Constant-Space P Systems with Active Membranes

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

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

Summary. We continue the investigation of the computational power of space-constrained P systems. We show that only a constant amount of space is needed in order to simulate a polynomial-space bounded Turing machine. Due to this result, we propose an alternative definition of space complexity for P systems, where the amount of information contained in individual objects and membrane labels is also taken into account. Finally, we prove that, when less than a logarithmic number of membrane labels is available, moving the input objects around the membrane structure without rewriting them is not enough to even distinguish inputs of the same length.

1 Introduction

This paper continues the recent investigations of the computational power of P systems with active membranes by looking at the problems that they are able to solve while working in constant space. It is already known that a super-polynomial amount of space is needed to solve problems outside **PSPACE** [6]. Recently [2], it has been shown that logarithmic space suffices to simulate a polynomial space-bounded Turing machine (TM). Here we show that the constant space is sufficient and, trivially, necessary to solve all problems in **PSPACE**.

This result challenges our intuition about space. How can we even be able to remember, for example, the position of the TM head when we have less than a logarithmic number of bits of information? We discuss the implication of this result and how the current definition of space in P systems can be changed in order to better represent out intuition about "space". With the new definition all the known results involving polynomial (or larger) amount of space, according to the old definition, still hold. Only in the case of P systems with severely tight bounds on space the new definition makes a difference.

Finally, we show a result highlighting the importance of membranes for P systems. In fact, when only movement rules, i.e., send-in, send-out, and dissolution without rewriting, are allowed on the input symbols and less than a logarithmic number of membrane labels are present, there is no possibility of correctly determining even if two inputs are distinct, unless the ordering of the symbols is discarded.

The paper is organised as follows. The basic notions necessary for the rest of the paper are presented in Section 2. The main result, that is, the simulation of a polynomial space-bounded TM, is described in Section 3. The current definition of space in P systems is discussed and an alternative definition is proposed in Section 4. In Section 5 we examine the limitations arising when only movement but no rewriting is possible for the input symbols. Finally, in Section 6 we present a brief summary of the results and some possible future research directions.

2 Basic Notions

We recall the basic definitions related to P systems with active membranes with input alphabet [7].

Definition 1. *A* P system with (elementary) active membranes *of initial degree $d \geq 1$ is a tuple $\Pi = (\Gamma, \Delta, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$, where:*

- *Γ is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- *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 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 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 rule 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 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 $b = a$).

- *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.
- *Dissolution rules*, of the form $[a]_h^\alpha \to b$
  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the membrane h is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b. If $b \in \Delta$ then $a = b$ must hold.
- *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 in the initial multiset are copied to both membranes. If $b \in \Delta$ (resp., $c \in \Delta$) then $a = b$ and $c \notin \Delta$ (resp., $a = c$ and $b \notin \Delta$) must hold.

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 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, 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, dissolution or elementary division 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 to communication, dissolution, or elementary division 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 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 computation step is conventionally described as a sequence of micro-steps as follows.

First, all evolution rules are applied inside the elementary membranes, followed by all communication, dissolution and division rules involving the membranes themselves; this process is then repeated 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. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; this way, the objects that are duplicated during the division are already the final ones.
- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence of configurations $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, 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 of Π are applicable in $\mathcal{C}_k$. A *non-halting* computation $\mathcal{C} = (\mathcal{C}_i : i \in \mathbb{N})$ 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; exactly one of these must be sent out from the outermost membrane in the last step of each computation, in order to signal acceptance or rejection, respectively; we also assume that all computations are halting. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. If this is not necessarily the case, then we have a *non-confluent* P system, and the overall result is established as for nondeterministic Turing machines: it is acceptance iff an accepting computation exists. Unless otherwise specified, the P systems in this paper are to be considered confluent.

In order to solve decision problems (i.e., decide languages), 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 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 [3].

Definition 2. *Let $\mathcal{E}$, $\mathcal{F}$ be classes of functions over strings. A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* $(\mathcal{E}, \mathcal{F})$-uniform *if the mapping $x \mapsto \Pi_x$ can be described by two functions $F \in \mathcal{F}$ (for "family") and $E \in \mathcal{E}$ (for "encoding") as follows:*

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

In particular, a family $\boldsymbol{\Pi}$ is said to be $(\mathbf{L}, \mathbf{L})$-uniform if the functions E and F can be computed by a deterministic Turing machine in logarithmic space.

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 [5, 7].

Definition 3. *Let $\mathcal{C}$ be a configuration of a P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the sum of the number of membranes in the current membrane structure and the total number of objects from 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*

$$|\Pi| = \sup\{|\vec{\mathcal{C}}| : \vec{\mathcal{C}} \text{ is a computation of } \Pi\}.$$

Finally, let $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recogniser P systems, and let $s \colon \mathbb{N} \to \mathbb{N}$. We say that $\boldsymbol{\Pi}$ operates within space bound s *iff $|\Pi_x| \leq s(|x|)$ for each $x \in \Sigma^\star$.*

Definition 4. *We denote by $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(f(n))$ the class of languages decidable by $(\mathbf{L}, \mathbf{L})$-uniform families of* confluent *P systems with active membranes within space bound f. In particular, $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1))$ denotes the class of languages decidable in constant space.*

3 Simulating Polynomial-Space Turing Machines

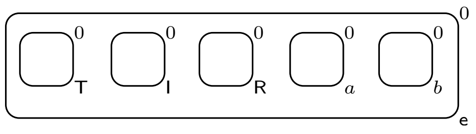
The idea behind the simulation of a polynomial-space bounded TM using only constant space in the P system is to use the input objects as a way to store the status of the TM tape. Since by rewriting more than a constant number of input symbols the amount of space would be non-constant, the only way to use them to store the state of the TM tape is to track the position of the symbols inside the membranes. We will use a tape alphabet consisting of two symbols, a and b; the construction presented here immediately generalises to alphabets of any size.

The simulation relies on two main ideas in order to store and retrieve the content of the simulated TM tape. The first idea is that, apart from a short initialisation procedure, the only relevant part of an input object σ_j is its subscript, that is, the position j in the TM tape. We will then have two membranes, named a and b, in which the input objects will be distributed. The interpretation of the fact that an object σ_j is in membrane a is that the j-th cell of the TM contains the symbol a. Notice that $\sigma = a$ is not required, since this information is not used after the initial phase of the simulation. The second idea is that it is possible to "read" a subscript of an input object σ_j without rewriting it and by using only a constant number of additional objects and membranes. An input object σ_j can use an evolution rule to generate a timer that, after j time steps, changes the charge of a membrane. Any other object that was observing that same membrane, i.e., it was counting together with the timer, is able to obtain the value j of the subscript of σ_j. The two ideas intuitively presented here, to be formalised in the construction below, allow us to store and retrieve the content of the tape of a TM using only a constant number of additional objects and membranes by "moving around" the input objects.

The simulation is divided into three phases. The first one is the initialisation, in which the input objects are distributed across the membranes of the systems. The second phase is simulation of a step of the TM, which requires the third phase, where the P system is reset in order to simulate the next step.

3.1 Membrane structure

The membrane structure consists of six membranes:



Each of these membranes has a specific semantics:

- T is a "temporary storage". It will contain objects that are not needed in that particular stage of the computation, and, then it will be emptied and the object contained will be moved to one of the other membranes. It also serves as the input membrane.
- I is used only during the initialisation, acting as a container for the "dispatching machinery" that moves an input object to the correct membrane.
- R is a membrane used to check the subscript of a specific input symbol. The input object σ_j generates a timer that changes the charge of the membrane after j time steps.
- Membrane a (resp. b) contains all input objects σ_j such that, in the currently simulated step of the TM, the j-th cell of the tape contains the symbol a (resp. b).
- e is the external membrane, containing all the others.

If the tape alphabet contains more than two symbols, it is possible to add more membranes inside e, one for each symbol.

3.2 Initialisation

The initial configuration of the P system contains all the input objects in membrane T and one auxiliary object move in membrane e. The initialisation procedure moves each input object a_j (resp., b_j) to membrane a (resp., b) and generates an object $q_{0,a}$, where q is the initial state of the TM, 0 indicates the position of the TM on the tape, and a indicates the fact that the P system simulating the TM will check if the symbol on the TM tape at position 0 is a. An example of the movement of one object to the correct membrane is shown in Fig. 1.

In the description of the simulation we assume that n is the size of the input and $p(n)$ is the length of the tape of the TM, which is polynomial in n.

The following set of rules uses the object move to transport an input object outside of membrane T or, if the membrane is empty, to start the simulation of the first step of the TM.

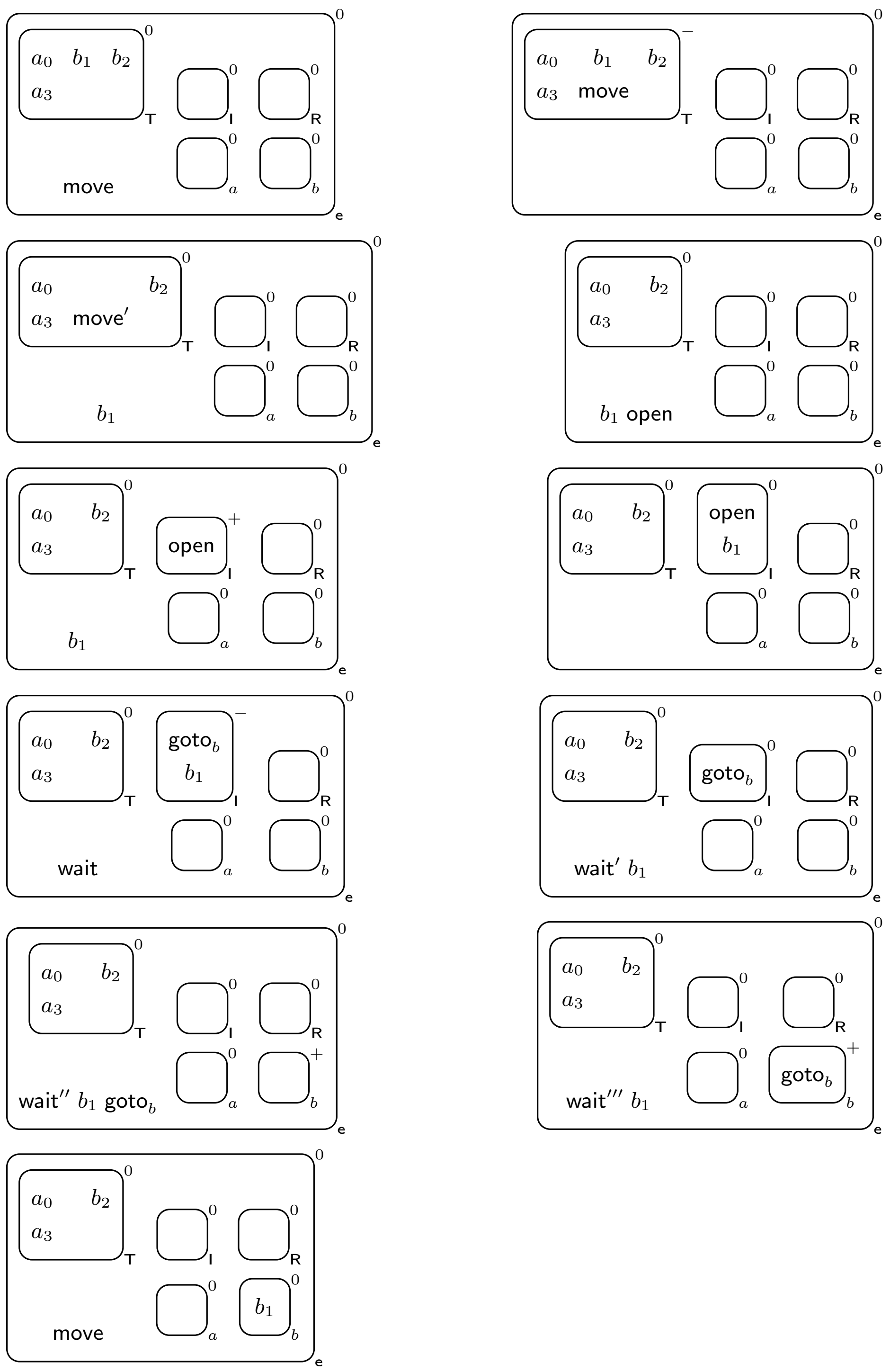
$$\begin{array}{ll}
\mathsf{move}\ [\ ]^0_{\mathsf{T}} \to [\mathsf{move}]^-_{\mathsf{T}} & \\
[\mathsf{move} \to \mathsf{move}']^-_{\mathsf{T}} & \\
[\mathsf{move}']^0_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ \mathsf{open} & \\
[a_i]^-_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ a_i & 0 \le i < p(n) \\
[b_i]^-_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ b_i & 0 \le i < p(n) \\
[\mathsf{move}']^-_{\mathsf{T}} \to [\ ]^0_{\mathsf{T}}\ q_{0,a} &
\end{array}$$

The next set of rules is used to move an input object from membrane e to I:

$$\begin{array}{ll}
\mathsf{open}\ [\ ]^0_{\mathsf{I}} \to [\mathsf{open}]^+_{\mathsf{I}} & \\
[\mathsf{open}]^0_{\mathsf{I}} \to [\ ]^-_{\mathsf{I}}\ \mathsf{wait} & \\
a_i\ [\ ]^+_{\mathsf{I}} \to [a_i]^0_{\mathsf{I}} & 0 \le i < p(n) \\
b_i\ [\ ]^+_{\mathsf{I}} \to [b_i]^0_{\mathsf{I}} & 0 \le i < p(n)
\end{array}$$

The following rules move an input object a_i (resp., b_i) to membrane a (resp., b):

$$\begin{array}{lll}
[a_i \to a_i\ \mathsf{goto}_a]^0_{\mathsf{I}} & 0 \le i < p(n) & (1) \\
[b_i \to a_i\ \mathsf{goto}_b]^0_{\mathsf{I}} & 0 \le i < p(n) & (2) \\
[\mathsf{goto}_a]^0_{\mathsf{I}} \to [\ ]^-_{\mathsf{I}}\ \mathsf{goto}_a & & \\
[\mathsf{goto}_b]^0_{\mathsf{I}} \to [\ ]^-_{\mathsf{I}}\ \mathsf{goto}_b & & \\
[a_i]^-_{\mathsf{I}} \to [\ ]^0_{\mathsf{I}}\ a_i & 0 \le i < p(n) & \\
[b_i]^-_{\mathsf{I}} \to [\ ]^0_{\mathsf{I}}\ b_i & 0 \le i < p(n) & \\
\mathsf{goto}_a\ [\ ]^0_a \to [\mathsf{goto}_a]^+_a & & \\
\mathsf{goto}_b\ [\ ]^0_b \to [\mathsf{goto}_b]^+_b & & \\
[\mathsf{goto}_a \to \epsilon]^+_a & &
\end{array}$$



a_0 b_1 b_2 a_3 move
T I R a b e
a_0 b_1 b_2 a_3 move
a_0 b_2 a_3 move′ b_1
a_0 b_2 a_3 b_1 open
a_0 b_2 a_3 open b_1
a_0 b_2 a_3 open b_1
a_0 b_2 a_3 goto$_b$ b_1 wait
a_0 b_2 a_3 goto$_b$ wait′ b_1
a_0 b_2 a_3 wait″ b_1 goto$_b$
a_0 b_2 a_3 goto$_b$ wait‴ b_1
a_0 b_2 a_3 b_1 move

Fig. 1. The movement of an input symbol to the correct membrane during the initialisation phase.

$$[\mathsf{goto}_b \to \epsilon]_b^+$$
$$a_i\ [\ ]_a^+ \to [a_i]_a^0 \qquad 0 \le i < p(n)$$
$$b_i\ [\ ]_b^+ \to [b_i]_b^0 \qquad 0 \le i < p(n) \tag{3}$$

Finally, the auxiliary object in membrane e has to wait for three steps before moving another input object outside of membrane T:

$$[\mathsf{wait} \to \mathsf{wait}']_{\mathsf{e}}^0$$
$$[\mathsf{wait}' \to \mathsf{wait}'']_{\mathsf{e}}^0$$
$$[\mathsf{wait}'' \to \mathsf{wait}''']_{\mathsf{e}}^0$$
$$[\mathsf{wait}''' \to \mathsf{move}]_{\mathsf{e}}^0$$

After the initialisation phase, all the input objects of the form a_i (resp., b_i) are located in membrane a (resp., b), and the actual simulation of the TM can start.

3.3 Simulation of a Step of the Turing Machine

To simulate a step of the TM we first define how its configuration is encoded as a configuration of the P system. Let $c_0, c_1, \ldots, c_{p(n)-1}$, with $c_i \in \{a, b\}$, be the tape of the TM at the current time step, and $\sigma_0, \ldots, \sigma_{n-1}$ the initial content of the tape. Then the P system contains, in membrane a, all σ_j such that $c_j = a$ and, in membrane b, all the input objects σ_j such that $c_j = b$. Notice that this allows a complete reconstruction of the tape of the TM. The state q of the TM and the position i of the head are encoded in an object $q_{i,\tau}$ in membrane e, where $\tau \in \{a, b\}$ indicates that the P system will check if the symbol in position i of the tape is τ.

The simulation proceeds as follows:

- The object $q_{i,\tau}$ moves an object σ_j from membrane τ, i.e., either a or b, to membrane R.
- In membrane R the object σ_j produces a timer that counts from j to 0, while $q_{i,\tau}$ counts from i to 0. When the first timer stops, it changes the charge of R, that is immediately changed again by the object σ_j exiting from R. This makes it possible to determine if $i = j$. In that case, the symbol on the tape of the TM in position i is actually τ, and it is possible to perform a step of the simulated machine. In the other case, i.e., $i \neq j$, the object σ_j is moved into membrane T and we return to the previous step.
- When membrane τ is empty, or the correct input object was found in the previous step, it is necessary to move back the objects from T to membrane τ. The search for the input object having subscript i will then proceed in membrane b (when $\tau = a$) or a (when $\tau = b$).

We will now detail the rules necessary to formally define the previous algorithm. In order to shorten the notation, in the following description we are going to write only half of the rules, those involving a. The missing half is obtained by swapping a and b.

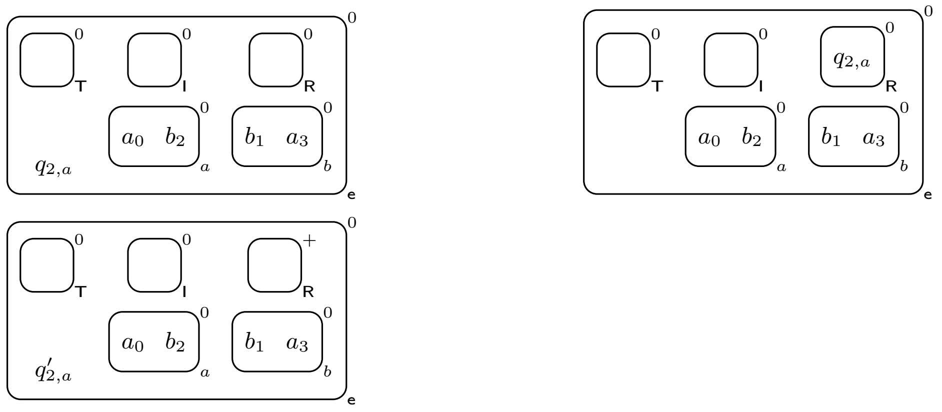


Fig. 2. The symbol $q_{2,a}$ changes the charge of the membrane R before starting the next phase of the simulation (continues in Fig 3).

Reading the Symbol under the Tape Head

This first set of rules uses the object $q_{i,a}$ to change the charge of membrane R to $+$, as shown in Fig. 2. The set Q' denotes the non-finals states of Q.

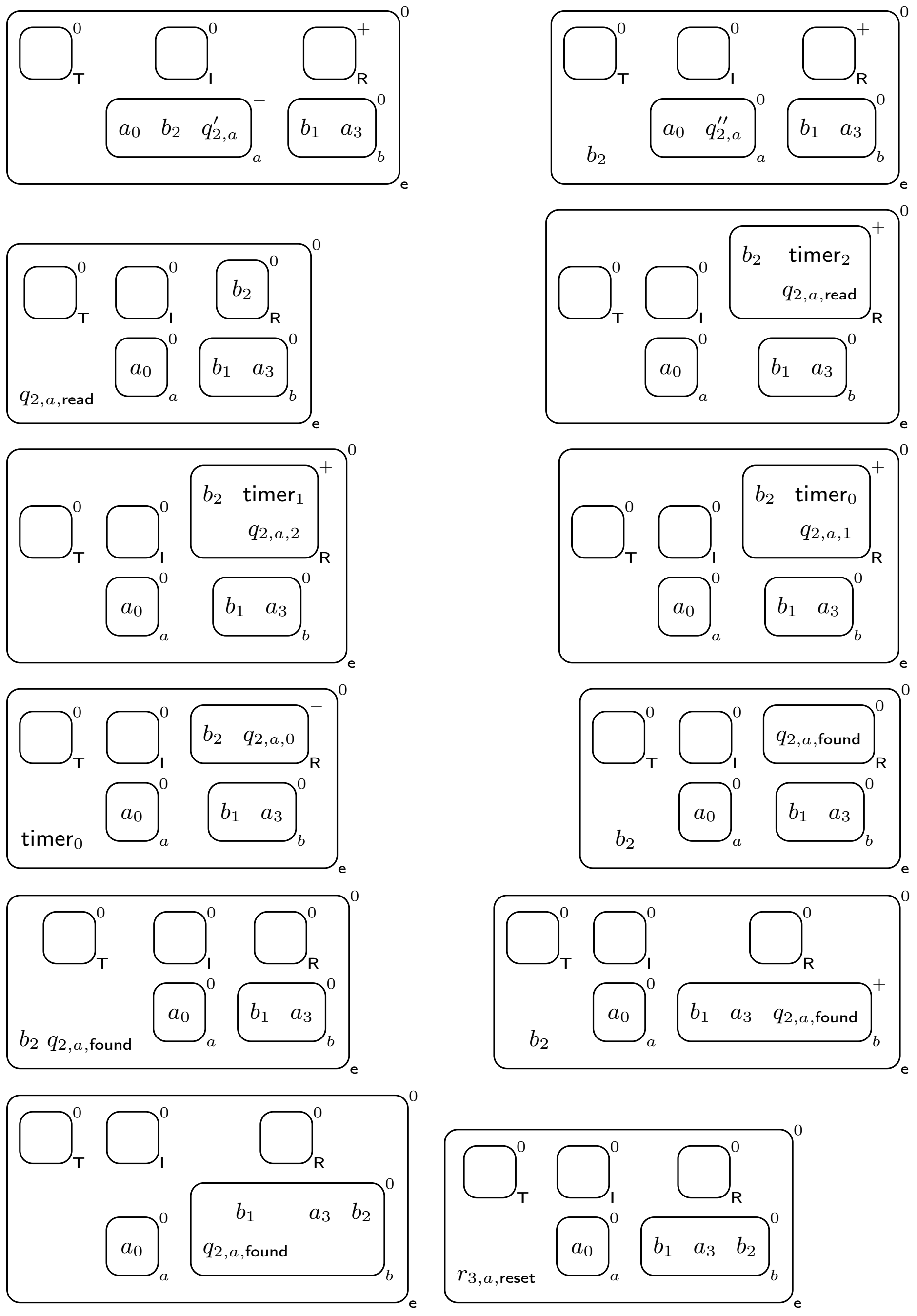
$$\begin{aligned}
&q_{i,a}\ [\ ]_{\mathsf{R}}^{0} \to [q_{i,a}]_{\mathsf{R}}^{0} && \text{for } q \in Q',\ 0 \le i < p(n)\\
&[q_{i,a}]_{\mathsf{R}}^{0} \to [\ ]_{\mathsf{R}}^{+}\ q'_{i,a} && \text{for } q \in Q',\ 0 \le i < p(n)
\end{aligned}$$

The following rules move an input object from membrane a to membrane R.

$$\begin{aligned}
&q'_{i,a}\ [\ ]_{a}^{0} \to [q'_{i,a}]_{a}^{-} && \text{for } q \in Q',\ 0 \le i < p(n)\\
&[q'_{i,a} \to q''_{i,a}]_{a}^{-} && \text{for } q \in Q',\ 0 \le i < p(n)\\
&[\sigma_j]_{a}^{-} \to [\ ]_{a}^{0}\ \sigma_j && \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n)\\
&[q''_{i,a}]_{a}^{0} \to [\ ]_{a}^{0}\ q_{i,a,\mathsf{read}} && \text{for } q \in Q',\ 0 \le i < p(n)\\
&[q''_{i,a}]_{a}^{-} \to [\ ]_{a}^{0}\ q_{i,a,\mathsf{reset}} && \text{for } q \in Q',\ 0 \le i < p(n) \qquad (4)\\
&\sigma_j\ [\ ]_{\mathsf{R}}^{+} \to [\sigma_j]_{\mathsf{R}}^{0} && \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n)
\end{aligned}$$

The next rules allow us to compare the position of the head of the TM (stored as a subscript of the object $q_{i,a,\mathsf{read}}$) with the subscript j of the object σ_j that has left membrane a. To accomplish this, the object σ_j in membrane R produces the object timer_j. At the same time step in which timer_j is produced, the object $q_{i,a,\mathsf{read}}$ enters R, changing its charge to $+$. Both timer_j and $q_{i,a,\mathsf{read}}$ rewrite themselves in order to count from j (resp., i) to 0. In this way, it is possible to determine if $i = j$. If it is so, then $q_{i,a,\mathsf{found}}$ exits from R. If not, it is $q_{i,a,\mathsf{not\text{-}found}}$ that appears. An example of application of those rules is presented in Fig. 3.

$$[\sigma_j \to \sigma_j\ \mathsf{timer}_j]_{\mathsf{R}}^{0} \qquad \text{for } \sigma \in \{a,b\},\ 0 \le j < p(n) \qquad (5)$$



a_0 b_2 $q'_{2,a}$
b_1 a_3
a_0 $q''_{2,a}$
b_2
b_2
$q_{2,a,\mathsf{read}}$
b_2 timer_2 $q_{2,a,\mathsf{read}}$
b_2 timer_1 $q_{2,a,2}$
b_2 timer_0 $q_{2,a,1}$
b_2 $q_{2,a,0}$
timer_0
$q_{2,a,\mathsf{found}}$
b_2 $q_{2,a,\mathsf{found}}$
b_1 a_3 $q_{2,a,\mathsf{found}}$
b_1 a_3 b_2 $q_{2,a,\mathsf{found}}$
$r_{3,a,\mathsf{reset}}$
b_1 a_3 b_2
T
I
R
e
a
b

Fig. 3. The symbol $q_{2,a}$ is used to discover that position 2 of the tape contains a. After that, a transition to state r and a movement of the tape head to position 3 is performed.

$$\begin{array}{ll}
q_{i,a,\mathsf{read}}\ [\ ]_{\mathsf{R}}^{0} \rightarrow [q_{i,a,\mathsf{read}}]_{\mathsf{R}}^{+} & \text{for } q \in Q',\ 0 \leq i < p(n) \\
[\mathsf{timer}_j \rightarrow \mathsf{timer}_{j-1}]_{\mathsf{R}}^{+} & \text{for } 1 \leq j \leq p(n) \\
[q_{i,a,\mathsf{read}} \rightarrow q_{i,a,i}]_{\mathsf{R}}^{+} & \text{for } q \in Q',\ 0 \leq i < p(n) \\
[q_{i,a,k} \rightarrow q_{i,a,k-1}]_{\mathsf{R}}^{+} & \text{for } q \in Q',\ 0 \leq i < p(n),\ 1 \leq k \leq p(n) \\
[\mathsf{timer}_0]_{\mathsf{R}}^{+} \rightarrow [\ ]_{\mathsf{R}}^{-}\ \mathsf{timer}_0 & \\
[\mathsf{timer}_0 \rightarrow \epsilon]_{\mathsf{e}}^{0} & \\
[\sigma_j]_{\mathsf{R}}^{-} \rightarrow [\ ]_{\mathsf{R}}^{0}\ \sigma_j & \text{for } \sigma \in \{a,b\},\ 0 \leq j < p(n) \\
[q_{i,a,0} \rightarrow q_{i,a,\mathsf{found}}]_{\mathsf{R}}^{-} & \text{for } q \in Q',\ 0 \leq i < p(n) \\
[q_{i,a,0} \rightarrow q_{i,a,\mathsf{not\text{-}found}}]_{\mathsf{R}}^{+} & \text{for } q \in Q',\ 0 \leq i < p(n) \\
[q_{i,a,0} \rightarrow q_{i,a,\mathsf{not\text{-}found}}]_{\mathsf{R}}^{0} & \text{for } q \in Q',\ 0 \leq i < p(n) \\
[q_{i,a,\mathsf{found}}]_{\mathsf{R}}^{0} \rightarrow [\ ]_{\mathsf{R}}^{0}\ q_{i,a,\mathsf{found}} & \text{for } q \in Q',\ 0 \leq i < p(n) \\
[q_{i,a,\mathsf{not\text{-}found}}]_{\mathsf{R}}^{0} \rightarrow [\ ]_{\mathsf{R}}^{0}\ q_{i,a,\mathsf{not\text{-}found}} & \text{for } q \in Q',\ 0 \leq i < p(n)
\end{array} \tag{6}$$

If $i \neq j$ we need to move the selected input object σ_j to membrane T:

$$\begin{array}{ll}
q_{i,a,\mathsf{not\text{-}found}}\ [\ ]_{\mathsf{T}}^{0} \rightarrow [q_{i,a,\mathsf{not\text{-}found}}]_{\mathsf{T}}^{+} & \text{for } q \in Q',\ 0 \leq i < p(n) \\
\sigma_j\ [\ ]_{\mathsf{T}}^{+} \rightarrow [\sigma_j]_{\mathsf{T}}^{0} & \text{for } \sigma \in \{a,b\},\ 0 \leq j < p(n) \\
[q_{i,a,\mathsf{not\text{-}found}}]_{\mathsf{T}}^{0} \rightarrow [\ ]_{\mathsf{T}}^{0}\ q_{i,a} & \text{for } q \in Q',\ 0 \leq i < p(n)
\end{array}$$

On the other hand, if $i = j$ we have correctly identified the symbol under the head of the TM. Assume $\delta(q,a) = (r,\tau,d)$ with r non final; then we define the following rules:

$$\begin{array}{ll}
q_{i,a,\mathsf{found}}\ [\ ]_{\tau}^{0} \rightarrow [q_{i,a,\mathsf{found}}]_{\tau}^{+} & \text{for } 0 \leq i < p(n) \\
\sigma_i\ [\ ]_{\tau}^{+} \rightarrow [\sigma_i]_{\tau}^{0} & \text{for } \sigma \in \{a,b\},\ 0 \leq i < p(n) \\
[q_{i,a,\mathsf{found}}]_{\tau}^{0} \rightarrow [\ ]_{\tau}^{0}\ r_{i+d,a,\mathsf{reset}} & \text{for } 0 \leq i < p(n)
\end{array} \tag{7}$$

Notice that, in the case of a nondeterministic TM, the simple repetition of rule (7) for each $(r,\tau,d) \in \delta(q,a)$ assures a non-deterministic simulation on a non-confluent P system.

Notice that the last rule has a change in both the state and in the position of the head. The presence of reset in the subscript indicates that this object will move all the objects from membrane T to membrane a. This happens also in another case (see rule (4)) but without any change of state and position of the TM head. This is an intended behaviour, since in both cases, before continuing the simulation, we need to restore the configuration of the P system by emptying membrane T. After that, the simulation with proceed with the object $r_{i+d,b}$ in this case, and $q_{i,b}$ in the other, as intended.

Clean-up

The following set of rules use the object $q_{i,a,\mathsf{reset}}$ to move all the objects from membrane T to membrane a. After that, the object is rewritten into $q_{i,b}$.

$$\begin{array}{ll}
q_{i,a,\mathsf{reset}}\ [\ ]_{\mathsf{T}}^{0} \to [q_{i,a,\mathsf{reset}}]_{\mathsf{T}}^{-} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[\sigma_j]_{\mathsf{T}}^{-} \to [\ ]_{\mathsf{T}}^{0}\ \sigma_j & \text{for } \sigma \in \{a, b\},\ 0 \le j < p(n) \\
[q_{i,a,\mathsf{reset}} \to q'_{i,a,\mathsf{reset}}]_{\mathsf{T}}^{-} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[q'_{i,a,\mathsf{reset}}]_{\mathsf{T}}^{0} \to [\ ]_{\mathsf{T}}^{0}\ q'_{i,a,\mathsf{reset}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
q'_{i,a,\mathsf{reset}}\ [\ ]_{a}^{0} \to [q'_{i,a,\mathsf{reset}}]_{a}^{+} & \text{for } q \in Q',\ 0 \le i < p(n) \\
\sigma_j\ [\ ]_{a}^{+} \to [\sigma_j]_{a}^{0} & \text{for } \sigma \in \{a, b\},\ 0 \le j < p(n) \\
[q'_{i,a,\mathsf{reset}}]_{a}^{0} \to [\ ]_{a}^{0}\ q_{i,a,\mathsf{reset}} & \text{for } q \in Q',\ 0 \le i < p(n) \\
[q'_{i,a,\mathsf{reset}}]_{\mathsf{T}}^{-} \to [\ ]_{\mathsf{T}}^{0}\ q_{i,b} & \text{for } q \in Q',\ 0 \le i < p(n)
\end{array}$$

Now all the input objects have been moved to either membrane a or membrane b, and the P system proceeds by checking whether the tape head is reading the symbol b in state q.

3.4 Halting

If $\delta(q, a) = (r, \tau, d)$ and r is an accepting (resp., rejecting) state of the TM, then, when simulating the last transition, instead of rule (7) one of the following rules is applied:

$$\begin{array}{ll}
[q_{i,a,\mathsf{found}}]_{\tau}^{0} \to [\ ]_{\tau}^{0}\ \mathsf{yes} & \text{for } 0 \le i < p(n), \text{ if } r \text{ is accepting} \\
[q_{i,a,\mathsf{found}}]_{\tau}^{0} \to [\ ]_{\tau}^{0}\ \mathsf{no} & \text{for } 0 \le i < p(n), \text{ if } r \text{ is rejecting}
\end{array}$$

Finally, the object yes or no is sent out from the outermost membrane, as the last computation step by, via the rules

$$[\mathsf{yes}]_{\mathsf{e}}^{0} \to [\ ]_{\mathsf{e}}^{0}\ \mathsf{yes}$$
$$[\mathsf{no}]_{\mathsf{e}}^{0} \to [\ ]_{\mathsf{e}}^{0}\ \mathsf{no}$$

No further rule can then be applied, since the only remaining objects are the input objects located in membranes T, a, and b, where they are not subject to any rule, as the membrane charges are neutral.

3.5 Main Result

The simulation of one step of the TM requires at most a polynomial number of steps of the P system, since each of the different "phases" (initialisation, checking if the input symbol σ_j has a subscript corresponding to the current position of the head of the TM, and moving the objects from membrane T to either a or b) requires at most polynomial time – actually $O(p(n))$ – and it is repeated at most once for each cell of the tape, that is, at most $p(n)$ times.

Even if we have presented a simulation of a TM having only two tape symbols, it is possible to extend the simulation to arbitrary alphabets by using one membrane for each symbol, similarly to membranes a and b, and by adding the corresponding rules. Therefore, in the following theorem we assume that a blank tape symbol $\sqcup$ is available.

Theorem 1. $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1)) = \mathbf{PSPACE}$.

Proof. Let $L \in \mathbf{PSPACE}$, and let M be a TM deciding L in space $p(n)$). We can construct a family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ such that $L(\boldsymbol{\Pi}) = L$ by letting $F(1^n) = \Pi_n$, where Π_n is the P system simulating M on inputs of length n, and $E(x_0 \cdots x_{n-1}) = x_{1,1} \cdots x_{n-1,n-1} \sqcup_n \cdots \sqcup_{p(n)-1}$, i.e., by padding the input string x with $p(n) - n$ blank symbols before indexing the result with the positions of the symbols. Both F and E can be computed in logarithmic space by Turing machines, since they only require adding subscripts having a logarithmic number of bits to rules or strings having a fixed structure, and the membrane structure is fixed for all Π_n. Since the simulation of M only requires a constant number of membranes and non-input objects, the inclusion $\mathbf{PSPACE} \subseteq (\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1))$ follows. The reverse inclusion was proved in [6]. $\square$

4 Rethinking the Definition of Space

The result of Theorem 1 shows that constant-space P systems with active membranes have the same computational power of Turing machines working in polynomial space. This raises some doubts about the definition of space complexity for P systems adopted until now [5]: does counting each non-input object and each membrane as unitary space really capture an intuitive notion of "space"?

From an information-theoretic perspective, we may observe that the constant number of non-input objects employed by the simulation of Section 3.5 actually encode $\Theta(\log n)$ bits of information, since they are taken from an alphabet Γ of polynomial size. It may be argued that this amount of information needs a proportional amount of physical storage space (e.g., as DNA molecules of comparable size). Similarly, the membranes themselves, being identified by a label, contain $\Theta(\log |\Lambda|)$ bits of information, which must have a physical counterpart.[1]

A more accurate estimate of the space required by a configuration of a P system might be given by the following alternative definition:

[1] We refer to the objects and membrane labels actually appearing during the course of the computation here; if part of the alphabet Γ or some labels from Λ never appear in a configuration, then the information content might be smaller.

Definition 5. *Let $\mathcal{C}$ be a configuration of a P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the number of membranes in the current membrane structure* multiplied by $\log |\Lambda|$, *plus the total number of objects from Γ (i.e, the non-input objects) they contain* multiplied by $\log |\Gamma|$.

Adopting this stricter definition does not significantly change space complexity results for polynomial or larger upper bounds, i.e., the complexity classes $\mathbf{PMCSPACE}_{\mathcal{AM}}$, $\mathbf{EXPMCSPACE}_{\mathcal{AM}}$, and larger ones [1] remain unchanged.

On the other hand, the simulation described in this paper would require logarithmic space according to Definition 5. Furthermore, the space bounds of the previous simulation of polynomial-space Turing machines by means of logarithmic-space P systems with active membranes [2] also increase to $\Theta(\log n \log \log n)$, since in that case each configuration of the P systems contained $\Theta(\log n)$ membranes with distinct labels and $O(1)$ non-input objects.

It remains to be established if space (as in Definition 5) can be freely exchanged between objects and labels, or if one of the two is strictly more powerful.

5 Computing without Evolving Input Objects

In this section we are going to show that, if input object are only moved around the membrane structure (without being themselves rewritten into other objects), evolution rules involving the input objects, such as (1), (2), and (5), are essential in order to perform a simulation of a TM. In fact, if only non-rewriting send-in, send-out, and dissolution rules are applied to input symbols, and the number of membrane labels is $o(\log n)$, it is impossible even to correctly distinguish two input strings of the same length. This happens independently of the space used by the P systems, as long as the multiset encoding of the input $x \in \Sigma^\star$ is "simple":

Definition 6. *Let A be an alphabet containing Σ, and let*

$$s \colon A^\star \to \{\sigma_i : \sigma \in A, i \in \mathbb{N}\}^\star$$

be the function defined by $s(x_0 \cdots x_{n-1}) = x_{0,0} \cdots x_{n-1,n-1}$, i.e., the function subscripting each symbol with its position in the string.

We say that an encoding E of $\Sigma^\star$ is "simple" if there exists a function $g \colon \mathbb{N} \to A^\star$ such that $E(x) = s(x \cdot g(|x|))$, i.e., $E(x)$ is the original input string x, concatenated with a string depending only on the length of x, and indexed with the positions of its symbols.

Notice that the encoding employed in Theorem 1 is indeed simple.

When the encoding is simple and the input alphabet is at least binary, P systems with the limitations described above accepting (resp., rejecting) a long enough string x also accept (resp., reject) another string obtained by swapping two symbols of x.

Theorem 2. *Let $\boldsymbol{\Pi}$ be a family of $(\mathcal{E}, \mathcal{F})$-uniform, possibly non-confluent P systems with active membranes, where $\mathcal{F}$ is unrestricted, and $\mathcal{E}$ is the class of simple encodings. Suppose that the only rules involving input symbols are send-in communication, send-out communication and membrane dissolution, that these rules never rewrite the input symbol, and that the family uses $o(\log n)$ membrane labels.*

Then, there exists $n_0 \in \mathbb{N}$ such that, for each string $x = x_0 \cdots x_{n-1} \in \Sigma^\star$ with $|x| \geq n_0$, there exist $i < j < n$ such that x can be written as $u \cdot x_i \cdot v \cdot x_j \cdot w$ and $x \in L(\boldsymbol{\Pi})$ if and only if $u \cdot x_j \cdot v \cdot x_i \cdot w \in L(\boldsymbol{\Pi})$.

Proof. Let $\boldsymbol{\Pi}$ be a family of P systems as defined in the statement of the theorem, let $F \in \mathcal{F}$ be its "family" function, and let

$$F(1^n) = \Pi_n = (\Gamma, \Delta, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R).$$

Assume that the objects in Δ have, in R, only rules of type send-in, send-out, and dissolution not rewriting them. We impose no restriction on rules involving objects in Γ. Let us consider the possible rules applicable to a fixed object $a \in \Delta$ and respecting the imposed restrictions:

- There are at most $3^2|\Lambda|$ send-in rules of the form $a\ [\ ]_h^\alpha \to [a]_h^\beta$, since it is possible to choose the label of the membrane and the two charges.
- Similarly, there are at most $3^2|\Lambda|$ send-out rules of the form $[a]_h^\alpha \to [\ ]_h^\beta\ a$.
- The number of dissolution rules of the form $[a]_h^\alpha \to a$ is at most $3|\Lambda|$, since, contrarily to the last two cases, there is no charge on the right-hand side of the rule.

Thus, there are $21|\Lambda|$ possible rules per input object and $2^{21|\Lambda|}$ possible sets of rules involving each input object. Since the encoding of the input string is simple, each input object has the form σ_i for some $\sigma \in A$; hence, for each position i in the input multiset there are $\left(2^{21|\Lambda|}\right)^{|A|} = 2^{21|\Lambda||A|}$ possible sets of rules.

A necessary condition to distinguish two input objects $a, b \in \Delta$ is that the set of rules involving b cannot be simply obtained by replacing a with b in the set of rules involving a (i.e., their sets of rules are not isomorphic); otherwise, replacing a with b in the input multiset would not change the result of the computation of Π. In particular, this holds for the first n input objects $x_{0,0}, \ldots, x_{n-1,n-1}$, obtained by indexing $x = x_0 \cdots x_{n-1} \in \Sigma^n$. In order to be able to distinguish these n input objects it is necessary that

$$2^{21|\Lambda||\Sigma|} \geq n$$

that is, that the sets of rules associated with the first n objects are pairwise non isomorphic. This means that

$$|\Lambda| \geq \frac{\log n}{21|\Sigma|}$$

But $|\Lambda|$ is $o(\log n)$; hence, the inequality does not hold for large enough n. Instead, there exists n_0 such that, for each $n \geq n_0$, there are two indistinguishable positions i and j with $0 \leq i < j < n$: for each $x = u \cdot x_i \cdot v \cdot x_j \cdot w \in \Sigma^n$, either x and $u \cdot x_j \cdot v \cdot x_i \cdot w$ are both accepted, or they are both rejected. □

6 Final Remarks

In this paper we have solved the problem of determining the computational power of P systems working in constant space, by showing that they can simulate polynomial-space bounded Turing machines. The simulation is also efficient, in the sense that it is only polynomially slower than the original machine.

The solution of this problem raised some interesting questions about the ability of the current definition of space to capture our intuitions about the size of P systems. We have challenged the existing definition by considering also the number of bits necessary to encode the auxiliary objects and the labels of the membranes. While the new definition does not change any result involving an amount of space polynomial or larger, it changes the current result and, according to the new definition, our simulation requires logarithmic space.

Finally, we have shown that rewriting input objects, while not exploited in other simulations [2], is essential when less than a logarithmic number of membrane labels is present. In fact, distinguishing two inputs is not possible when the input objects are only moved around in the membrane structure, even when no restriction on the space are present.

In the future we plan to investigate the relationship between the size of the set of objects Γ and the set of membrane labels Λ. It would be interesting to understand if we can easily exchange the way the information is distributed between the two sets according to the new definition of space (Definition 5).

References

1. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: Space complexity equivalence of P systems with active membranes and Turing machines. Theoretical Computer Science 529, 69–81 (2014)
2. Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: A gap in the space hierarchy of P systems with active membranes (2014), submitted
3. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
4. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
5. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Introducing a space complexity measure for P systems. International Journal of Computers, Communications & Control 4(3), 301–310 (2009)
6. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes working in polynomial space. International Journal of Foundations of Computer Science 22(1), 65–73 (2011)
7. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) Membrane Computing, 13th International Conference, CMC 2012, Lecture Notes in Computer Science, vol. 7762, pp. 342–357. Springer (2013)