Fundamenta Informaticae XX (2014) 1–18
DOI 10.3233/FI-2012-0000
IOS Press

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*

Abstract. We show that a constant amount of space is sufficient to simulate a polynomial-space bounded Turing machine by P systems with active membranes. We thus obtain a new characterisation of **PSPACE**, which raises interesting questions about the definition of space complexity for P systems. We then propose an alternative definition, where the size of the alphabet and the number of membrane labels of each P system are 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 by our research group on 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** [7]; on the other hand, it has recently been shown [2] that logarithmic space suffices to simulate a polynomial-space bounded deterministic Turing machine. Here we show that a constant amount of space is sufficient and, trivially, necessary to solve all problems in **PSPACE**.

This result challenges our intuition about the space complexity of P systems. How could we possibly store, for example, the position of a Turing machine head when we can only use less than a logarithmic number of bits of information? We discuss the implications of this result and how the current definition of space complexity in P systems can be changed in order to better represent our intuition about the notion of “space”. With the new definition all the known results involving at least a polynomial amount

of space, according to the old definition, still hold. Only in the case of P systems with severely tight bounds on the amount of space used during computations the new definition makes a difference.

Finally, we show a result highlighting the importance of rewriting rules for P systems with a sub-logarithmic number of membrane labels. In fact, when the input objects can only be moved without rewriting them, and less than a logarithmic number of membrane labels are present, there is not even the possibility to determine if two inputs are distinct, unless the ordering of the symbols is not taken into account.

The paper is organised as follows. The basic notions used in the rest of the paper are presented in Section 2. The main result, that is, the simulation of a Turing machine working in polynomial space, is described in Section 3. The current definition of space complexity for P systems is then discussed, and an alternative definition is proposed in Section 4. In Section 5 we examine the limitations arising when only moving input objects but not rewriting them is possible. Finally, in Section 6 we present a brief summary of the results and some possible directions for future research.

2. Basic Notions

For a comprehensive introduction to P systems we refer the reader to *The Oxford Handbook of Membrane Computing* [5]. Here we start by recalling the basic definitions related to P systems with active membranes with an input alphabet [8].

Definition 2.1. A *P system with (elementary) active membranes* having 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 in 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 $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, 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 of Π are applicable 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 language *recognisers* by employing two distinguished objects yes and no; exactly one of these must be sent out from the outermost membrane, and only 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 $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. 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, as discussed in detail in [3].

Definition 2.2. Let $\mathcal{E}, \mathcal{F}$ be classes of functions over strings. A family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ 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 $\mathbf{\Pi}$ 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 [6, 8].

Definition 2.3. Let $\mathcal{C}$ be a configuration of a recogniser 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 obtained by computing the space required by all computations of Π and taking the supremum:

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

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

Definition 2.4. The class of languages decidable by $(\mathbf{L}, \mathbf{L})$-uniform families of *confluent* recogniser P systems with active membranes within space bound f is denoted by $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(f(n))$. In particular, the class of languages decidable in constant space is denoted by $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1))$.

3. Simulating Polynomial-Space Turing Machines

Let M be a single-tape, deterministic Turing machine working in polynomial space $p(n)$. Let Q be the set of states of M, including the initial state s, and let $Q' \subsetneq Q$ be the set of non-final states. Let $\Sigma = \{a, b\}$ denote the tape alphabet, and $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, 1\}$ 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 constant space.

The idea behind the simulation is to use the input objects of Π_x as a way to store the contents of the tape of M. Since by rewriting more than a constant number of input symbols the amount of space would be non-constant, the only way to use such symbols to store the state of the tape is to track their position inside the membrane structure. For simplicity, we will use a tape alphabet consisting of just two symbols, a and b; however, the construction presented here immediately generalises to alphabets of any size.

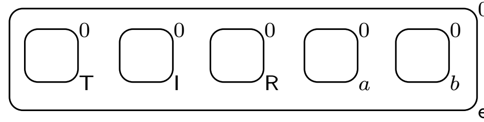
The simulation relies on two main ideas in order to store and retrieve the contents of the tape of M. The first idea is that, apart from during a short initialisation procedure, the only relevant part of an input object σ_j (with $\sigma \in \Sigma$ and $0 \leq j < p(n)$) is its subscript, that is, the position j on the 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 will be that the j-th cell of the tape contains the symbol a. Notice that $\sigma = a$ is not required, since this information is not used after the initialisation 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, as follows. The 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 counting together with the timer is able to observe the charge of the membrane, and thus 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 contents of the tape of M 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 Π_x. The second phase is the simulation of one computation step of M; this requires the third phase, where the configuration of Π_x is reset in order to simulate the next step of M.

3.1. Membrane structure

For any $x \in \Sigma^\star$, the membrane structure of Π_x (namely, $\Pi_n = F(1^n)$, where $n = |x|$), consists of six membranes:



Each of these membranes has a specific semantics:

- T is a “temporary storage”. It will contain objects which are not needed during a given phase of the computation. When that phase of computation terminates, the temporary storage is emptied and all the objects contained in it are moved to one of the other membranes. T also serves as the input membrane of Π_n.
- I is used only during the initialisation phase, acting as a container for the “dispatching machinery” that moves the input objects to the correct membranes.
- R is a membrane used to check the subscript of a specific input object σ_j: when required, such object generates a timer that changes the charge of this membrane after j time steps.
- Membrane a (resp. b) contains all input objects σ_j such that, in the currently simulated step of the Turing machine M, the j-th cell of the tape contains the symbol a (resp. b).
- e is the external (skin) membrane, containing all the other membranes.

If the tape alphabet of M contains more than two symbols, further membranes are added inside e, one for each symbol.

3.2. Initialisation

The initial configuration of Π_x contains all the input objects $\sigma_0, \sigma_1, \ldots, \sigma_{p(n)-1}$ representing x (padded to length $p(n)$, the length of the tape of M) 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 $s_{0,a}$, where s is the initial state of M, 0 indicates the position of the read/write head

on the tape, and a indicates the fact that Π_x will check whether the symbol on the tape of M at position 0 is a. The configuration of Π_x thus obtained corresponds to the initial configuration of M. An example of the movement of one object to the correct membrane during the initialisation phase is shown in Fig. 1.

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 M.

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

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

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

The following rules move an input object a_i (resp., b_i) to membrane a (resp., b); here ϵ denotes the empty multiset.

$$[a_i \to a_i\ \mathsf{goto}_a]_{\mathsf{I}}^{0} \qquad 0 \le i < p(n) \qquad (1)$$
$$[b_i \to b_i\ \mathsf{goto}_b]_{\mathsf{I}}^{0} \qquad 0 \le i < p(n) \qquad (2)$$
$$[\mathsf{goto}_a]_{\mathsf{I}}^{0} \to [\ ]_{\mathsf{I}}^{0}\ \mathsf{goto}_a$$
$$[\mathsf{goto}_b]_{\mathsf{I}}^{0} \to [\ ]_{\mathsf{I}}^{0}\ \mathsf{goto}_b$$
$$[a_i]_{\mathsf{I}}^{-} \to [\ ]_{\mathsf{I}}^{0}\ a_i \qquad 0 \le i < p(n)$$
$$[b_i]_{\mathsf{I}}^{-} \to [\ ]_{\mathsf{I}}^{0}\ b_i \qquad 0 \le i < p(n)$$
$$\mathsf{goto}_a\ [\ ]_{a}^{0} \to [\mathsf{goto}_a]_{a}^{+}$$
$$\mathsf{goto}_b\ [\ ]_{b}^{0} \to [\mathsf{goto}_b]_{b}^{+}$$
$$[\mathsf{goto}_a \to \epsilon]_{a}^{+}$$
$$[\mathsf{goto}_b \to \epsilon]_{b}^{+}$$
$$a_i\ [\ ]_{a}^{+} \to [a_i]_{a}^{0} \qquad 0 \le i < p(n)$$
$$b_i\ [\ ]_{b}^{+} \to [b_i]_{b}^{0} \qquad 0 \le i < p(n) \qquad (3)$$

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

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

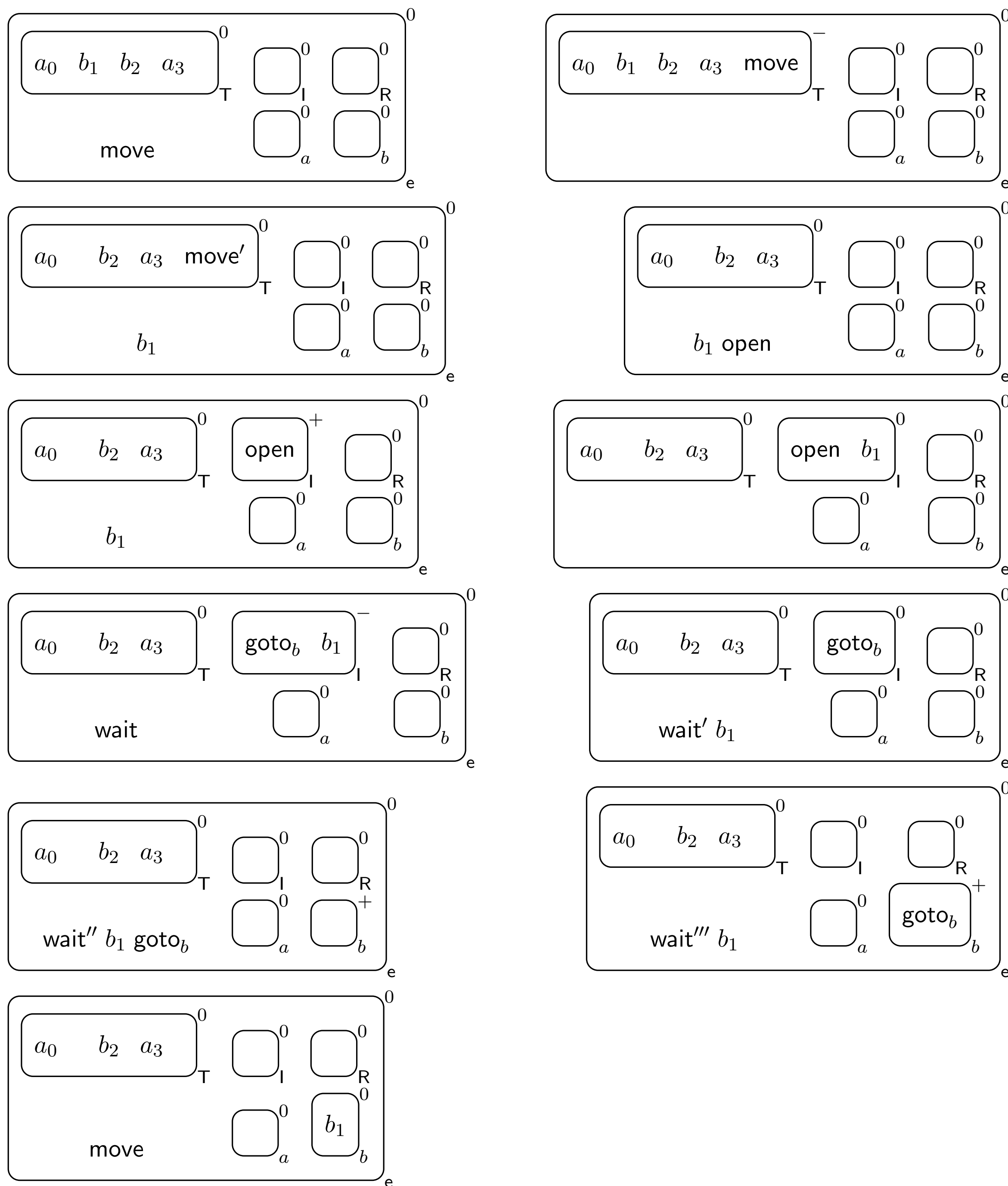


Figure 1. The movement of an input symbol to the correct membrane during the initialisation phase. Configurations are listed row by row, and each row is read from left to right.

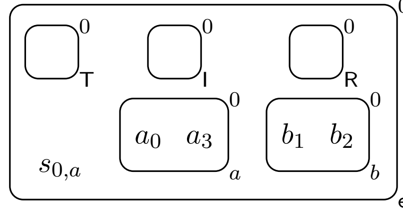


Figure 2. The initial configuration of M, assuming the contents of the tape are $abba$, as encoded by Π_x.

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

After the initialisation phase, all the input objects of the form a_i (resp., b_i) are located in membrane a (resp., b). The initial configuration of M is thus encoded in a configuration of Π_x, as shown in Fig. 2, and the simulation can start.

3.3. Simulation of a Step of the Turing Machine

To simulate a computation step of M we first define how its (generic) configuration is encoded as a configuration of Π_x. Let $\sigma_0, \ldots, \sigma_{p(n)-1}$, with $\sigma_j \in \{a, b\}$, be the initial contents of the tape of M. Fig. 2 shows the encoding of the initial configuration of M as a configuration of Π_x. Note that, in this configuration, all input objects σ_j such that $\sigma = a$ (resp., $\sigma = b$) are located inside membrane a (resp, b). As stated above, this correspondence is lost when representing a generic (non-initial) configuration of M. In fact, let $c_0, c_1, \ldots, c_{p(n)-1}$ be the contents of the tape of M at the current time step. Then the P system contains, in membrane a, all input objects σ_j such that $c_j = a$ and, in membrane b, all the objects σ_j such that $c_j = b$. Notice that this allows a complete reconstruction of the contents of the tape of M. The state q and the position i of the tape head are encoded in an object $q_{i,\tau}$ occurring 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 $q_{i,\tau}$ makes an object σ_j move 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 former timer stops, it changes the charge of R, which is then immediately changed again by the object σ_j while leaving from R. When this happens, the latter timer is able to determine if $i = j$. In that case, the symbol on the tape of M 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 the execution returns 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 $\tau = b$).

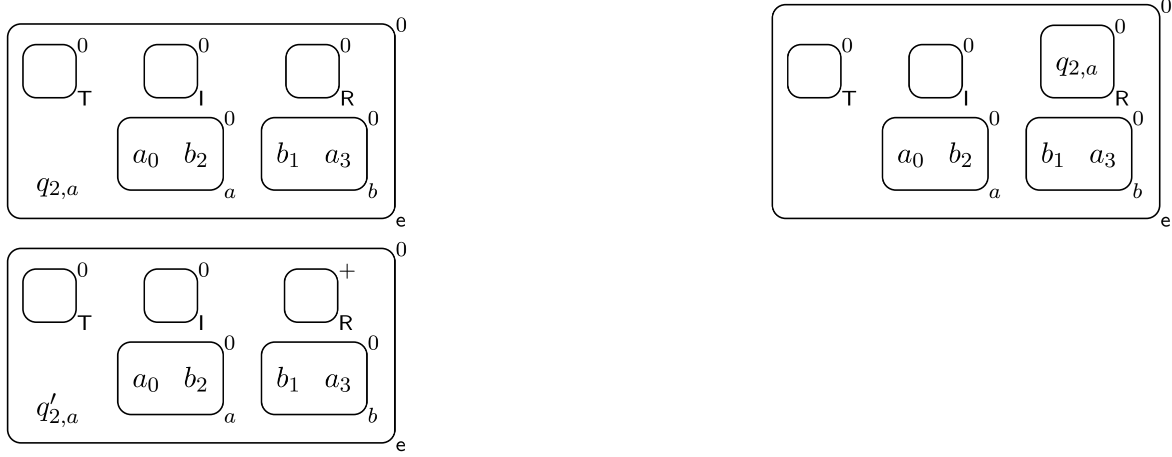


Figure 3. The symbol $q_{2,a}$ changes the charge of membrane R before starting the next phase of the simulation (continues in Fig 4).

We will now detail the rules necessary to formally define the previous algorithm. In order not to make the presentation unnecessarily long, 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 with b.

3.3.1. Reading the Symbol under the Tape Head

The first set of rules uses the object $q_{i,a}$ to change the charge of membrane R to $+$, as shown in Fig. 3. Recall that Q' denotes the set of non-final states of Q.

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

The following rules move an input object from membrane a to membrane R and produce the object $q_{i,a,\mathsf{read}}$, which indicates in its third subscript that the operation succeeded. On the other hand, if membrane a was empty then the object $q_{i,a,\mathsf{reset}}$ is produced, signalling that the clean-up operation must be performed. This operation, described later, moves all the objects contained in membrane T back to membrane a, and then switches the focus of the simulation from membrane a to membrane b, producing the object $q_{i,b}$.

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

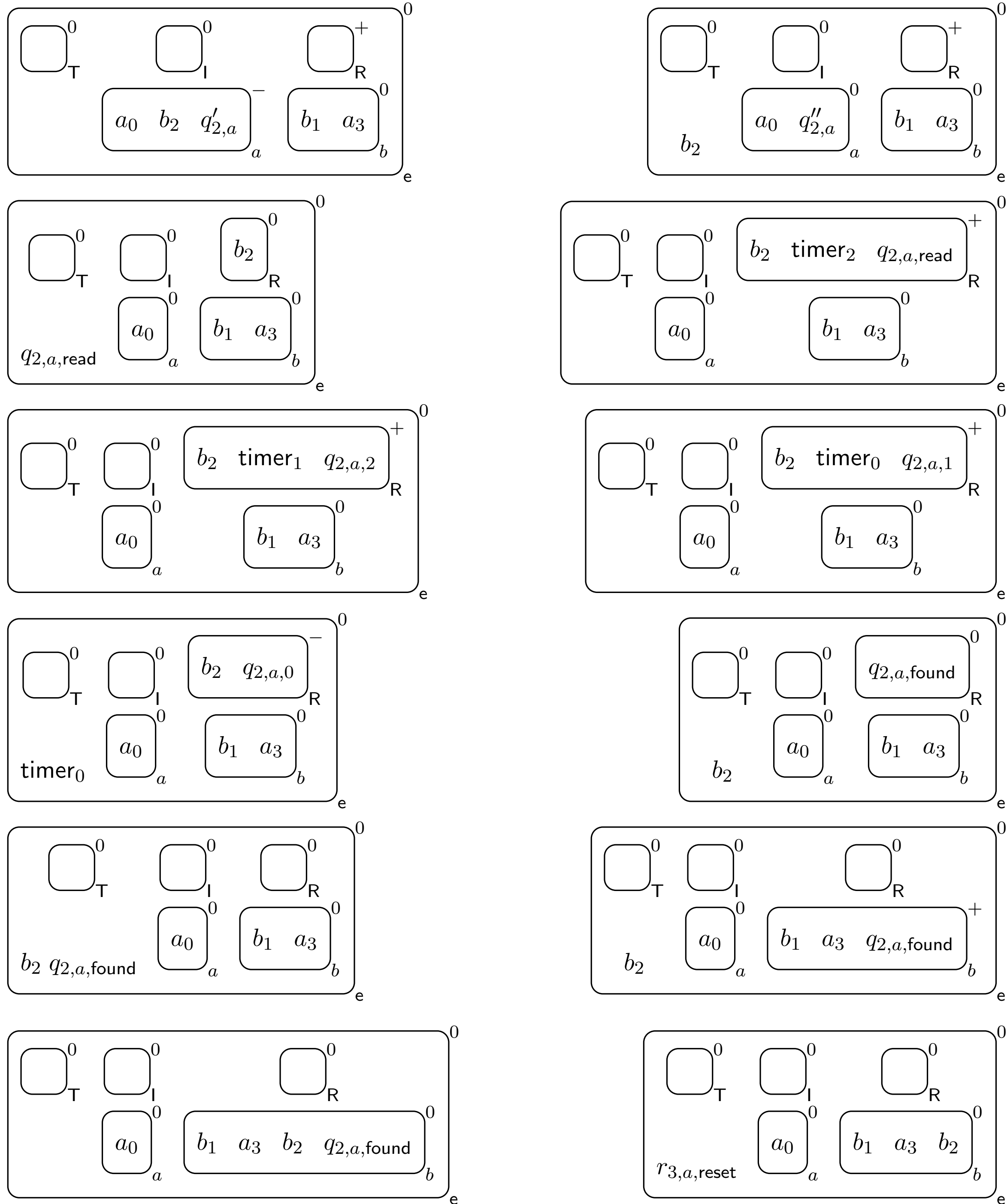


Figure 4. The symbol $q_{2,a}$ is used to discover that position 2 of the tape contains a. After that, a transition to state r and a movement of the tape head to position 3 is performed.

The next rules allow the P system to compare the position of the tape head of M (stored as a subscript of the object $q_{i,a,\mathsf{read}}$) with the subscript j of the object σ_j which 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 the next steps, in order to count from j (resp., i) to 0. If $i = j$, the object timer_j becomes timer_0, and at the very next step it leaves membrane R, setting its charge to negative. Simultaneously, object $q_{i,a,0}$ appears; by looking at the charge of R, it rewrites itself as $q_{i,a,\mathsf{found}}$, thus signalling that $i = j$. If $i < j$, the object $q_{i,a,0}$ appears while the charge of membrane R is still positive; it then rewrites itself into $q_{i,a,\mathsf{not\text{-}found}}$, thus signalling that $i \neq j$. Finally, if $i > j$, when the charge of R becomes negative (due to timer_0 having been sent out) an object $q_{i,a,k}$ with $k > 0$ still occurs; it immediately rewrites itself into $q_{i,a,\mathsf{not\text{-}found}}$ as before.

An example of application of these rules is presented in Fig. 4, for the case $i = j$.

$$
\begin{array}{lll}
[\sigma_j \to \sigma_j \ \mathsf{timer}_j]_{\mathsf{R}}^0 & \text{for } \sigma \in \{a,b\}, 0 \le j < p(n) & (5) \\
q_{i,a,\mathsf{read}} \ [\ ]_{\mathsf{R}}^0 \to [q_{i,a,\mathsf{read}}]_{\mathsf{R}}^+ & \text{for } q \in Q', 0 \le i < p(n) & \\
[\mathsf{timer}_j \to \mathsf{timer}_{j-1}]_{\mathsf{R}}^+ & \text{for } 1 \le j < p(n) & \\
[q_{i,a,\mathsf{read}} \to q_{i,a,i}]_{\mathsf{R}}^+ & \text{for } q \in Q', 0 \le i < p(n) & \\
[q_{i,a,k} \to q_{i,a,k-1}]_{\mathsf{R}}^+ & \text{for } q \in Q', 0 \le i < p(n), 1 \le k < p(n) & \\
[\mathsf{timer}_0]_{\mathsf{R}}^+ \to [\ ]_{\mathsf{R}}^- \ \mathsf{timer}_0 & & \\
[\mathsf{timer}_0 \to \epsilon]_{\mathsf{e}}^0 & & \\
[\sigma_j]_{\mathsf{R}}^- \to [\ ]_{\mathsf{R}}^0 \ \sigma_j & \text{for } \sigma \in \{a,b\}, 0 \le j < p(n) & \\
[q_{i,a,0} \to q_{i,a,\mathsf{found}}]_{\mathsf{R}}^- & \text{for } q \in Q', 0 \le i < p(n) & \\
[q_{i,a,0} \to q_{i,a,\mathsf{not\text{-}found}}]_{\mathsf{R}}^+ & \text{for } q \in Q', 0 \le i < p(n) & \\
[q_{i,a,k} \to q_{i,a,\mathsf{not\text{-}found}}]_{\mathsf{R}}^- & \text{for } q \in Q', 0 \le i < p(n), 1 \le k < p(n) & \\
[q_{i,a,\mathsf{found}}]_{\mathsf{R}}^0 \to [\ ]_{\mathsf{R}}^0 \ q_{i,a,\mathsf{found}} & \text{for } q \in Q', 0 \le i < p(n) & \\
[q_{i,a,\mathsf{not\text{-}found}}]_{\mathsf{R}}^0 \to [\ ]_{\mathsf{R}}^0 \ q_{i,a,\mathsf{not\text{-}found}} & \text{for } q \in Q', 0 \le i < p(n) & (6)
\end{array}
$$

If $i \neq j$, the selected input object σ_j has to be moved to membrane T, with the help of object $q_{i,a,\mathsf{not\text{-}found}}$. Then a new copy of object $q_{i,a}$ is produced, in order to repeat the process of candidate selection for the symbol which is under the tape head of M:

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

On the other hand, if $i = j$ then the symbol under the tape head of M has been correctly identified. Assume $\delta(q,a) = (r,\tau,d)$, with r non-final; the following rules simulate the computation of the next state r, the symbol τ to be written on the tape, and the movement d of the tape head:

$$
\begin{array}{ll}
q_{i,a,\mathsf{found}} \ [\ ]_{\tau}^0 \to [q_{i,a,\mathsf{found}}]_{\tau}^+ & \text{for } 0 \le i < p(n) \\
\sigma_i \ [\ ]_{\tau}^+ \to [\sigma_i]_{\tau}^0 & \text{for } \sigma \in \{a,b\}, 0 \le i < p(n)
\end{array}
$$

$$[q_{i,a,\mathsf{found}}]_\tau^0 \to [\,]_\tau^0\ r_{i+d,a,\mathsf{reset}} \qquad \text{for } 0 \le i < p(n) \tag{7}$$

Notice that, in the case M is a nondeterministic Turing machine, 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 the clean-up phase must be now executed, to move all objects from membrane T back to membrane a. As stated above, the clean-up phase is also executed after applying rule (4), this time however without any change of state and position of the tape 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 will proceed with the object $r_{i+d,b}$ in the former case, and with $q_{i,b}$ in the latter, as intended.

3.3.2. 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 $q_{i,b}$.

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

When all the input objects have been moved to either membrane a or membrane b, the P system can proceed by checking whether the tape head is reading the symbol b in position i when the machine M is 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:

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

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

$$\begin{aligned}
&[\mathsf{yes}]_\mathsf{e}^0 \to [\,]_\mathsf{e}^0\ \mathsf{yes}\\
&[\mathsf{no}]_\mathsf{e}^0 \to [\,]_\mathsf{e}^0\ \mathsf{no}
\end{aligned}$$

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 Turing machine 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 tape head, and moving the objects from membrane T to either membrane a or b) requires at most polynomial time – actually $O(p(n))$, where $p(n)$ is the space required by the Turing machine – 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 Turing machine 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, without loss of generality, that a blank tape symbol $\sqcup$ is also available.

Theorem 3.1. $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1)) = \mathbf{PSPACE}$.

Proof:
Let $L \in \mathbf{PSPACE}$, and let M be a Turing machine deciding L in space $p(n)$. We can construct a family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ such that $L(\mathbf{\Pi}) = L$ 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 on the tape. 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-bounded number of membranes (four plus the size of the alphabet of M) and non-input objects (at most two in each configuration), the inclusion of $\mathbf{PSPACE}$ in $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{AM}}(O(1))$ follows. The reverse inclusion was proved in [7]. $\square$

4. Rethinking the Definition of Space

The result of Theorem 3.1 shows that constant-space P systems with active membranes have the same computational power of Turing machines working in polynomial space. This raises some questions about the definition of space complexity for P systems adopted until now [6], in particular: does counting each non-input object and each membrane as unitary space really capture an intuitive notion of the amount of space used by a P system during a computation? Is it fair to allow a polynomial padding of the input string when encoding it as a multiset?

First of all, we observe that the constant number of non-input objects appearing in each configuration of the simulation described in Section 3.5 actually encode $\Theta(\log n)$ bits of information, since they are taken from an alphabet Γ of polynomial size. For instance, the non-input objects $q_{i,a}$ have a subscript i ranging from 0 to $p(n) - 1$; according to the original definition of space (Definition 2.3), each of these objects would only require unitary space, whereas the binary representation of the subscript i

requires $\log p(n) = \Theta(\log n)$ bits. It may be argued that this amount of information needs a proportional amount of physical storage space. Similarly, each membrane label contains $\Theta(\log |\Lambda|)$ bits of information, which must also have a physical counterpart.[1]

The information stored in the *positions* of the objects within the membrane structure is also not taken into account by Definition 2.3, although the number of configurations reachable during a computation is exponential even for just two regions (the configurations correspond to the subsets of n distinct input objects located in one of the two regions). However, the information on the location of the objects is part of the system and it is *not* stored elsewhere, exactly as the information on the location of the tape head in a Turing machine, which is not counted as space.

Due to the above considerations, we propose an alternative definition which we think might capture in a more accurate way this intuition of space.

Definition 4.1. 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 involving polynomial or larger upper bounds, i.e., the complexity classes $\mathbf{PMCSPACE}_{\mathcal{AM}}$, $\mathbf{EXPMCSPACE}_{\mathcal{AM}}$, and larger ones [1] remain unchanged.

As for padding the input string, one may argue that this operation provides the P system with some "free" storage, since input objects are not counted by Definition 2.3. The proof of Theorem 3.1 exploits the ability to encode an input string of length n as a polynomially larger multiset in a substantial way, as allowed by the most common uniformity conditions, including $\mathbf{P}$ and $\mathbf{L}$-uniformity, but also weaker ones such as $\mathbf{AC}^0$ or $\mathbf{DLOGTIME}$-uniformity [8, 3].

According to the above discussion, the simulation described in this paper would require logarithmic space according to Definition 4.1. Furthermore, the space bounds of the previous simulation of polynomial-space Turing machines by means of logarithmic-space P systems with active membranes described in [2] also increase to $\Theta(\log n \log \log n)$, since in that case each configuration of the P systems contains $\Theta(\log n)$ membranes with distinct labels and $O(1)$ non-input objects. Both simulations would be limited to *linear*-space Turing machines, rather than polynomial-space ones, if input padding were disallowed.[2]

It remains to be established if the amount of space (as measured in Definition 4.1) can be freely distributed between objects and labels while obtaining the same computing power, or if one of the two elements produces strictly more powerful P systems than the other.

5. Computing without Rewriting Input Objects

In this section we are going to show that, if input objects are only moved around the membrane structure (without rewriting them into other objects), then evolution rules involving the input objects, such as (1), (2), and (5) from Section 3, are essential in order to perform a simulation of a Turing machine using

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

[2]Notice that deciding whether a deterministic Turing machine accepts its input in linear space remains $\mathbf{PSPACE}$-complete, although this time the padding is performed during the reduction to this problem.

less than a logarithmic number of membrane labels. 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 $o(\log n)$, then it is even impossible 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 function E encoding the input $x \in \Sigma^\star$ as a multiset w_x is "simple":

Definition 5.1. 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.

An encoding E of $\Sigma^\star$ is "simple" if there exists a function $g\colon \mathbb{N} \to A^\star$ such that $E(x) = s(x \cdot g(|x|))$ for all $x \in \Sigma^\star$, 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 3.1 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 5.2. Let $\mathbf{\Pi}$ be a family of $(\mathcal{E}, \mathcal{F})$-uniform, possibly non-confluent recogniser P systems with active membranes, where $\mathcal{F}$ is unrestricted, and $\mathcal{E}$ is a class of simple encodings. Suppose that the only rules involving input symbols are send-in, send-out and membrane dissolution, that these rules never rewrite the input symbols, 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 $|x| = n \geq n_0$, there exist $i < j < n$ such that x can be written as $u \cdot x_i \cdot v \cdot x_j \cdot w$ with $u, v, w \in \Sigma^\star$, and $x \in L(\mathbf{\Pi})$ if and only if $u \cdot x_j \cdot v \cdot x_i \cdot w \in L(\mathbf{\Pi})$.

Proof:
Let $\mathbf{\Pi}$ 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 are subject, in R, only to rules of type send-in, send-out, and dissolution not rewriting them. On the other hand, 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 $h \in \Lambda$ and the charges $\alpha, \beta \in \{+, 0, -\}$.
- 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 hence $2^{21|\Lambda|}$ 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_x. 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 thus 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

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

However, since $|\Lambda|$ is $o(\log n)$, 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 recogniser 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 non-input objects and the labels of the membranes. While the new definition does not change any result involving an amount of space which is 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 restrictions on the amount of 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 4.1).

Acknowledgements

We would like to thank the anonymous referees for their comments, particularly those concerning the discussion on the definition of space complexity in Section 4.

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

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**, 2014, 69–81.

[2] Leporati, A., Mauri, G., Porreca, A. E., Zandron, C.: A gap in the space hierarchy of P systems with active membranes, *Journal of Automata, Languages and Combinatorics*, **19**(1–4), 2014, 173–184.

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

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

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

[6] 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), 2009, 301–310.

[7] 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), 2011, 65–73.

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