Simulating Turing Machines with Polarizationless P Systems with Active Membranes

Zsolt Gazdag[1], Gábor Kolonits[1], and Miguel A. Gutiérrez-Naranjo[2]

[1] Department of Algorithms and Their Applications, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary
{gazdagzs,kolomax}@inf.elte.hu
[2] Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, 41012 Sevilla, Spain
magutier@us.es

Abstract. We prove that every single-tape deterministic Turing machine working in $t(n)$ time, for some function $t : \mathbb{N} \to \mathbb{N}$, can be simulated by a uniform family of polarizationless P systems with active membranes. Moreover, this is done without significant slowdown in the working time. Furthermore, if $\log t(n)$ is space constructible, then the members of the uniform family can be constructed by a family machine that uses $O(\log t(n))$ space.

1 Introduction

The simulation of the behaviour of Turing machines by families of P systems has a long tradition in Membrane Computing (see, e.g., [1,8,11,13]). The purpose of such simulations is twofold. On the one hand, they allow to prove new properties on complexity classes and, on the other hand, they provide constructive proofs of results which have been proved via indirect methods[1].

In this paper, we give a new step on the second research line, by showing that Turing machines can be simulated efficiently by families of polarizationless P systems with active membranes. By efficiency we mean that these P systems can simulate Turing machines without significant slowdown in the working time. Moreover, the space complexity of the presented P systems is quadratic in the time complexity of the Turing machine.

The conclusions obtained from such simulations are well-known: the decision problems solved by Turing machines can also be solved by families of devices in the corresponding P system models. However, one has to be careful when giving such simulations. It is well known, for example, that the solution of a decision problem X belonging to the complexity class **P** via a polynomially uniform

[1] The reader is supposed to be familiar with standard techniques and notations used in Membrane Computing. For a detailed description see [10].

family of recognizer P systems is trivial, since the polynomial encoding of the input can involve the solution of the problem (see [3,7]).

This fact can be generalized to wider situations: the solution of a decision problem X by a uniform family of P systems $\mathbf{\Pi}$ may be trivial in the following sense. Let us consider a Turing machine that computes the encoding of the instances of X (also called the *encoding machine*). If this machine is powerful enough to decide if an instance is a positive instance of X or not, then a trivial P system can be used to send out to the environment the correct answer.

In order to avoid such trivial solutions, the encoding machine and the Turing machine that computes the members of $\mathbf{\Pi}$ (often called the *family machine*) should be reasonably weak. More precisely, if the problem X belongs to a complexity class $\mathcal{C}$, then the family machine and the encoding machine should belong to a class of Turing machines that can compute only a strict subclass of $\mathcal{C}$ (see [8]).

According to this, to simulate a Turing machine M working in $t(n)$ time, for some function $f : \mathbb{N} \to \mathbb{N}$ such that $\log t(n)$ is space constructible, we will use a family of P systems whose members can be constructed by a family machine using $O(\log t(n))$ space. In particular, if t is a polynomial, then the family machine uses logarithmic space. Moreover, we will use the following function *pos* to encode the input words of M: For a given input word w, $pos(w)$ is a multiset where every letter of w is coupled with its position in w. Furthermore, the positions of the letters are encoded in binary words. It was discussed in [8] that *pos* is computable by deterministic random-access Turing machines using logarithmic time (in other words, *pos* is **DLOGTIME** computable). In this way, there is no risk that *pos* can compute a solution of a problem outside of **DLOGTIME**. Since **DLOGTIME** is a rather small complexity class, it follows that we can use *pos* safely as the input encoding function during the simulation of M.

The result presented in this paper resembles the one appearing in [1] stating that every single-tape deterministic Turing machine can be simulated by uniform families of P systems with active membranes with a cubic slowdown and quadratic space overhead. However, this result and ours are not directly comparable, as the constructions in [1] use the polarizations of the membranes, while our solution does not.

The paper is organized as follows. First of all, we recall some basic definitions used along the paper. Then, in Section 3, we present the main result. Finally, we give some concluding remarks in Section 4.

2 Preliminaries

First, we recall some basic concepts used later.

Alphabets, Words, Multisets. An *alphabet* Σ is a non-empty and finite set of symbols. The elements of Σ are called *letters* and Σ^* denotes the set of all finite *words* (or *strings*) over Σ, including the *empty word* ε. The length of a word $w \in \Sigma^*$ is denoted by $l(w)$. We will use *multisets* of objects in the membranes

of a P system. As usual, these multisets will be represented by strings over the object alphabet of the P system.

The set of natural numbers is denoted by $\mathbb{N}$. For $i, j \in \mathbb{N}$, $[i, j]$ denotes the set $\{i, i+1, \ldots, j\}$ (notice that if $j < i$, then $[i, j] = \varnothing$). For the sake of simplicity, we will write $[n]$ instead of $[1, n]$. For a number $i \in \mathbb{N}$, $b(i)$ denotes its binary form and $b(\mathbb{N}) = \{b(i) \mid i \in \mathbb{N}\}$. Given an alphabet Σ, the function $pos : \Sigma^* \to (\Sigma \times b(\mathbb{N}))^*$ is defined in the following way. For a word $w = a_1 \ldots a_n \in \Sigma^*$, where $a_i \in \Sigma$, $i \in [n]$, $pos(w) := (a_1, b(1)) \ldots (a_n, b(n))$. If a is the ith letter of w, then we will also write the ith letter of $pos(w)$ in the form $a_{b(i)}$.

Turing Machines. Turing machines are well known computational devices. In the following we describe the variant appearing, e.g., in [12]. A *(deterministic) Turing machine* is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where

- Q is the finite set of *states*,
- Σ is the *input alphabet*,
- Γ is the *tape alphabet* including Σ and a distinguished symbol $\sqcup \notin \Sigma$, called the *blank symbol*,
- $\delta : (Q - \{q_a, q_r\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the *transition function*,
- $q_0 \in Q$ is the *initial state*,
- $q_a \in Q$ is the *accepting state*,
- $q_r \in Q$ is the *rejecting state*.

M works on a single infinite tape that is closed on the left-hand side. During the computation of M, the tape contains only finitely many non-blank symbols, and it is blank everywhere else. Let us consider a word $w \in \Sigma^*$. The initial configuration of M on w is the configuration where w is placed at the beginning of the tape, the head points to the first letter of w, and the current state of M is q_0. A configuration step performed by M can be described as follows. If M is in state p and the head of M reads the symbol X, then M can change its state to q and write X' onto X if and only if $\delta(p, X) = (q, X', d)$, for some $d \in \{L, R\}$. Moreover, if $d = R$ (resp. $d = L$), then M moves its head one cell to the right (resp. to the left) (as usual, M can never move the head off the left-hand end of the tape even if the head points to the first cell and $d = L$). We say that M accepts (resp. rejects) w, if M can reach from the initial configuration on w the accepting state q_a (resp. the rejecting state q_r). Notice that M can stop only in these states. The language accepted by M is the set $L(M)$ consisting of those words in Σ^* that are accepted by M. It is said that M works in $t(n)$ time ($t : \mathbb{N} \to \mathbb{N}$) if, for every word $w \in \Sigma^*$, w stops on w after at most $t(l(w))$ steps; M works using $s(n)$ space ($s : \mathbb{N} \to \mathbb{N}$) if it uses at most $s(n)$ cells when it is started on an input word with length n. As usual, if M is a multi-tape Turing machine and it does not write any symbol on its input tape, then those cells that are used to hold the input word are not counted when the space complexity of M is measured[2]. Let us consider a function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n)$ is at least

[2] For the formal definitions of the well known complexity classes concerning Turing machines (such as **L**, **P**, **TIME**$(t(n))$ and **SPACE**$(s(n))$), the interested reader is referred to [12].

$O(\log n)$. We say that f is *space constructible* if there is a Turing machine M that works using $O(f(n))$ space and M always halts with the unary representation of $f(n)$ on its tape when started on input 1^n.

Recognizer P Systems. A *P system* is a construct of the form $\Pi = (\Gamma, H, \mu, w_1, \ldots, w_m, R)$, where $m \geq 1$ (the *initial degree* of the system); Γ is the *working alphabet of objects*; H is a finite set of *labels* for membranes; μ is a *membrane structure* (a rooted tree), consisting of m membranes, labelled with elements of H; $w_1, \ldots, w_m$ are strings over Γ, describing the *initial multisets of objects* placed in the m regions of μ; and R is a finite set of *developmental rules*.

A *P system with input* is a tuple (Π, Σ, i_0), where Π is a P system with working alphabet Γ, with m membranes, and initial multisets $w_1, \ldots, w_m$ associated with them; Σ is an (input) alphabet strictly contained in Γ; the initial multisets are over $\Gamma - \Sigma$; and i_0 is the label of a distinguished (input) membrane.

We say that Π is a *recognizer P system* [4,5] if Π is a P system with input alphabet Σ and working alphabet Γ; Γ has two designated objects *yes* and *no*; every computation of Π halts and sends out to the environment either *yes* or *no*, but not both, and this is done exactly in the last step of the computation; and, for a word $w \in \Sigma^*$, called the *input of* Π, w can be added to the system by placing it into the input membrane i_0 in the initial configuration.

A P system Π is *deterministic* if it has only a single computation from its initial configuration to its unique halting configuration. Π is *confluent* if every computation of Π halts and sends out to the environment the same object. Notice that, by definition, recognizing P systems are confluent.

P Systems with Active Membranes. In this paper, we investigate recognizer P systems with active membranes [9]. These systems have the following types of rules. As we are dealing with P systems that do not use the polarizations of the membranes, we leave out this feature from the definition.

(a) $[a \to v]_h$, for $h \in H, a \in \Gamma, v \in \Gamma^*$
(object evolution rules, associated with membranes and depending on the label of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part in the application of these rules nor are they modified by them);

(b) $a[\ ]_h \to [b]_h$, for $h \in H$, $a, b \in \Gamma$
(*send-in* communication rules, sending an object into a membrane, maybe modified during this process);

(c) $[a]_h \to [\ ]_h u$, for $h \in H$, $a \in \Gamma$, $u \in \Gamma^*$
(*send-out* communication rules; an object is sent out of the membrane, maybe modified during this process);

(d) $[a]_h \to u$, for $h \in H$, $a \in \Gamma$, $u \in \Gamma^*$
(membrane dissolving rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(e) $[a]_h \to [b]_h [c]_h$, for $h \in H$, $a, b, c \in \Gamma$
(division rules for elementary membranes; in reaction with an object, the

membrane is divided into two membranes; the object a specified in the rule is replaced in the two new membranes by (possibly new) objects b and c respectively, and the remaining objects are duplicated; the new membranes have the same labels as the divided one).

We note that we use the rules of type (c) and (d) in a slightly generalized way as we allow here an object a to evolve into an arbitrary string u over Γ. It is clear, however, that an application of a rule $[a]_h \to [\ ]_h u$ (resp. a rule $[a]_h \to u$) can be simulated by applying a rule of the form $[a]_h \to [\ ]_h b$ (resp. $[a]_h \to b$), where $b \in \Gamma$, and an object evolution rule. Thus, the use of these generalized rules will not speed up the running time of our P systems significantly.

As usual, a P system with active membranes works in a *maximally parallel* manner:

- In one step, any object of a membrane that can evolve must evolve, but one object can be used by only one rule in (a)-(e);
- when some rules in (b)-(e) can be applied to a certain membrane, then one of them must be applied, but a membrane can be the subject of only one of these rules during each step.

We will use uniform families of P system to decide a language $L \subseteq \Sigma^*$. In this paper, we follow the notion of uniformity used in [8]. Let E and F be classes of computable functions. A family $\mathbf{\Pi} = (\Pi(i))_{i\in\mathbb{N}}$ of recognizing P systems is called (E, F)-*uniform* if and only if (i) there is a function $f \in F$ such that, for every $n \in \mathbb{N}$, $\Pi(n) = f(1^n)$ (i.e., f maps the unary representation of each natural number to an encoding of the P system processing all the inputs of length n); (ii) there is a function $e \in E$ that maps every word $x \in \Sigma^*$ to a multiset $e(x) = w_x$ over the input alphabet of $\Pi(l(x))$.

An (E, F)-*uniform* family of P systems $\mathbf{\Pi} = (\Pi(i))_{i\in\mathbb{N}}$ *decides a language* $L \subseteq \Sigma^*$ if, for every word $x \in \Sigma^*$, starting $\Pi(l(x))$ with w_x in its input membrane, $\Pi(l(x))$ sends out to the environment *yes* if and only if $x \in L$. In general, E and F are well known complexity classes such as $\mathbf{P}$ or $\mathbf{L}$.

We say that $\Pi(n)$ *works in* $t(n)$ *time* ($t : \mathbb{N} \to \mathbb{N}$) if $\Pi(n)$ halts in at most $t(n)$ steps, for every input multiset in its input membrane. Next, we adopt the notion of space complexity for families of recognizer P systems similarly to the definition appearing in [8] (see also [6]). Let $\mathcal{C}$ be a configuration of a P system Π. The *size of* $\mathcal{C}$ (denoted by $|\mathcal{C}|$) is the sum of the number of membranes and the total number of objects in $\mathcal{C}$. If $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a halting computation of Π, then the space required by $\mathcal{C}$ is defined as $|\mathcal{C}| = max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\}$. The space required by Π is $|\Pi| = sup\{|\mathcal{C}| \mid \mathcal{C}$ is a halting computation of $\Pi\}$. Let us note that in this paper the presented P systems will have finitely many different halting computations. This clearly implies that $|\Pi| \in \mathbb{N}$. Finally, $\Pi(n)$ *works using* $s(n)$ space ($s : \mathbb{N} \to \mathbb{N}$), if $|\Pi(n)| \leq s(n)$, for every input multiset in its input membrane.

3 The Main Result

In this section we prove the following result:

Theorem 1. *Let $t : \mathbb{N} \to \mathbb{N}$ be a function such that $\log t(n)$ is space constructible and consider a Turing machine M working in $t(n)$ time. Then M can be simulated by a $(\mathbf{DLOGTIME}, \mathbf{SPACE}(\log t(n)))$-uniform family $\mathbf{\Pi}_M = (\Pi_M(i))_{i\in\mathbb{N}}$ of recognizer P systems with the following properties:*

- *the members of $\mathbf{\Pi}_M$ are polarizationless P systems with active membranes, without using membrane division rules, and*
- *for every $n \in \mathbb{N}$, $\Pi_M(n)$ works in $O(t(n))$ time and in $O(t^2(n))$ space.*

The rest of this section is devoted to the proof of this theorem. Let us consider a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ working in $t(n)$ time. We construct a uniform family of recognizer P systems $\mathbf{\Pi}_M = (\Pi_M(i))_{i\in\mathbb{N}}$ that decides the language $M(L)$. Assume that $Q = \{s_1, \ldots, s_m\}$, for some $m \geq 3$, where $s_1 = q_0$, $s_{m-1} = q_a$ and $s_m = q_r$. Moreover, $\Gamma = \{X_1, \ldots, X_k\}$ for some $k > |\Sigma|$, where $X_k = \sqcup$ is the blank symbol of the working tape.

Before giving the precise construction of $\Pi_M(n)$, we describe informally some of its components. As the number of certain components of $\Pi_M(n)$ will depend on n, when the family machine that constructs $\Pi_M(n)$ enumerates these components, an efficient representation of numbers depending on n should be used. Thus, instead of using a number to denote a component of $\Pi_M(n)$, we will use the binary form of this number.

As M stops in at most $t(n)$ steps, the segment of the tape of M that is used during its work consists of at most $t(n)$ cells. This segment of the tape will be represented by the nested membrane structure appearing on Fig. 1. Here the first membrane in the skin represents the first cell of the tape, while the innermost membrane represents the $t(n)$th one. We will call these membranes of $\Pi_M(n)$ *tape-membranes*. Let us consider a tape-membrane representing the lth cell of the tape. We call this membrane the lth *tape-membrane*. Notice that the lth tape-membrane has label $b(l)$, if $l \leq n$, and it has label $b(n+1)$ otherwise (we distinguish the indexes of the first $n+1$ tape-membranes in order to ensure that the objects in the input multiset are able to find their corresponding tape-membranes).

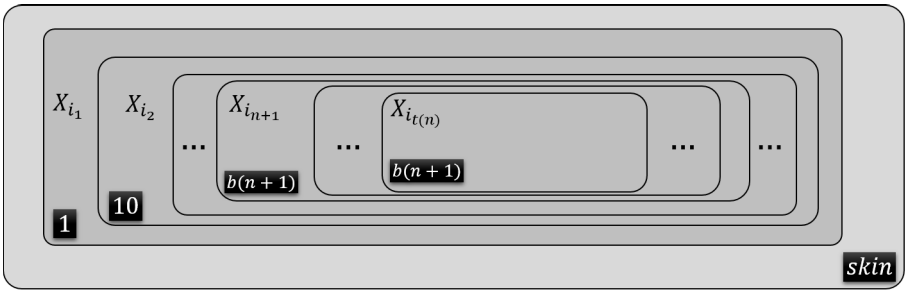

Fig. 1. The membrane structure corresponding to the simulated tape

For every $l \in [t(n)]$, the lth tape-membrane contains further membranes: for every state s_i ($i \in [m-2]$), it contains $t(n)$ copies of a membrane with label s_i. Such a membrane contains a further elementary membrane with label s'_i. Moreover, the lth tape-membrane contains a symbol $X_j \in \Gamma$ if and only if the lth cell of M contains the symbol X_j. Furthermore, if M is in state s_i and the head of M points to the lth cell, then an object $\uparrow_i$ is placed into the lth tape-membrane to represent this information (see Fig. 2 where we assumed that the head of M points to the third cell).

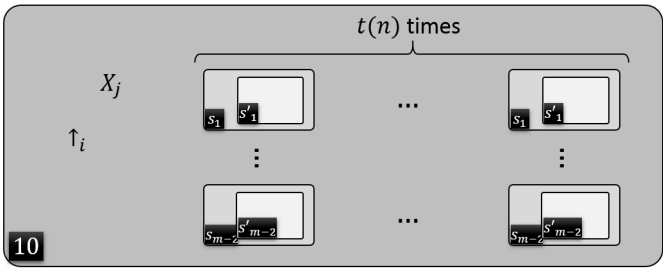

Fig. 2. The membrane structure of the third tape-membrane

We will see that when $\uparrow_i$ and X_j appear in the lth tape-membrane, then these objects will dissolve a membrane pair $[[\ ]_{s'_i}]_{s_i}$ and introduce new objects corresponding to the value $\delta(s_j, X_i)$. With these new objects $\Pi_M(n)$ will be able to maintain its configurations so that finally its current configuration corresponds to the new configuration of M.

The formal definition of $\Pi_M(n)$ is as follows. For every $n \in \mathbb{N}$, let $\Pi_M(n) := (\Sigma', \Gamma', H, \mu, W, R)$, where:

- $\Sigma' := \{a_{b(i)} \mid a \in \Sigma, 1 \leq i \leq n\}$
- $\Gamma' := \Sigma' \cup \Gamma \cup \{\uparrow_i, \downarrow_{i,d} \mid 1 \leq i \leq m, d \in \{L, R\}\} \cup \{d_{b(0)}, \ldots, d_{b(2n)}\} \cup \{yes, no\}$
- $H := \{skin, b(1), \ldots, b(n+1)\} \cup \{s_1, \ldots, s_m, s'_1, \ldots, s'_m\}$;
- μ is a nested membrane structure $[[[\ldots[\ldots[\ ]_{b(n+1)}\cdots]_{b(n+1)}\cdots]_{b(2)}]_{b(1)}]_{skin}$ (containing $t(n)-n$ membranes with label $b(n+1)$), such that each membrane in this structure contains a further membrane structure ν, where ν consists of $t(n)$ copies of the membrane structure $[[\ ]_{s'_i}]_{s_i}$, for every $i \in [m-2]$. The input membrane is $[\ ]_{skin}$;
- $W := w_{skin}, w_{b(1)}, \ldots, w_{b(n+1)}, w_{s_1}, \ldots, w_{s_m}, w_{s'_1}, \ldots, w_{s'_m}$, where
  - $w_{skin} := \varepsilon$;
  - $w_{b(1)} := d_0$, $w_{b(l)} := \varepsilon$, for every $l \in [2, n]$, and $w_{b(n+1)} := X_k$ (i.e., $w_{b(n+1)}$ is the blank symbol);
  - $w_{s_i} = w_{s'_i} := \varepsilon$, for every $i \in [m-2]$;
- R is the set of the following rules:
  - Rules to set up the initial configuration of M:

(a) $a_{b(i)}[\ ]_{b(l)} \to [a_{b(i)}]_{b(l)}$, $a_{b(i)}[\ ]_{b(i)} \to [a]_{b(i)}$, for every $i \in [n]$ and $l < i$;
(b) $[d_{b(i)} \to d_{b(i+1)}]_1$, $[d_{b(2n)} \to \uparrow_1]_1$, for every $i \in [2n-1]$;

- Rules for simulating a configuration step of M:
  (c) $\uparrow_i [\ ]_{s_i} \to [\uparrow_i]_{s_i}$, $[\uparrow_i]_{s_i} \to \varepsilon$, for every $i \in [m-2]$;
  (d) $X_j[\ ]_{s'_i} \to [X_j]_{s'_i}$, $[X_j]_{s'_i} \to X_r \downarrow_{t,d}$, for every $i \in [m-2]$, $j \in [k]$ and $(X_r, s_t, d) = \delta(s_i, X_j)$;
  (e) $\downarrow_{i,R} [\ ]_{b(l)} \to [\uparrow_i]_{b(l)}$, for every $i \in [m-2]$, $l \in [n+1]$;
  (f) $[\downarrow_{i,L}]_{b(l)} \to [\ ]_{b(l)} \uparrow_i$, $[\downarrow_{i,L}]_1 \to [\uparrow_i]_1$, for every $i \in [m-2]$, $l \in [2, n+1]$;
- Rules for sending out the computed answer to the environment:
  (g) $[\downarrow_{m-1,d} \to yes]_{b(l)}$, $[\downarrow_{m,d} \to no]_{b(l)}$, for every $l \in [n+1]$ and $d \in \{L, R\}$;
  (h) $[yes]_{b(l)} \to [\ ]_{b(l)} yes$, $[no]_{b(l)} \to [\ ]_{b(l)} no$, for every $l \in [n+1]$;
  (i) $[yes]_{skin} \to [\ ]_{skin} yes$, $[no]_{skin} \to [\ ]_{skin} no$.

Next, we describe how $\Pi_M(n)$ simulates the work of M. We will see that $\Pi_M(n)$ can set up the initial configuration of M in $O(n)$ steps and that every configuration step of M can be simulated by the P system performing a constant number of steps. We distinguish the following three main stages of the simulation:

Stage 1: Setting up the initial configuration of M. Assume that M is provided with the input word $a_1 a_2 \dots a_n$ ($a_i \in \Sigma, i \in [n]$). Then the input multiset of $\Pi_M(n)$ is $pos(w)$. During the first $2n$ steps, every object $a_{b(i)}$ in the input multiset finds its corresponding membrane with label $b(i)$. At the last step, $a_{b(i)}$ evolves to a, as the sub-index $b(i)$ is not needed any more. Meanwhile, in membrane 1 object d_0 evolves to object $d_{b(2n)}$ and $d_{b(2n)}$ evolves to $\uparrow_1$. After these steps, the lth tape-membrane of the system contains an object $X \in \Gamma$ if and only if the lth cell of the tape of M contains X. Moreover, the object $\uparrow_1$ occurring in the first tape-membrane represents that M's current state is s_1 (that is, the initial state) and that the head of M points to the first cell. Thus, after $2n$ steps the configuration of $\Pi_M(n)$ corresponds to the initial configuration of M.

Stage 2: Simulating a configuration step of M. Assume that M has the configuration appearing in Fig. 3 and that $\Pi_M(n)$ has the corresponding configuration appearing in Fig. 4 (for the sake of simplicity we assume that $l \in [n+1]$; the case when $l > l+1$ can be treated similarly). The simulation of the computation step of M starts as follows. Firstly, $\uparrow_i$ goes into a membrane with label s_i and then dissolves it using rules in (*c*). Meanwhile, $\uparrow_i$ evolves to ε. Let us remark that the system can always find a membrane with label s_i in the corresponding tape-membrane. Indeed, at the beginning of the computation, every tape-membrane contains $t(n)$ copies of a membrane with label s_i. Moreover, M can perform at most $t(n)$ steps and the simulation of one step dissolves exactly one membrane with label s_i.

Next, X_j goes into the membrane s'_i and then dissolves it. During the dissolution two new objects, X_r and $\downarrow_{t,d}$ are introduced according to the value

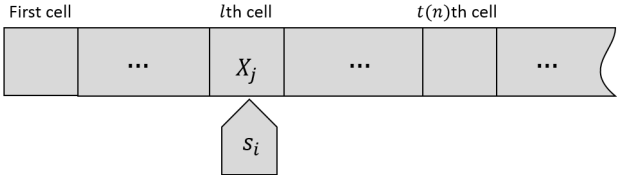

Fig. 3. A configuration of M

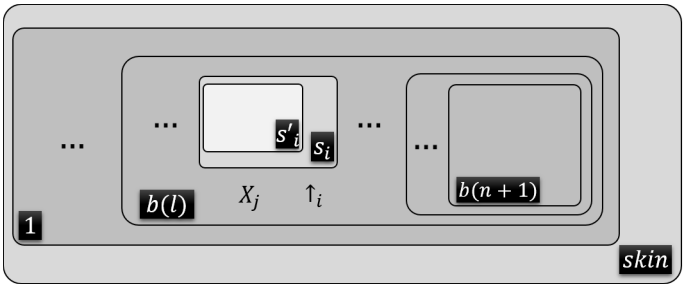

Fig. 4. The corresponding configuration of $\Pi_M(n)$

$\delta(s_i, X_j)$. Notice that in $\downarrow_{t,d}$, the index t corresponds to the index of the new state of M and d denotes the direction of the tape head. Now the simulation of the corresponding movement of the head is done as follows. According to the value of d we distinguish the following cases:

Case 1: $d = R$. In this case $\Pi_M(n)$ applies rules in (e): $\downarrow_{t,R}$ is sent into the next inner tape-membrane and, meanwhile, it evolves to $\uparrow_t$. This corresponds to the move of the tape head to the right.

Case 2: $d = L$. This case is similar to the previous one, but here $\Pi_M(n)$ applies rules in (f): $\downarrow_{t,L}$ is sent out of the current tape-membrane and it evolves to $\uparrow_t$. This corresponds to the move of the tape head to the left. Notice that if $l = 1$, then $\Pi_M(n)$ can apply only the second rule in (f) which means that in this case $\uparrow_t$ remains in the first tape-membrane. This still corresponds to the step of M, since in this case the head of M cannot move left.

Stage 3: Sending the correct answer to the environment. Whenever an object $\downarrow_{m-1,d}$ ($d \in \{L, R\}$) is introduced in a tape-membrane (i.e., when M enters its accepting state), the system introduces object *yes* using the first rule in (g). Then this object is sent out of the tape-membranes until it reaches the skin membrane using rules in (h). Finally, *yes* is sent out to the environment using the first rule in (i). $\Pi_M(n)$ performs a similar computation concerning object *no*.

It can be seen using the notes above that $\Pi_M(n)$ is a confluent polarizationless recognizer P system that simulates M correctly. It is also clear that $\Pi_M(n)$ does not employ membrane division rules. The other properties of $\Pi_M(n)$ mentioned in Theorem 1 are discussed next.

Time and Space Complexity of $\Pi_M(n)$. The time complexity of $\Pi_M(n)$ is measured as follows. As we already discussed, *Stage 1* takes $O(n)$ steps. It can be seen that the simulation of a step of M takes five steps. Thus, *Stage 2* takes $O(t(n))$ steps. Finally, *Stage 3* takes also $O(t(n))$ steps. Thus, for every word $x \in \Sigma^*$ with length n, starting $\Pi_M(n)$ with $pos(x)$ in its input membrane, it halts in $O(t(n))$ steps.

Concerning the space complexity, a configuration of $\Pi_M(n)$ contains $t(n)$ tape-membranes and every tape membrane contains $t(n)$ copies of the membrane structure $[[\ ]_{s'_i}]_{s_i}$, for every $i \in [m-2]$. Moreover, every cell of $\Pi_M(n)$ contains a constant number of objects. Thus, the space complexity of $\Pi_M(n)$ is $O(t^2(n))$.

(**DLOGTIME**, **SPACE**$(\log t(n))$)**-uniformity.** We have already discussed that our input encoding function is in **DLOGTIME**. Thus, it remains to describe a deterministic Turing machine F that can construct $\Pi_M(n)$ using $O(\log t(n))$ space. It is clear that the objects in Γ' and the rules of $\Pi_M(n)$ can be enumerated by F using $O(\log n)$ cells. Indeed, Γ' contains $O(mn)$ objects, but m here is a constant that depends only on M. Moreover, $\Pi_M(n)$ has $O(n)$ different rules.

Furthermore, as $\log t(n)$ is space constructible, F can construct $\log t(n)$ in unary form using $O(\log t(n))$ space. Using the unary representation of $\log t(n)$, the initial membrane structure of $\Pi_M(n)$ can be constructed by F as follows: when F constructs the lth tape-membrane, then it stores $b(l)$ on one of its tapes using at most $\log t(n)$ cells. Furthermore, when F constructs the kth membrane structure of the form $[[\ ]_{s'_i}]_{s_i}$ $(k \in [t(n)], i \in [m-2])$ in the lth tape-membrane, then it stores the words $b(k)$ and $b(i)$ on one of its tapes. This also needs $O(\log t(n))$ cells. Thus, the total number of cells used on the work tapes of F when it constructs $\Pi_M(n)$ is $O(\log t(n))$.

4 Conclusions

The simulation of the behaviour of a device of a computation model in a different model allows to see all problems from a new point of view. One of the frontiers of the current research in Membrane Computing corresponds to the computational power of P systems according to the power of the function that encodes the input and the function that constructs the family of P systems.

In this paper, we prove a general result in this line, since we show that every single-tape deterministic Turing machine working in $t(n)$ time can be simulated by a uniform family of recognizer polarizationless P systems with active membranes. Moreover, this is done without significant slowdown in the working time. Furthermore, if $\log t(n)$ is space constructible, then the members of the family

can be constructed by a family machine that uses $O(\log t(n))$ space. As a particular case, this means that if $t(n)$ is a polynomial function, then the used family of P systems is $(\mathbf{DLOGTIME}, \mathbf{L})$-uniform. Likewise, if $t(n)$ is an exponential function, then the used family is $(\mathbf{DLOGTIME}, \mathbf{PSPACE})$-uniform.

As it is pointed out in [2], uniform families of polarizationless P systems with active membranes and without dissolution rules are at most as powerful as the used input encoding function (see Theorem 10 in [2]). This fact, together with the result of this paper, illustrates the importance of dissolution rules in P systems with active membranes when the polarizations of the membranes are not allowed.

It remains as an open question if $(\mathbf{DLOGTIME}, \mathbf{SPACE}(\log t(n)))$-uniformity in Theorem 1 can be strengthened to $(\mathbf{DLOGTIME}, \mathbf{L})$-uniformity (i.e., whether the construction of $\mathbf{\Pi}_M$ can be done using logarithmic space whatever the running time of M is). In our construction membrane division rules are not employed. Nevertheless, even if we used these rules, it is not clear how the tape-membranes or the membrane structures occurring in them could be constructed using logarithmic space. This might be a subject of further research.

Acknowledgements. Miguel A. Gutiérrez–Naranjo acknowledges the support of the project TIN2012-37434 of the Ministerio de Economía y Competitividad of Spain.

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 Compuer Science **529**, 69–81 (2014)
2. Murphy, N., Woods, D.: The computational complexity of uniformity and semi-uniformity in membrane systems. In: Martínez-del-Amor, M.A., Orejuela-Pinedo, E.F., Păun, Gh., Pérez-Hurtado, I., Riscos-Núñez, A. (eds.) Seventh Brainstorming Week on Membrane Computing, vol. II, pp. 73–84. Fénix Editora, Sevilla (2009)
3. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity - membrane division, membrane creation. In: Păun et al. [10], pp. 302–336
4. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. In: Csuhaj-Varjú, E., Kintala, C., Wotschke, D., Vaszil, Gy. (eds.) Proceeding of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003, pp. 284–294 (2003)
5. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. Journal of Automata, Languages and Combinatorics **11**(4), 423–434 (2006)
6. Porreca, A., Leporati, A., Mauri, G., Zandron, C.: Introducing a space complexity measure for P systems. International Journal of Computers, Communications and Control **4**(3), 301–310 (2009)
7. Porreca, A.E.: Computational Complexity Classes for Membrane System. Master's thesis, Universitá di Milano-Bicocca, Italy (2008)
8. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, Gy. (eds.) CMC 2012. LNCS, vol. 7762, pp. 342–357. Springer, Heidelberg (2013)

9. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics **6**(1), 75–90 (2001)
10. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
11. Romero Jiménez, A., Pérez-Jiménez, M.J.: Simulating Turing machines by P systems with external output. Fundamenta Informaticae **49**(1–3), 273–278 (2002)
12. Sipser, M.: Introduction to the Theory of Computation. Cengage Learning (2012)
13. Valsecchi, A., Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: An efficient simulation of polynomial-space Turing machines by P systems with active membranes. In: Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 461–478. Springer, Heidelberg (2010)