$, so that the only object τ_i having the correct subscript can leave the substructure corresponding to the sought symbol,

and reach the skin membrane. The depth of each substructure is logarithmic, thus allowing to represent a polynomial number of possible head positions. As a result, we can simulate any polynomial space computation of a deterministic Turing machine with only a logarithmic number of symbols (plus a polynomial number of read-only input symbols) and membranes. Indeed, as can be inferred from the technical details of the simulation, only a constant number of auxiliary objects is needed.

This result highlights a large space gap in the hierarchy of space complexity classes for P systems: in order to solve harder problems than those solved in logarithmic space, one needs *super-polynomial* space. As was previously proved [1, Corollary 14], this phenomenon does *not* occur for super-polynomial space bounds, where a polynomial increase of space strictly increases the computing power of P systems.

Having characterised the class $\mathbf{LMCSPACE}_{\mathcal{AM}}$, we still need to investigate sublogarithmic space P systems, and in particular those working in constant space, in order to establish whether they are weaker than logarithmic-space ones. We expect this to be true, since a logarithmic number of nested membranes might be necessary to correctly identify the i-th input object, i.e., to preserve the ordering of symbols in the original input string.

References

[1] A. Alhazov, A. Leporati, G. Mauri, A. E. Porreca, C. Zandron, Space complexity equivalence of P systems with active membranes and Turing machines. *Theoretical Computer Science* **529** (2013), 69–81.

[2] A. Alhazov, C. Martín-Vide, L. Pan, Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae* **58** (2003) 2, 67–77.

[3] N. Murphy, D. Woods, The computational power of membrane systems under tight uniformity conditions. *Natural Computing* **10** (2011) 1, 613–632.

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

[5] Gh. Păun, G. Rozenberg, A. Salomaa (eds.), *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[6] A. E. Porreca, A. Leporati, G. Mauri, C. Zandron, Introducing a space complexity measure for P systems. *International Journal of Computers, Communications & Control* **4** (2009) 3, 301–310.

[7] A. E. Porreca, A. Leporati, G. Mauri, C. Zandron, P systems with active membranes working in polynomial space. *International Journal of Foundations of Computer Science* **22** (2011) 1, 65–73.

[8] A. E. Porreca, A. Leporati, G. Mauri, C. Zandron, Sublinear-space P systems with active membranes. In: E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, G. Vaszil (eds.), *Membrane Computing, 13th International Conference, CMC 2012*. Lecture Notes in Computer Science 7762, Springer, 2013, 342–357.