The computational power of exponential-space P systems with active membranes

Artiom Alhazov[1,2], Alberto Leporati[1], Giancarlo Mauri[1], Antonio E. Porreca[1], and Claudio Zandron[1]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
artiom.alhazov@unimib.it
{leporati,mauri,porreca,zandron}@disco.unimib.it
[2] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
artiom@math.md

Abstract. We show that exponential-space P systems with active membranes characterise the complexity class **EXPSPACE**. This result is proved by simulating Turing machines working in exponential space via uniform families of P systems with restricted elementary active membranes; the simulation is efficient, in the sense that the time and space required are at most polynomial with respect to the resources employed by the simulated Turing machine.

1 Introduction

P systems with active membranes have been introduced in [9] as a variant of P systems where the membranes have an active role during computations: they have an electrical charge that can inhibit or activate the rules that govern the evolution of the system, and they can grow in number by using division rules.

In several papers these systems were used to attack computationally hard problems, by exploiting the possibility to create, in polynomial time, an exponential number of membranes that evolve in parallel. Hence, for instance, it has been proved that P systems with active membranes can solve **PSPACE**-complete problems [11, 2] in polynomial time. When division rules operate only on *elementary* membranes (i.e. membranes not containing other membranes), such systems are still able to efficiently solve **NP**-complete problems [12, 5]. More recently, in [7] it was proved that all problems in $\mathbf{P}^{\mathbf{PP}}$ (a possibly larger class including the polynomial hierarchy) can also be solved in polynomial time using P systems with elementary membrane division. On the other hand, if division of membranes is not allowed then the efficiency apparently decreases [12]: no **NP**-complete problem can be solved in polynomial time without using division rules unless $\mathbf{P} = \mathbf{NP}$ holds.

A measure of space complexity for P systems has been introduced [6] in order to analyse the time-space trade-off exploited when P systems are used to efficiently solve computationally hard problems. The space required by a P system is the maximal size it can reach during any computation, defined as the sum of the number of membranes and the number of objects. A uniform family $\boldsymbol{\Pi}$ of recogniser P systems is said to solve a problem in space $f\colon \mathbb{N} \to \mathbb{N}$ if no P system in $\boldsymbol{\Pi}$ associated to an input string of length n requires more than $f(n)$ space. Under this notion of space complexity, in [8] it has been proved that the class of problems solvable in polynomial space by P systems with active membranes, denoted by $\mathbf{PMCSPACE}_{\mathcal{AM}}$, coincides with **PSPACE**. This result is proved by mutual simulation of P systems and Turing machines.

The techniques used up to now to simulate a polynomial-space Turing machine via a polynomial-space family of P systems [7] do not seem to apply when the space bound is less strict, i.e., exponential or even super-exponential. Indeed, we would need P systems with an exponential number of membranes with distinct labels, and such systems cannot be built in a polynomial number of steps by a deterministic Turing machine (as required by the notion of polynomial-time uniformity usually employed in the literature [5]).

Here we show that, by using different techniques, exponential-space Turing machines can be simulated by exponential-space P systems; hence, the classes of problems solvable by P systems with active membranes and by Turing machines in exponential space coincide; in symbols, $\mathbf{EXPMCSPACE}_{\mathcal{AM}} = \mathbf{EXPSPACE}$.

The rest of the paper is organized as follows. In section 2 we recall some definitions concerning P systems with active membranes and their space complexity. In section 3 we describe how P systems with restricted elementary membranes can be used to simulate Turing machines; an analysis on the resources (time and space) needed to perform this simulation is also given. Section 4 contains the statement of our characterization of **EXPSPACE**, while section 5 provides the conclusions as well as some directions for further research.

2 Definitions

We assume the reader to be familiar with the basic terminology and results concerning P systems with active membranes (see [10], chapters 11–12 for a survey). Here we just recall some definitions that are relevant for the results presented in this paper.

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

- Γ *is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- Λ *is a finite set of labels for the membranes;*
- μ *is a membrane structure (i.e., a rooted* unordered *tree, usually represented by nested brackets) consisting of* d *membranes enumerated by* $1, \ldots, d$*; furthermore, each membrane is labeled by an element of* Λ*, not necessarily in a one-to-one way;*

– $w_1, \ldots, w_d$ *are strings over* Γ*, describing the initial multisets of objects placed in the* d *regions of* μ*;*
– R *is a finite set of rules.*

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

The rules are of the following kinds:

– *Object evolution rules*, of the form $[a \to w]_h^\alpha$
They can be applied inside a membrane labeled 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 every object in w).
– *Send-in communication rules*, of the form $a\,[\,]_h^\alpha \to [b]_h^\beta$
They can be applied to a membrane labeled 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.
– *Send-out communication rules*, of the form $[a]_h^\alpha \to [\,]_h^\beta\, b$
They can be applied to a membrane labeled 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.
– *Dissolution rules*, of the form $[a]_h^\alpha \to b$
They can be applied to a membrane labeled 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.
– *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$
They can be applied to a membrane labeled 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 charge 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.
– *Nonelementary division rules*, of the form

$$\left[[\,]_{h_1}^+ \cdots [\,]_{h_k}^+ [\,]_{h_{k+1}}^- \cdots [\,]_{h_n}^- \right]_h^\alpha \to \left[[\,]_{h_1}^\delta \cdots [\,]_{h_k}^\delta \right]_h^\beta \left[[\,]_{h_{k+1}}^\epsilon \cdots [\,]_{h_n}^\epsilon \right]_h^\gamma$$

They can be applied to a membrane labeled by h, having charge α, containing the positively charged membranes $h_1, \ldots, h_k$, the negatively charged membranes $h_{k+1}, \ldots, h_n$, and possibly some neutral membranes. The membrane h is divided into two copies having charge β and γ, respectively; the positive children are placed inside the former membrane, their charge changed to δ, while the negative ones are placed inside the latter membrane, their charges changed to ϵ. Any neutral membrane inside h is duplicated and placed inside both copies.

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 any number of 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 reasoning applies to each membrane that can be involved to communication, dissolution, elementary or nonelementary 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.
- While all the chosen rules are considered to be applied simultaneously during each computation step, they are logically applied in a bottom-up fashion: first, all evolution rules are applied to the elementary membranes, then all communication, dissolution and division rules; then the application proceeds towards the root of the membrane structure. In other words, each membrane evolves only after its internal configuration has been updated.
- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

The precise variant of P systems we use in this paper does not use dissolution or nonelementary division rules.

Definition 2. *A* P system with restricted elementary active membranes *is a P system with active membranes where only object evolution, send-in, send-out, and elementary division rules are used. This kind of P systems is denoted by* $\mathcal{AM}(-\mathrm{d}, -\mathrm{n})$.

A *halting computation* of the P system Π is a finite sequence of configurations $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration, every $\mathcal{C}_{i+1}$ is reachable by $\mathcal{C}_i$ via a single computation step, and no rules can be applied anymore in $\mathcal{C}_k$. A *non-halting* computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps, where the applicable rules are never exhausted.

P systems can be used as *recognisers* by employing two distinguished objects YES and NO; exactly one of these must be sent out from the outermost membrane during each computation, in order to signal acceptance or rejection respectively; we also assume that all computations are halting. 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. All P systems in this paper are 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 3. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* (polynomial-time) uniform *if the mapping $x \mapsto \Pi_x$ can be computed by two deterministic polynomial-time Turing machines F (for "family") and E (for "encoding") as follows:*

- *The machine F, taking as input* the length n of x *in unary notation, constructs a P system Π_n, which is common for all inputs of length n, with a distinguished input membrane.*
- *The machine E, on input x, outputs a multiset w_x (an encoding of the specific input x).*
- *Finally, Π_x is simply Π_n with w_x added to the multiset placed inside its input membrane.*[3]

Definition 4. *If the mapping $x \mapsto \Pi_x$ is computed by a* single *polynomial-time Turing machine, the family $\boldsymbol{\Pi}$ is said to be* semi-uniform. *In this case, inputs of the same size may be associated with P systems having possibly different membrane structures and rules.*

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].

Definition 5. *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 they contain. If $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a halting computation of Π, then the* space required by $\boldsymbol{\mathcal{C}}$ *is defined as*

$$|\boldsymbol{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\}$$

[3] Notice that this definition of uniformity is (possibly) weaker than the other one commonly used in membrane computing [5], where the Turing machine F maps each input x to a P system $\Pi_{s(x)}$, where $s\colon \Sigma^\star \to \mathbb{N}$ is a measure of the size of the input; in our case, $s(x)$ is always $|x|$.

or, in the case of a non-halting computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$,

$$|\boldsymbol{\mathcal{C}}| = \sup\{|\mathcal{C}_i| : i \in \mathbb{N}\}.$$

Non-halting computations might require an infinite amount of space (in symbols $|\boldsymbol{\mathcal{C}}| = \infty$*): for example, if the number of objects strictly increases at each computation step.*

The space required by Π *itself is then*

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

Notice that $|\Pi| = \infty$ *might occur if either* Π *has a non-halting computation requiring infinite space (as described above), or* Π *has an infinite set of halting computations, such that for each bound* $b \in \mathbb{N}$ *there exists a computation requiring space larger than* b*.*

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

By $\mathbf{MCSPACE}_{\mathcal{D}}(f(n))$ we denote the class of languages which can be decided by uniform families of confluent P systems of type $\mathcal{D}$ where each $\Pi_x \in \boldsymbol{\Pi}$ operates within space bound $f(|x|)$. The class of languages decidable in exponential space by uniform families of P systems of type $\mathcal{D}$ is denoted by $\mathbf{EXPMCSPACE}_{\mathcal{D}}$, while the corresponding class for semi-uniform families is $\mathbf{EXPMCSPACE}^\star_{\mathcal{D}}$. The classes defined in terms of non-confluent P systems are denoted by $\mathbf{NEXPMCSPACE}_{\mathcal{D}}$ and $\mathbf{NEXPMCSPACE}^\star_{\mathcal{D}}$, respectively.

For the precise definitions and properties of Turing machines and, in particular, the space complexity classes **PSPACE** and **EXPSPACE**, we refer the reader to [4].

3 Simulating a Turing machine

In this section we show that exponential-space deterministic Turing machines can be simulated by P systems with restricted elementary active membranes with a polynomial slowdown and a polynomial growth in space.

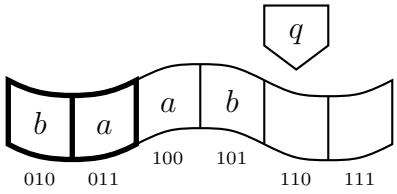
Theorem 1. *Let* M *be a single-tape deterministic Turing machine working in time* $t(n)$ *and space* $s(n)$*, where* $s(n) \leq n + 2^{p(n)}$ *for some polynomial* p*. Then there exists a uniform family of confluent P systems with restricted elementary active membranes* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *operating in time* $O\big(t(n)s(n)\log s(n)\big)$ *and space* $O\big(s(n)\log s(n)\big)$ *such that* $L(\boldsymbol{\Pi}) = L(M)$*.*

We describe how the simulation is carried out by examining a specific example, and generalising from there. Let M be a Turing machine having tape alphabet $\Gamma = \{a, b, \sqcup\}$, where $\sqcup$ denotes a blank tape cell, and using space

$n + 2^n$ (i.e., we choose $p(n) = n$). Also let Q be the set of non-final states of M, and

$$\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{\triangleleft, \triangleright\}$$

its transition function. Assume that M processes the input $x = ba$ of length 2: then M uses a total of $2+2^2 = 6$ tape cells. Suppose that, after a few computation steps, M reaches the following configuration:

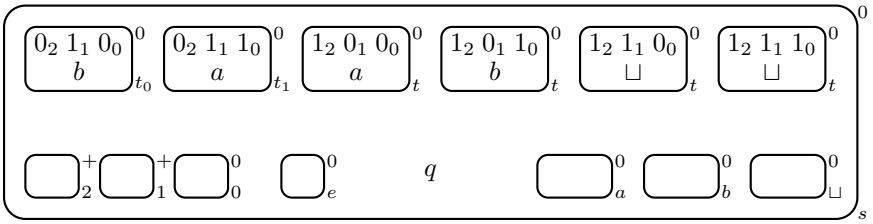


that is, the state of M is q, the tape contains the string *baab* followed by two blank cells, and the tape head is located on the fifth cell. The picture also shows (in binary) the non-standard numbering scheme for tape cells that we employ:

- The first n cells, that initially contain the input (highlighted by a thick border), are denoted by $2^{p(n)} - n, \ldots, 2^{p(n)} - 1$ (e.g., 010 and 011 in our example). These numbers, when written in binary over $p(n) + 1$ bits, all have 0 as their most significant bit.
- The remaining $2^{p(n)}$ cells are denoted by $2^{p(n)}, \ldots, 2 \times 2^{p(n)} - 1$ (e.g., 100 to 111 in our example). These numbers, when written in binary over $p(n) + 1$ bits, all have 1 as their most significant bit.

3.1 Representing the configuration of the Turing machine

The configuration of M described above is encoded in the following configuration $\mathcal{C}_1$ of the P system Π_x simulating it (how this configuration of Π_x is reached from its initial configuration will be described later):



Inside the outermost membrane, labelled by s, we have $n + 2^{p(n)}$ membranes (6 in our example) representing the tape cells of M; n of them are labelled $t_0, \ldots, t_{n-1}$, and the remaining ones (which are generated by membrane division, as described below) by t. We refer to these membranes as *tape-membranes*. Each

tape-membrane contains two pieces of information: a set of $p(n)+1$ (3 in our example) subscripted bits, the *bit-objects*, encoding the number of the tape cell of M it represents (the subscript are used to preserve the order of the bits), and an object taken from the alphabet of M, denoting the symbol written in that tape cell (the *symbol-object*). For instance, the membrane $[1_2 0_1 1_0 b]_t^0$ corresponds to tape cell 101, which contains the symbol b.

The state of M is represented by a *state-object* (q in the example), which will regulate the simulation of each computation step of M. At the beginning of the simulation of each computation step of M, the state-object resides in membrane s.

On the lower-left side of the picture we have $p(n)+1$ membranes, called *position-membranes* and labelled by $p(n), \ldots, 0$, whose electrical charge encodes in binary the current position of the tape head of M; here a positive charge represents a 1 bit, while a neutral charge denotes 0. For instance, in the picture we have $[\,]_2^+ [\,]_1^+ [\,]_0^0$ representing position 110.

The auxiliary membrane labelled by e, the *error-membrane*, will have its charge set to positive whenever Π_x nondeterministically chooses a "wrong" computation path while simulating a computation step of M (see below).

Finally, on the lower-right side of the picture, we have membranes labelled by symbols from the alphabet of M (the *symbol-membranes*). These will be used, once again by setting their charge, to read the symbol currently under the tape head of M.

We shall now describe how to simulate a computation step of M starting from its current configuration, as encoded by Π_x. Later on we will describe how the configuration of Π_x representing the initial configuration of M can be obtained.

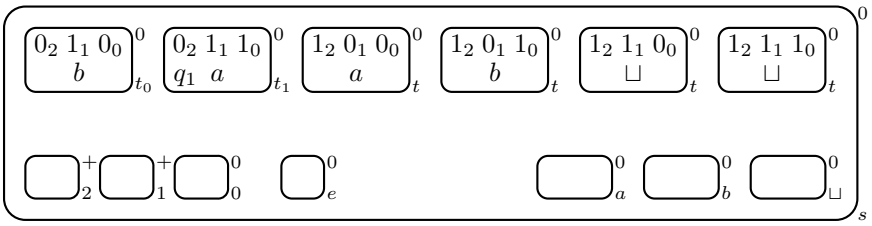
3.2 Simulating a computation step of M

In order to simulate a computation step of M, we need to identify which symbol is located under its tape head; note that the state q is already stored in the state-object. Since most of the tape-membranes of Π_x have the same label t (and those labelled by $t_0, \ldots, t_{n-1}$ behave the same way in this phase, i.e., have the same associated set of rules) there is no way to identify the correct tape-membrane from the outside. Hence, we shall *guess* the tape-membrane corresponding to the cell under the head, then check if selected the right one.

This "guessing" is performed by the state-object, which nondeterministically enters one of the tape membranes using one of the following rules:

$$q\,[\,]_h^0 \to [q_1]_h^0 \qquad \text{for } q \in Q \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\}.$$

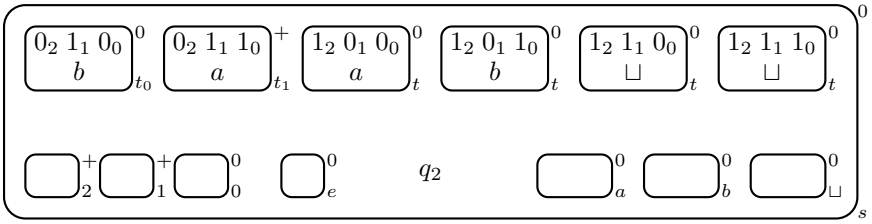
First, suppose q enters the wrong membrane, e.g., 011 instead of 110, producing the following configuration:



The state-object q_1 is immediately sent back out, changing the charge of the membrane to positive using one of the rules

$$[q_1]_h^0 \to [\ ]_h^+\ q_2 \qquad \text{for } q \in Q \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\}.$$

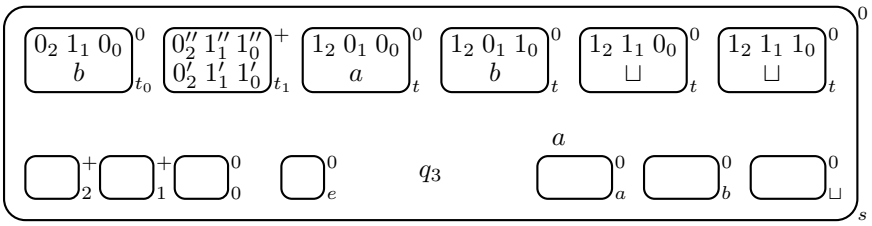
Note that there will always be at most one positive tape-membrane, i.e., the membrane being checked at the current time.



When a tape-membrane is positive, the symbol-object it contains (a in the example) is sent out, while the bit-objects are replicated in a primed and a double-primed versions. At the same time, the state-object waits by increasing its subscript (such waiting steps will be implicit from now on). The corresponding rules are

$$\begin{aligned}
&[\gamma]_h^+ \to [\ ]_h^+\ \gamma && \text{for } \gamma \in \Gamma \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\}\\
&[0_i \to 0_i' 0_i'']_h^+ && \text{for } 0 \le i \le p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\}\\
&[1_i \to 1_i' 1_i'']_h^+ && \text{for } 0 \le i \le p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\}\\
&[q_2 \to q_3]_s^0 && \text{for } q \in Q.
\end{aligned}$$

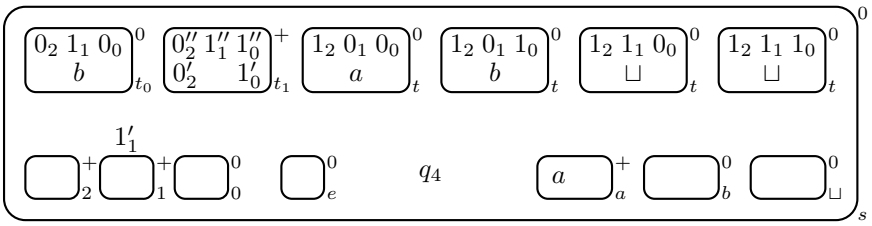
In our example, we obtain the following configuration:

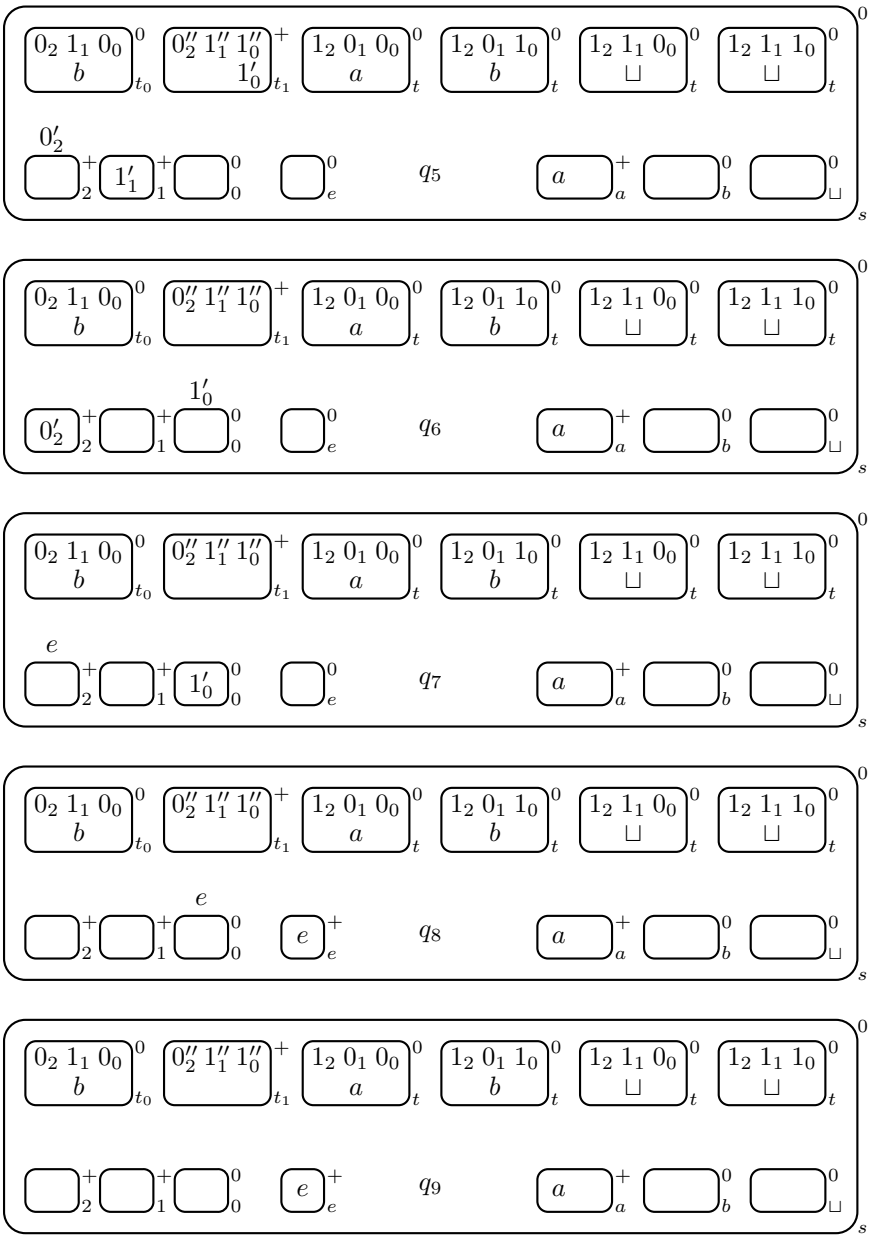


Now, the symbol-object is sent to the corresponding symbol-membrane, setting its charge to positive (thus allowing the state-object to identify the symbol under the tape head). At the same time, the primed bit-objects inside the positive tape-membrane will be sent (in nondeterministic order) to the corresponding position-membranes and compared with their charge. In our example we have $0'_2 1'_1 1'_0$ and $[\,]^+_2 [\,]^+_1 [\,]^0_0$ (where the most significant and the least significant bits differ). If there is a mismatch on a certain bit, the corresponding bit-object will produce an error-object e, otherwise it will be deleted. The error-objects will set the charge of the error-membrane to positive, so that the state-object may identify the error when all comparisons have been made, and will be in turn deleted. This phase, whose duration is $p(n) + 4$ steps, involves the following rules:

$\gamma\,[\,]^0_\gamma \to [\gamma]^+_\gamma$	for $\gamma \in \Gamma$
$[0'_i]^+_h \to [\,]^+_h\, 0'_i$	for $0 \le i \le p(n)$ and $h \in \{t_0, \ldots, t_{n-1}, t\}$
$[1'_i]^+_h \to [\,]^+_h\, 1'_i$	for $0 \le i \le p(n)$ and $h \in \{t_0, \ldots, t_{n-1}, t\}$
$0'_i\,[\,]^\alpha_i \to [0'_i]^\alpha_i$	for $0 \le i \le p(n)$ and $\alpha \in \{0, +\}$
$1'_i\,[\,]^\alpha_i \to [1'_i]^\alpha_i$	for $0 \le i \le p(n)$ and $\alpha \in \{0, +\}$
$[0'_i \to \lambda]^0_i$	for $0 \le i \le p(n)$
$[1'_i \to \lambda]^+_i$	for $0 \le i \le p(n)$
$[0'_i]^+_i \to [\,]^+_i\, e$	for $0 \le i \le p(n)$
$[1'_i]^0_i \to [\,]^0_i\, e$	for $0 \le i \le p(n)$
$e\,[\,]^\alpha_e \to [e]^+_e$	for $\alpha \in \{0, +\}$
$[e \to \lambda]^+_e$	
$[q_j \to q_{j+1}]^0_s$	for $3 \le j \le p(n) + 6$ and $q \in Q$.

In our example, the computation may proceed as follows (for some concrete nondeterministic choices in the order the bit-objects are sent out).



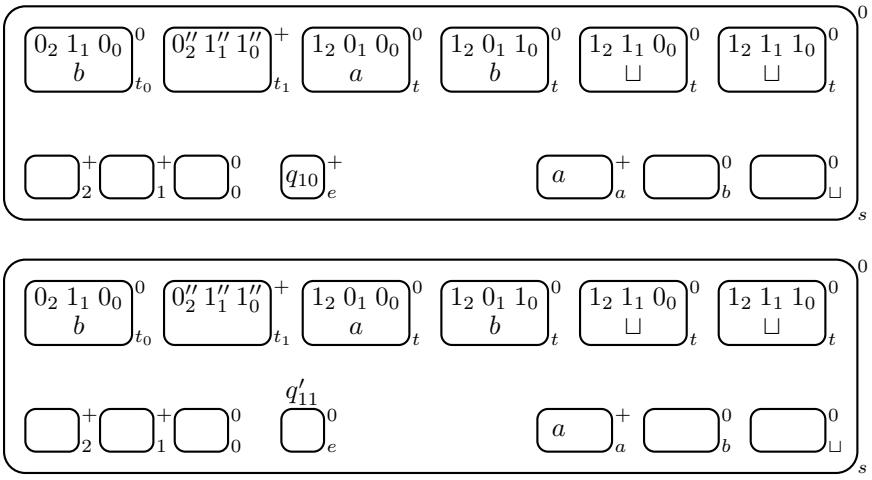


While any remaining error-object is deleted, the state-object may now enter the error-membrane in order to check if a bit mismatch has been found (thus, if the system chose the wrong tape-membrane). It is sent out in a primed version if this is the case, while simultaneously resetting the charge of e to neutral. We

use the following rules:

$$
\begin{array}{ll}
q_{p(n)+7}\ [\ ]_e^{\alpha} \to [q_{p(n)+8}]_e^{\alpha} & \text{for } q \in Q \text{ and } \alpha \in \{0,+\} \\
[q_{p(n)+8}]_e^{+} \to [\ ]_e^{0}\ q'_{p(n)+9} & \text{for } q \in Q.
\end{array}
$$

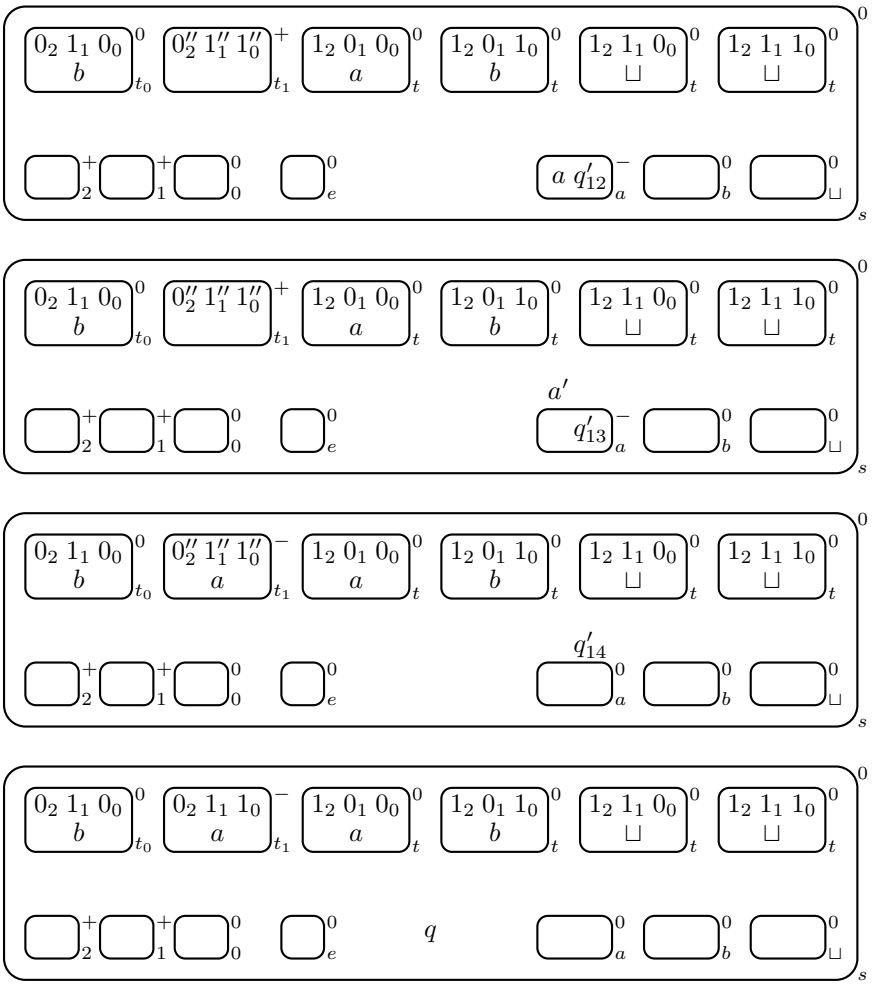
In our example, these two steps produce the following configurations:



Having guessed the wrong tape-membrane, the system must now send the symbol-object back to its original tape-membrane (the only positively charged one); it does so by setting the charge of the symbol-membrane to negative. In the subsequent three steps, the configuration of Π_x is reset to $\mathcal{C}_1$, *except that the tape-membrane we chose is set to negative*. This requires the following rules:

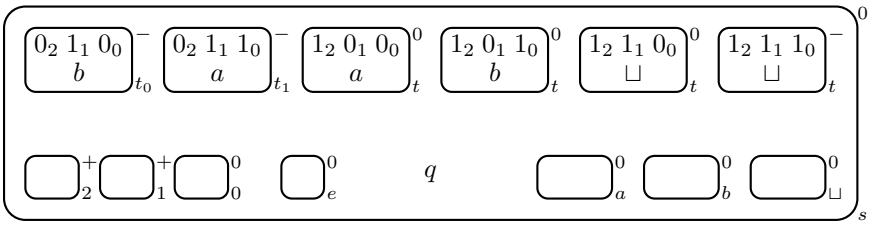
$$
\begin{array}{lll}
q'_{p(n)+9}\ [\ ]_\gamma^{+} \to [q'_{p(n)+10}]_\gamma^{-} & \text{for } q \in Q \text{ and } \gamma \in \Gamma & \\
[\gamma]_\gamma^{-} \to [\ ]_\gamma^{-}\ \gamma' & \text{for } \gamma \in \Gamma & \\
[q'_{p(n)+10} \to q'_{p(n)+11}]_\gamma^{-} & \text{for } q \in Q \text{ and } \gamma \in \Gamma & \\
\gamma'\ [\ ]_h^{+} \to [\gamma]_h^{-} & \text{for } \gamma \in \Gamma \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} & \\
[q'_{p(n)+11}]_\gamma^{-} \to [\ ]_\gamma^{0}\ q'_{p(n)+12} & \text{for } q \in Q \text{ and } \gamma \in \Gamma & \\
[0''_i \to 0_i]_h^{-} & \text{for } 0 \le i \le p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} & (1) \\
[1''_i \to 1_i]_h^{-} & \text{for } 0 \le i \le p(n) \text{ and } h \in \{t_0, \ldots, t_{n-1}, t\} & (2) \\
[q'_{p(n)+12} \to q]_s^{0} & \text{for } q \in Q. &
\end{array}
$$

In the example, we obtain the following sequence of configurations:

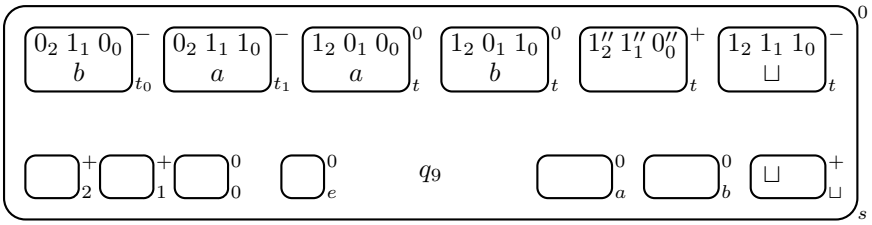


The system can now guess another tape-membrane, repeating the previous steps while making wrong guesses and thus increasing the number of negatively charged tape-membranes (which are ignored during the guessing step). After at most $n + 2^{p(n)} - 1$ wrong guesses (e.g., 5 guesses in our example), the system finally chooses the correct tape-membrane.

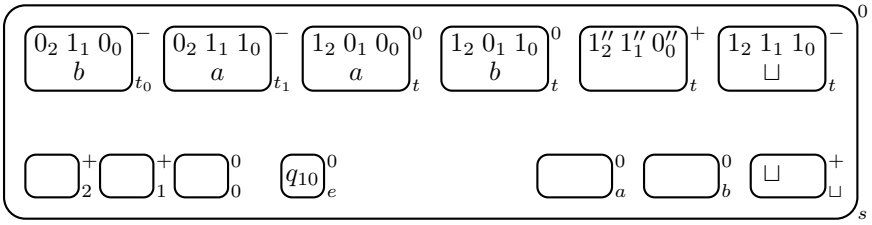
For instance, suppose Π_x made the three consecutive wrong guesses 011, 111 and 010, thus reaching the following configuration:



Now assume that the state-object finally enters the correct membrane 110. The bit-checking phase proceeds as described above for $p(n) + 7$ steps (i.e., 9 steps in our case), making Π_x reach the following configuration:



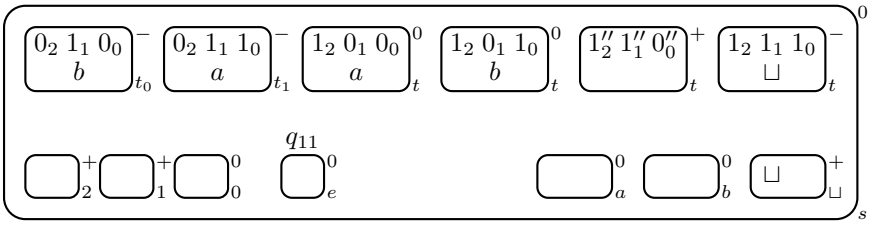
In this configuration the error-membrane is neutral, as all bit-objects match the corresponding position-membranes. The state-object enters membrane e



but this time, since the error-membrane is neutral, it is sent out in a non-primed version, using the rule

$$[q_{p(n)+8}]_e^+ \to [\ ]_e^0\ q_{p(n)+9} \qquad \text{for } q \in Q$$

thus producing the configuration



As before, the state-object is sent to the only positively charged symbol-membrane, but this time it sets it to neutral:

$$q_{p(n)+9}\ [\ ]^{+}_{\gamma} \to [q_{p(n)+10}]^{0}_{\gamma} \qquad \text{for } q \in Q \text{ and } \gamma \in \Gamma.$$

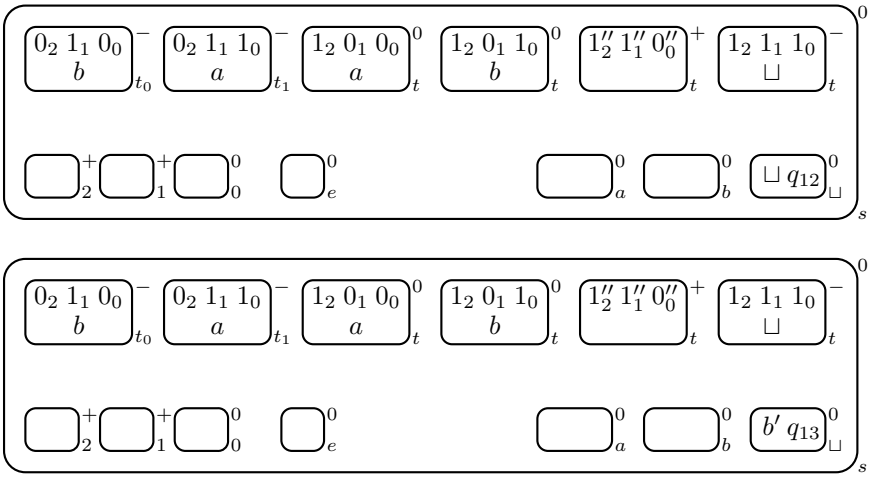
The symbol-object inside responds to this change of charge by deleting itself,

$$[\gamma \to \lambda]^{0}_{\gamma} \qquad \text{for } \gamma \in \Gamma$$

while at the same time the state-object produces the primed version of the new symbol-object corresponding to the symbol written by the Turing machine. Assume that the transition function of M specifies that $\delta(q, \sqcup) = (r, b, \triangleleft)$; the corresponding rules are

$$[q_{p(n)+10} \to q_{p(n)+11}\ \sigma']^{0}_{\gamma} \quad \text{if } \delta(q, \gamma) = (r, \sigma, d) \text{ for some } r \in Q,\ \sigma \in \Gamma,\ d \in \{\triangleleft, \triangleright\}.$$

Hence, our example configuration evolves as follows:

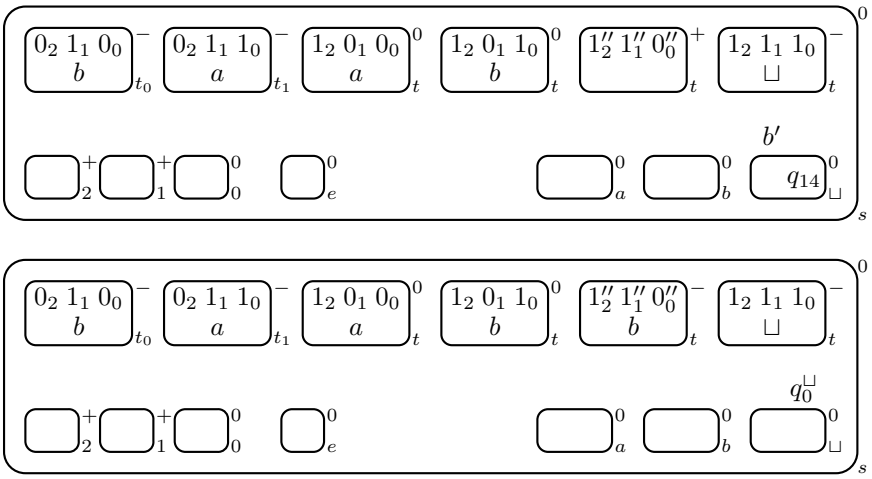


The new primed symbol-object is sent back to the only positive tape-membrane as before (see page 13), while the state-object is sent out as a new state-object q_0^{γ}, having the tape symbol as a superscript and a new counter, starting from 0, as a subscript (there will be no conflict with the previous rules due to the new superscript):

$$[\sigma']^{0}_{\gamma} \to [\ ]^{0}_{\gamma}\ \sigma' \qquad \text{for } \gamma, \sigma \in \Gamma$$

$$[q_{p(n)+11} \to q_{p(n)+12}]^{0}_{\gamma} \qquad \text{for } q \in Q \text{ and } \gamma \in \Gamma$$

$$[q_{p(n)+12}]^{0}_{\gamma} \to [\ ]^{0}_{\gamma}\ q_0^{\gamma} \qquad \text{for } q \in Q \text{ and } \gamma \in \Gamma.$$



While the doubly-primed bit-objects are reset to their initial state as described earlier, the state-object begins to update the position-membranes, reflecting the movement of the tape head. Recall that incrementing a binary counter means flipping its bits, from the least to the most significant one, until a 0 is flipped into an 1 (i.e., the remaining bits are left unchanged). Similarly, decrementing it means flipping its bits in that order until a 1 is flipped into a 0. The subscript of the state-object, initially 0, records the next bit position to flip. The flipping operation is carried out by entering and exiting the corresponding position-membrane and updating its charge. When a 0 has been flipped into an 1 (for an increment), or a 1 into a 0 (for a decrement), the subscript of the state-object becomes $p(n) + 1$, thus leaving the subsequent bits unchanged.

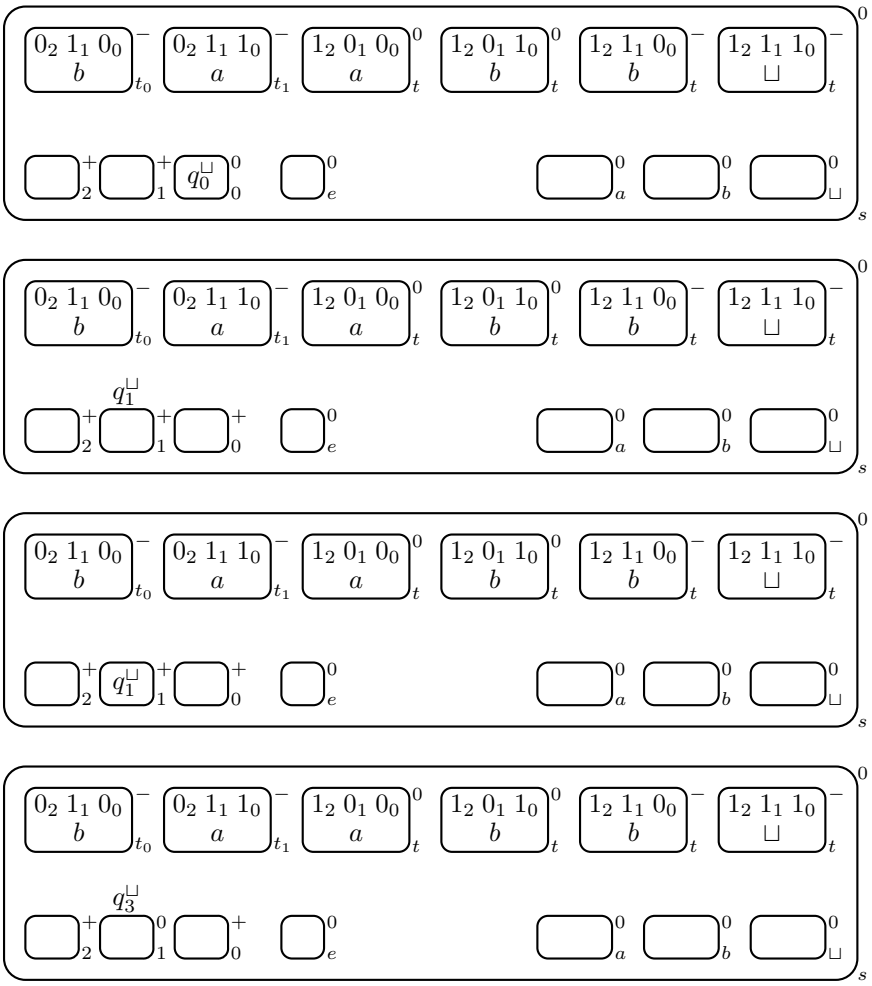
If $\delta(q, \gamma) = (r, \sigma, \triangleright)$ for some $r \in Q$, $\sigma \in \Gamma$ (i.e., the tape head moves to the right), then the position-updating procedure is performed via the following rules:

$$
\begin{array}{ll}
q_i^{\gamma}\ [\ ]_i^{\alpha} \to [q_i^{\gamma}]_i^{\alpha} & \text{for } \gamma \in \Gamma,\ q \in Q,\ \alpha \in \{0, +\} \text{ and } 0 \le i \le p(n) \\
[q_i^{\gamma}]_i^{0} \to [\ ]_i^{+}\ q_{p(n)+1}^{\gamma} & \text{for } \gamma \in \Gamma,\ q \in Q \text{ and } 0 \le i \le p(n) \\
[q_i^{\gamma}]_i^{+} \to [\ ]_i^{0}\ q_{i+1}^{\gamma} & \text{for } \gamma \in \Gamma,\ q \in Q \text{ and } 0 \le i \le p(n)
\end{array}
$$

If $\delta(q, \gamma) = (r, \sigma, \triangleleft)$ for some $r \in Q$, $\sigma \in \Gamma$ (i.e., the tape head moves to the left), then the rules are:

$$
\begin{array}{ll}
q_i^{\gamma}\ [\ ]_i^{\alpha} \to [q_i^{\gamma}]_i^{\alpha} & \text{for } \gamma \in \Gamma,\ q \in Q,\ \alpha \in \{0, +\} \text{ and } 0 \le i \le p(n) \\
[q_i^{\gamma}]_i^{0} \to [\ ]_i^{+}\ q_{i+1}^{\gamma} & \text{for } \gamma \in \Gamma,\ q \in Q \text{ and } 0 \le i \le p(n) \\
[q_i^{\gamma}]_i^{+} \to [\ ]_i^{0}\ q_{p(n)+1}^{\gamma} & \text{for } \gamma \in \Gamma,\ q \in Q \text{ and } 0 \le i \le p(n)
\end{array}
$$

In our example we have to decrement the head position from 110 to 101 by flipping only the two least significant bits (notice how the subscript 2 is skipped):

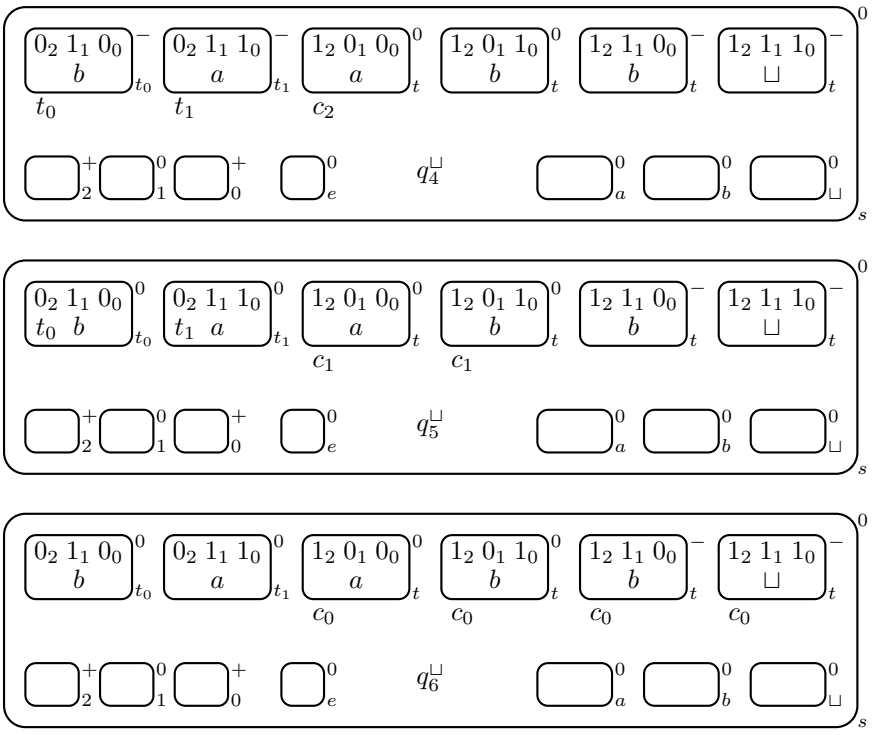


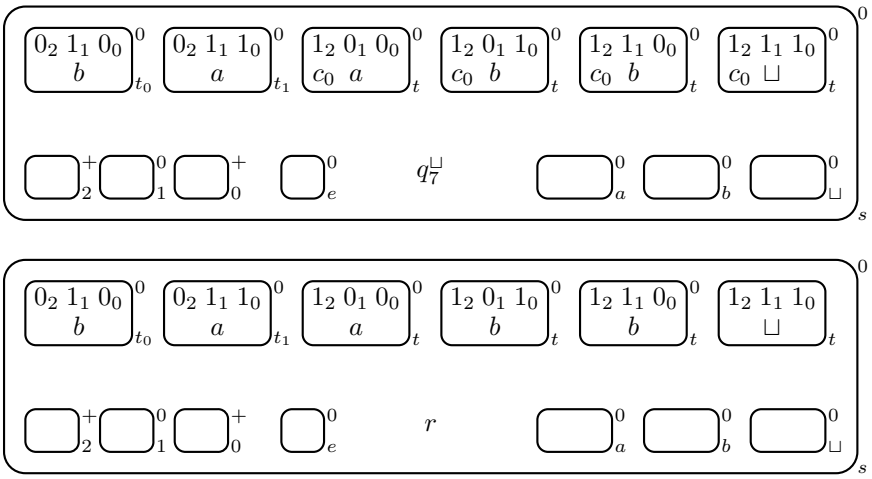
When the subscript of q^γ reaches $p(n)+1$ (i.e., when the position updating has been completed) the system begins preparing the encoding of next configuration of M. This requires resetting all charges of tape-membranes to neutral: this is accomplished by creating n objects $t_0, \ldots, t_{n-1}$ (resetting the membranes having the same name) and $2^{p(n)}$ copies of object c_0 (resetting the membranes having label t). The latter objects are created by an initial object $c_{p(n)}$, which is rewritten as two copies of $c_{p(n)-1}$, each of them rewritten as two copies of $c_{p(n)-2}$, and so on. The state-object waits for this process to terminate, and then finally

becomes the new state of M, as described by the transition function.

$[q^{\gamma}_{p(n)+1} \to q^{\gamma}_{p(n)+2}\ t_0 \cdots t_{n-1}\ c_{p(n)}]^0_s$	for $q \in Q$ and $\gamma \in \Gamma$
$[q^{\gamma}_{p(n)+k} \to q^{\gamma}_{p(n)+k+1}]^0_s$	for $q \in Q$, $\gamma \in \Gamma$ and $2 \le k \le p(n)+2$
$t_j\ [\ ]^{\alpha}_{t_j} \to [t_j]^0_{t_j}$	for $0 \le j \le n-1$ and $\alpha \in \{0, -\}$
$[t_j \to \lambda]^0_{t_j}$	for $0 \le j \le n-1$
$[c_i \to c_{i-1}c_{i-1}]^0_s$	for $1 \le i \le p(n)$
$c_0\ [\ ]^{\alpha}_t \to [c_0]^0_t$	for $\alpha \in \{0, -\}$
$[c_0 \to \lambda]^0_t$	
$[q^{\gamma}_{2p(n)+3} \to r]^0_s$	for $q, r \in Q$ and $\gamma \in \Gamma$, if $\delta(q, \gamma) = (r, \sigma, d)$ for some $\sigma \in \Gamma$ and $d \in \{\triangleleft, \triangleright\}$.

In our example, the computation evolves as follows:

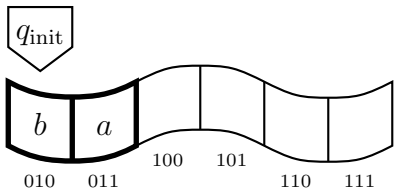




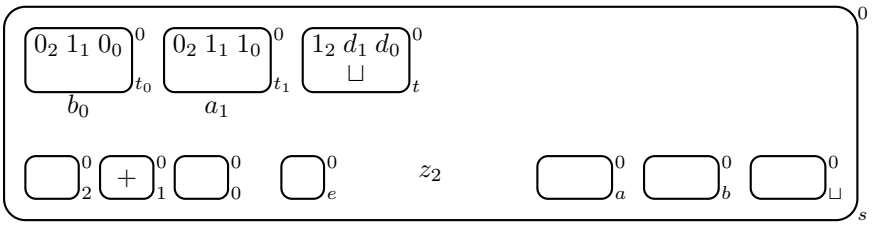
We have finally reached the configuration of Π_x corresponding to the configuration of M after it has performed its computation step, and we are ready to start simulating a new step of M.

3.3 Creating the initial configuration

In the previous section we described how to simulate a computation step of M starting from an arbitrary configuration of the Turing machine. We still need to describe how to encode the *initial* configuration of M (represented in the following picture) in the P system Π_x simulating it.



We use the following as the initial configuration *of the P system*:



This initial configuration consists of a membrane s containing:

- Membranes $t_0, \ldots, t_{n-1}$, each containing $p(n)+1$ bit-objects encoding the position numbers of the input cells (as described above).
- One single copy of membrane t, containing the bit-object $1_{p(n)}$ (recall that the most significant bit for the non-input tape cells is always 1) and the "bit variables" $d_0, \ldots, d_{p(n)-1}$.
- The position-membranes, labelled by $0, \ldots, p(n)+1$, whose initial charge is 0 by definition. Those which have to be set to positive in order to set up the initial head position (i.e., $2^{p(n)} - n$) contain a + object.
- The error-membrane e.
- The symbol-membranes, labelled by the elements of Γ.
- The object $z_{p(n)}$.

All these items only depend on the *size* of the input of the Turing machine M. The input itself is encoded by a set of objects denoting the symbols, subscripted by an index indicating their position in the string (counting from 0), and placed into the input membrane s. In our example, the input ba of M is encoded in Π_x as $b_0 a_1$.

During the initialisation phase of Π_x, several operations are carried out. First of all, the input-objects are sent to the corresponding tape-membranes (indicated in their subscripts). This is accomplished by using the following rules:

$$\gamma_i \, [\,]^0_{t_i} \to [\gamma]^0_{t_i} \qquad \text{for } \gamma \in \Gamma \text{ and } 0 \le i \le n-1.$$

The position-membranes have their charges set to + by sending out the + objects, which are then deleted using

$$\begin{aligned} &[+]^0_i \to [\,]^+_i + && \text{for } 0 \le i \le p(n) \\ &[+ \to \lambda]^0_s. \end{aligned}$$

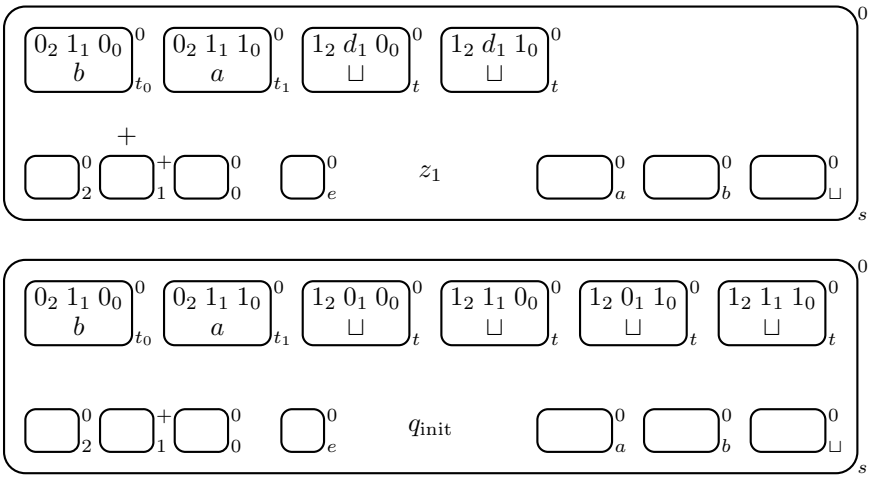
The tape-membranes corresponding to the working portion of the tape, of size $2^{p(n)}$, are created by iterated elementary membrane division, starting from the single initial tape-membrane t. The objects d_i are rewritten as 0_i on one side, and as 1_i on the other, whenever the membrane is divided. This process creates all the $2^{p(n)}$ cell numbers in binary. The corresponding rules are

$$[d_i]^0_t \to [0_i]^0_t \; [1_i]^0_t \qquad \text{for } 0 \le i \le p(n)-1.$$

This latter operation requires $p(n)$ steps. The object $z_{p(n)}$ has its subscript decremented to 1, and then finally becomes the state-object corresponding to the initial state of M:

$$\begin{aligned} &[z_i \to z_{i-1}]^0_s && \text{for } 2 \le i \le p(n) \\ &[z_1 \to q_{\text{INIT}}]^0_s. \end{aligned}$$

In our example, the initialisation phase proceeds as follows.



After having initialised the P system Π_x according to the initial configuration of M on input x, the simulation is carried out step-by-step as described in the previous section.

3.4 Halting and output

The only missing part of our simulation concerns the operations to carry out when the simulated machine halts by accepting or rejecting. Assuming the transition function δ of M is undefined on its accepting state q_{YES} and its rejecting state q_{NO}, we can simply proceed as follows: if the machine enters q_{YES}, then in Π_x the state-object q_{YES} appears inside the outermost membrane s; we can then send that object out to the environment as YES to make Π_x accept. The behaviour is analogous for the rejecting state q_{NO}.

$$[q_{\text{YES}}]^0_s \to [\ ]^0_s \text{ YES} \qquad\qquad [q_{\text{NO}}]^0_s \to [\ ]^0_s \text{ NO}.$$

3.5 Completing the proof

The initialisation phase of Π_x, simulating M on an input x of length n, requires $O(p(n))$ time in order to create $2^{p(n)}$ copies of membrane t by a sequence of elementary divisions.

Then, the $t(n)$ steps performed by M are simulated. Each step involves guessing the tape-membrane corresponding to the cell currently under the tape head; each simulated step may require up to $O(s(n))$ guesses in the worst case. For each guess, we need to check if the correct tape-membrane was selected, and this requires time proportional to the number of bit positions, i.e., $O\big(\log s(n)\big)$ steps. If the membrane is incorrect, then $O(1)$ steps are required to set its charge to negative and prepare the system for a further guess. If the membrane was the right one, the state, head position and tape symbol have to be updated, and this

requires further $O\big(\log s(n)\big)$ steps. Hence, each simulated step of M requires $O\big(s(n)\log s(n)\big)$ steps of Π_x, for a total of $O\big(t(n)s(n)\log s(n)\big)$ steps.

As the output step only requires constant time, the whole simulation can be carried out in $O\big(t(n)s(n)\log s(n)\big)$ time. Since $s(n)$ is $O(t(n))$ for a Turing machine (assuming it at least reads its whole input), the simulation time can be expressed as a function of $t(n)$ as $O\big(t(n)^2\log t(n)\big)$. Hence, this is an "efficient" simulation: if M works in polynomial time, then the family $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ simulating it also works in polynomial time; if M runs in exponential (resp., doubly-exponential) time, then $\boldsymbol{\Pi}$ also runs in exponential (resp., doubly-exponential time).

Notice that the actual running time of the simulation depends on the sequence of nondeterministic choices performed when the system has to guess the correct tape-membrane. In the best case, when the correct guess is always the first one, the time reduces to $O\big(t(n)\log s(n)\big)$ instead of $O\big(t(n)s(n)\log s(n)\big)$ as in the worst case.

The space required by Π_x is asymptotically due to the tape-membranes. These are $s(n)$ in number, and each of them contains $O(\log s(n))$ bit-objects denoting its position on the tape. Hence, the simulation requires $O\big(s(n)\log s(n)\big)$ space: a polynomial-space Turing machine is simulated in polynomial space, and an exponential-space one in exponential space.

In order to complete the proof of Theorem 1, we only need to check that the family $\boldsymbol{\Pi}$ is polynomial-time uniform. It is easy to verify that all the rules and the initial configuration of Π_x actually depend only on the length of x (except for the input objects). There is a constant number of different kinds of rules parametric with respect to n or $p(n)$; the larger sets of rules are (1) and (2) on page 12, consisting of $O\big(n \times p(n)\big)$ rules each.

4 Characterising exponential space

In the previous section we described a simulation of deterministic Turing machines working in exponential space by means of P systems. Combining this result with the converse simulation illustrated in [8], we can show that the computational power of Turing machines and of P systems with active membranes coincide when these devices operate within an exponential space limit:

Corollary 1. *The following inclusions hold:*

$$\begin{array}{ccc} \mathbf{EXPMCSPACE}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})} & \subseteq & \mathbf{EXPMCSPACE}^{[\star]}_{\mathcal{AM}} \\ \cup\mathsf{I} & & \mathsf{I}\cap \\ \mathbf{EXPSPACE} & \supseteq & \mathbf{NEXPMCSPACE}^{\star}_{\mathcal{AM}} \end{array}$$

where $[\star]$ *denotes optional semi-uniformity (instead of uniformity). Hence, all classes shown in the diagram coincide.*

Proof. The chain of inclusions

$$\mathbf{EXPMCSPACE}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})} \subseteq \mathbf{EXPMCSPACE}^{[\star]}_{\mathcal{AM}} \subseteq \mathbf{NEXPMCSPACE}^{\star}_{\mathcal{AM}}$$

holds by definition. That $\mathbf{NEXPMCSPACE}^{\star}_{\mathcal{AM}} \subseteq \mathbf{EXPSPACE}$ is an immediate corollary of Theorem 5 in [8]. Finally, the inclusion of **EXPSPACE** in $\mathbf{EXPMCSPACE}_{\mathcal{AM}(-\mathrm{d},-\mathrm{n})}$ directly follows from Theorem 1. □

Let us remark that the power of the complexity class $\mathbf{EXPMCSPACE}_{\mathcal{AM}}$ is mostly due to the families of P systems themselves, as opposed to the Turing machines providing the uniformity condition; indeed, these would only be able to solve the strictly smaller [4] class **P** of decision problems.

5 Conclusions

We showed that the class of problems solvable by P systems with active membranes in exponential space coincides with the class of problems solved by Turing machines in exponential space, that is, $\mathbf{EXPMCSPACE}_{\mathcal{AM}} = \mathbf{EXPSPACE}$.

Again, the techniques used to prove this result cannot be applied immediately when the space bound is less strict, i.e., super-exponential. In fact, in this case we would need systems using indexed bits, where the index ranges over a super-polynomial set of values; as a consequence, such systems cannot be generated in a uniform way in a polynomial number of steps, as requested by Definition 3. Thus, it remains open for this case the question whether these kinds of P systems with active membranes have the same computing power as Turing machines working under the same space constraints.

Let us note that if membrane creation [1] is used instead of membrane division, then the simulation may be straightforward and faster (the slowdown would be by a constant factor only). The simulation would also be deterministic, instead of requiring "wild" nondeterminism as in our result. Turing machine cells may be represented by *nested* membranes, created when needed; this is a construction that generalises even to super-exponential space. However, with membrane division only, the depth of membrane hierarchy cannot increase during the computation, and it is originally polynomial under our current definition.

As a direction for future research, it might also be interesting to analyse the behaviour of families of P systems with active membranes working in logarithmic space. However, there are two major issues to be considered in this case: first, we should slightly change the notion of space complexity, in order to allow for a "read-only" input multiset that is not counted when the space required by the P system is measured (similarly to the input tape of a logspace Turing machine). Furthermore, the notion of uniformity used to define the families of P systems should be weakened, since polynomial-time Turing machines constructing the families might be able to solve the problems altogether by themselves. More general forms of uniformity have already been investigated [3], and that work is going to be useful when attacking this problem.

Acknowledgements

Artiom Alhazov gratefully acknowledges the project RetroNet by the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618). The work of the other authors was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo per la Ricerca (FAR) 2011.

References

1. Artiom Alhazov, Rudolf Freund, and Agustín Riscos-Núñez. Membrane division, restricted membrane creation and object complexity in P systems. *International Journal of Computer Mathematics*, 83(7):529–547, 2006.
2. Artiom Alhazov, Carlos Martín-Vide, and Linqiang Pan. Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae*, 58(2):67–77, 2003.
3. Niall Murphy and Damien Woods. The computational power of membrane systems under tight uniformity conditions. *Natural Computing*, 10(1):613–632, 2011.
4. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993.
5. Mario J. Pérez-Jiménez, Álvaro Romero-Jiménez, and Fernando Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2(3):265–284, 2003.
6. Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. Introducing a space complexity measure for P systems. *International Journal of Computers, Communications & Control*, 4(3):301–310, 2009.
7. Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. P systems simulating oracle computations. In Marian Gheorghe, Gheorghe Păun, Arto Salomaa, Grzegorz Rozenberg, and Sergey Verlan, editors, *Membrane Computing, 12th International Conference, CMC 2011*, volume 7184 of *Lecture Notes in Computer Science*, pages 346–358. Springer, 2011.
8. Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron. P systems with active membranes working in polynomial space. *International Journal of Foundations of Computer Science*, 22(1):65–73, 2011.
9. Gheorghe Păun. P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.
10. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
11. Petr Sosík. The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2(3):287–298, 2003.
12. Claudio Zandron, Claudio Ferretti, and Giancarlo Mauri. Solving NP-complete problems using P systems with active membranes. In Ioannis Antoniou, Cristian S. Calude, and Michael J. Dinneen, editors, *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference*, pages 289–301. Springer, 2001.