Sublinear-Space P Systems with Active Membranes

Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, and Claudio Zandron

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

Abstract. We introduce a weak uniformity condition for families of P systems, **DLOGTIME** uniformity, inspired by Boolean circuit complexity. We then prove that **DLOGTIME**-uniform families of P systems with active membranes working in logarithmic space (not counting their input) can simulate logarithmic-space deterministic Turing machines.

1 Introduction

Research on the space complexity of P systems with active membranes [4] has shown that these devices, when working in polynomial and exponential space, have the same computing power of Turing machines subject to the same restrictions [7, 1]. In this paper we investigate the behaviour of P systems working in sublinear space.

This requires us, first of all, to define a meaningful notion of sublinear space in the framework of P systems, inspired by sublinear space Turing machines, where the size of the input is not counted as work space.

Since sublinear-space Turing machines are weaker (possibly strictly weaker) than those working in polynomial time, we also define a uniformity condition for the families of P systems that is weaker than the usual **P** uniformity, i.e., **DLOGTIME** uniformity, as usually employed for families of Boolean circuits [2].

Using these restrictions, we show that logarithmic-space P systems with active membranes are able to simulate logarithmic-space deterministic Turing machines, and thus to solve all problems in **L**.

2 Definitions

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

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

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

Each membrane possesses, besides its label and position in μ, another attribute called *electrical charge* (or polarization), 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 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). 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 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. 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 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. 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 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. 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 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 charges β and γ; the object a is replaced, respectively, by b and c while the other objects in the initial multiset are copied to both membranes. If $b \in \Delta$ (resp., $c \in \Delta$) then $a = b$ and $c \notin \Delta$ (resp., $a = c$ and $b \notin \Delta$) must hold.

Each instantaneous configuration of a P system with active membranes is described by the current membrane structure, including the electrical charges, together with the multisets located in the corresponding regions. A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules (inside each membrane several evolution rules can be applied simultaneously).
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or elementary division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it). The same principle applies to each membrane that can be involved to communication, dissolution, or elementary division rules. In other words, the only objects and membranes that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.
- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached after a computation step.
- In each computation step, all the chosen rules are applied simultaneously (in an atomic way). However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of microsteps 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 $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$, 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 we will consider in this paper are confluent.

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

Definition 2. *Let E and F be classes of functions. A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* (E, F)-uniform *if and only if*

- *There exists a function $f \in F$ such that $f(1^n) = \Pi_n$, i.e., mapping the unary representation of each natural number to an encoding of the P system processing all inputs of length n.*
- *There exists a function $e \in E$ mapping each string $x \in \Sigma^\star$ to a multiset $e(x) = w_x$ (represented as a string) over the input alphabet of Π_n, where $n = |x|$.*
- *For each $x \in \Sigma^\star$ we have $\Pi_x = \Pi_n(w_x)$, i.e., Π_x is Π_n with the multiset encoding x placed inside the input membrane.*

Generally, the above mentioned classes of functions E and F are complexity classes; in the most common uniformity condition E and F denote polynomial-time computable functions.

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [3] for further details on the encoding of P systems.

Finally, we describe how space complexity for families of recogniser P systems is measured, and the related complexity classes. The following definition differs from the standard one [6] in one aspect: the input objects do not contribute to the size of the configuration of a P system. This way, only the actual working space of the P system is measured, and P systems working in sublinear space may be analysed. To the best knowledge of the authors, no previously published space complexity result is invalidated by assuming that the input multiset is not counted (the two space measures differ only by a polynomial amount).

Definition 3. *Let $\mathcal{C}$ be a configuration of a P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the sum of the number of membranes in the current membrane structure and* the total number of objects in Γ (i.e., the non-input 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*

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

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 requiring unbounded space.*

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 (E, F)-$\mathbf{MCSPACE}_{\mathcal{D}}(f(n))$ we denote the class of languages which can be decided by (E, F)-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 problems solvable in (E, F)-logarithmic space is denoted by (E, F)-$\mathbf{LMCSPACE}_{\mathcal{D}}$.

3 DLOGTIME-Uniform Families of P Systems

When using uniformity conditions for a family of devices, one should ensure that the chosen uniformity condition is less powerful than the devices themselves if the results deriving from the existence of such family are to be meaningful. For instance, polynomial-time uniformity [5] is acceptable when the resulting family of P systems is able to solve **NP** or **PSPACE**-complete problems (which are conjectured to be outside **P**) in polynomial time. Indeed, in this case the constructed P systems are stronger than the Turing machine constructing them (assuming $\mathbf{P} \neq \mathbf{NP}$ or $\mathbf{P} \neq \mathbf{PSPACE}$, respectively). On the other hand, a polynomial-time uniformity condition is not appropriate when solving a problem in **P**, as the entire computation can be carried out during the construction of the family (by encoding the input instance as a `yes` or as a `no` object, which can be done in polynomial time by hypothesis), and the P systems themselves can accept or reject immediately by sending out the aforementioned object during their first computation step.

Choosing an appropriate uniformity condition is thus very important when the family of devices is, in some sense, "weak". The question has already been investigated in the setting of membrane computing by Murphy and Woods [3], where $\mathbf{AC}^0$ circuits (or, equivalently, a variant of constant-time concurrent random access machines) are used. Here we propose deterministic log-time Turing machines (the usual uniformity condition for $\mathbf{AC}^0$ circuits) themselves as a uniformity condition for P systems. In a later section we shall argue that this particularly weak construction is probably sufficient to replicate most solutions in the literature without requiring major changes.

Definition 4 (Mix Barrington, Immerman [2]). *A* deterministic log-time (**DLOGTIME**) Turing machine *is a Turing machine having a read-only input tape of length n, a constant number of read-write work tapes of length $O(\log n)$, and a read-write address tape, also of length $O(\log n)$. The input tape is not accessed by using a sequential tape head (as the other tapes are); instead, during each step the machine has access to the i-th symbol on the input tape, where i is the number written in binary on the address tape (if $i \geq |n|$ the machine reads an appropriate end-of-input symbol, such as a blank symbol). The machine is required to operate in time $O(\log n)$.*

Notice how only $O(\log n)$ bits of information of the input may be read during a **DLOGTIME** computation. These machines are able to compute the length of their input, compute sums, differences and logarithms of numbers of $O(\log n)$ bits, decode simple pairing functions on strings of length $O(\log n)$ and extract portions of the input of size $O(\log n)$ [2]. Due to their time restrictions, **DLOGTIME** machines are not used to compute the whole representation of a circuit, but rather to describe the "local" connections between the gates (by deciding the immediate predecessors and the type of a single gate [8]).

As P systems are more complicated devices than Boolean circuits, we define a *series* of predicates describing the various features. These predicates will define a function $1^n \mapsto \Pi_n$ for $n \in \mathbb{N}$.

Let $\Pi_n = (\Gamma, \Delta, \Lambda, \mu, w_1, \ldots, w_d, R)$.[1]

Alphabet. The predicate $\textsc{alphabet}(1^n, m)$ holds for a single integer m such that $\Gamma \cup \Delta \subseteq \{0,1\}^m$, i.e., each symbol of the alphabets of Π_n (whose index is provided in unary notation) can be represented as a binary number of m bits. Here m is not necessarily the *minimum* number of bits needed; we can choose a larger number of bits for simplicity, but the number must be $O(\log n)$ as the alphabet is at most polynomial in size with respect to n.

Labels. Analogously, the predicate $\textsc{labels}(1^n, m)$ is true for a single integer m such that $\Lambda \subseteq \{0,1\}^m$, with the same restrictions as the ALPHABET predicate.

Membrane structure. The predicate $\textsc{outermost}(1^n, h)$ holds iff the membrane labelled by h is the outermost membrane of the P system Π_n. The predicate $\textsc{inside}(1^n, h_1, h_2)$ holds iff the membrane labelled by h_1 is immediately contained in h_2 in the initial configuration of Π_n. The resulting graph $\mu = (V, E)$, where

$$V = \{h : \textsc{outermost}(1^n, h)\} \cup \{h_1 : \textsc{inside}(1^n, h_1, h_2)\}$$
$$E = \big\{\{h_1, h_2\} : \textsc{inside}(1^n, h_1, h_2)\big\},$$

must be a tree, where the root is identified by the predicate OUTERMOST. Furthermore, μ must be polynomial in size with respect to n. Here the labels h, h_1, h_2 are provided as strings of bits of appropriate length, as described above.

The predicate $\textsc{input}(1^n, h)$ holds iff the input membrane of Π_n is h.

[1] We use this simplified form for the P system instead of the more formally correct $\Pi_n = (\Gamma_n, \Delta_n, \Lambda_n, \mu_n, w_1, \ldots, w_{d_n}, R_n)$ in order to ease the notation.

Initial multisets. For each multiset in the initial configuration of Π_n choose a fixed string $w \in \Gamma^\star$ representing it. The predicate $\textsc{multiset}(1^n, h, i, a)$ holds iff the i-th symbol of the string representing the multiset contained in membrane h is a, where the symbol a is provided as a string of bits as described above. The predicate is always false for $i \geq |w|$. The length of w must be at most polynomial with respect to n.

Evolution rules. The predicate $\#\textsc{evolution}(1^n, h, \alpha, a, m)$ holds iff Π_n has m object evolution rules of the form $[a \to w]^\alpha_h$, where m is polynomial in n.

The right-hand side of each rule can be recovered by evaluating the predicate $\textsc{evolution}(1^n, h, \alpha, a, i, j, b)$, which is true when the i-th rule of the form $[a \to w]^\alpha_h$ (under any chosen, fixed total order of the rules) has $w_j = b$ (and is false for $j \geq |w|$). Once again, $|w|$ must be polynomial in n.

Other kinds of rules. The following predicates describe the communication, dissolution and elementary division rules of Π_n:

$$
\begin{aligned}
&\textsc{send-in}(1^n, h, \alpha, a, \beta, b) && \iff && a\,[\,]^\alpha_h \to [b]^\beta_h \in R;\\
&\textsc{send-out}(1^n, h, \alpha, a, \beta, b) && \iff && [a]^\alpha_h \to [\,]^\beta_h\, b \in R;\\
&\textsc{dissolve}(1^n, h, \alpha, a, b) && \iff && [a]^\alpha_h \to b \in R;\\
&\textsc{elem-divide}(1^n, h, \alpha, a, \beta, b, \gamma, c) && \iff && [a]^\alpha_h \to [b]^\beta_h\, [c]^\gamma_h \in R\\
& && && \quad \text{and } h \text{ is elementary.}
\end{aligned}
$$

These predicates completely describe a mapping $1^n \mapsto \Pi_n$ for every $n \in \mathbb{N}$.

Definition 5. *The mapping $1^n \mapsto \Pi_n$ is said to be* **DLOGTIME***-computable if all the predicates* LABELS*,* ALPHABET*,* OUTERMOST*,* INSIDE*,* INPUT*,* MULTISET*,* #EVOLUTION*,* EVOLUTION*,* SEND-IN*,* SEND-OUT*,* DISSOLVE*, and* ELEM-DIVIDE *are* **DLOGTIME***-computable.*

Each P system Π_n will be used to process all inputs $x \in \Sigma^n$, once they have been suitably encoded as a multiset w_x over the input alphabet of Π_n.

Input multiset. The predicate $\textsc{encoding}(x, i, a)$ holds when the i-th object of the input multiset encoding x is a (the predicate is false if there is no i-th object). The multiset size must be polynomial with respect to $n = |x|$.

Definition 6. *The mapping $x \mapsto w_x$ is said to be* **DLOGTIME***-computable iff the predicate* ENCODING *is* **DLOGTIME***-computable.*

We are now finally able to define $(\mathbf{DLOGTIME}, \mathbf{DLOGTIME})$-uniform (or $(\mathbf{DLT}, \mathbf{DLT})$-uniform for brevity) families of P systems according to Definition 2.

4 Simulating Logspace Turing Machines

In this section we prove that logarithmic-space Turing machines can be simulated by logarithmic-space families of P systems with active membranes even if we use a $(\mathbf{DLT}, \mathbf{DLT})$ uniformity condition.

Theorem 1. *Let M be a deterministic Turing machine with an input tape (of length n) and a work tape of length $O(\log n)$. Then, there exists a $(\mathbf{DLT}, \mathbf{DLT})$-uniform family $\boldsymbol{\Pi}$ of confluent recogniser P systems with active membranes working in logarithmic space such that $L(M) = L(\boldsymbol{\Pi})$.*

Proof. Let $s(n) = k \log n$ be an upper bound on the length of the work tape of the Turing machine M, let Σ be the alphabet of M (including the blank symbol $\sqcup$) and Q its set of non-final states. Also, for all $n \in \mathbb{N}$, let $\ell(n)$ be the minimum number of bits required in order to represent the integers $\{0, \ldots, n-1\}$, that is, $\ell(n) = \lfloor \log(n-1) \rfloor + 1$.

The initial configuration of Π_n, the P system simulating M on inputs of length n, consists of:

- An outermost membrane labelled by $\mathtt{h}$. This membrane contains the object $\mathtt{q}_{0,0}$, whose subscripts are written using $\ell(n)$ and $\ell(s(n))$ bits respectively. This is called the *state object*. In general, the existence of the object $q_{i,w}$ for some $q \in Q$ and $i, w \in \mathbb{N}$ indicates that the simulated Turing machine M is currently in state q and its tape heads are located on the i-th symbol on the input tape and on the w-th symbol of the work tape.
- $\ell(n)$ nested membranes labelled by $\mathtt{i}_0, \ldots, \mathtt{i}_{\ell(n)-1}$ (where the subscripts are all represented in binary with exactly $\ell(\ell(n))$ bits), called the *input tape membranes*. The innermost membrane $\mathtt{i}_0$ is the input membrane of Π_n.
- $s(n)$ membranes placed inside $\mathtt{h}$ and labelled by $\mathtt{w}_0, \ldots, \mathtt{w}_{s(n)-1}$ (using $\ell(s(n))$ bits for the subscripts), called the *work tape membranes*. Each membrane $\mathtt{w}_w$ initially contains the object $\sqcup$, indicating that the w-th cell of the work tape of M is blank.
- Two sets of membranes $\{a_{\mathtt{i}} : a \in \Sigma\}$ and $\{a_{\mathtt{w}} : a \in \Sigma\}$, placed inside $\mathtt{h}$ and respectively called *input tape symbol membranes* and *work tape symbol membranes*.

The input $x \in \Sigma^\star$ of Π_n is encoded as a multiset by subscripting each symbol with its position inside x, counting from 0 and using $\ell(n)$ bits. This multiset is then placed inside membrane $\mathtt{i}_0$. (See Fig. 1.)

Now assume that a few steps of M have been simulated by Π_x. The current configuration of the P system will be similar to the initial one, except that the initial state object $\mathtt{q}_{0,0}$ is replaced by some $q_{i,w}$ (with $q \in Q$, $0 \leq i < n$, $0 \leq w < s(n)$) and the membranes $\mathtt{w}_0, \ldots, \mathtt{w}_{s(n)-1}$ contain objects corresponding to the symbols on the work tape of M. (See Fig. 2.)

The state object now enters the membranes $\mathtt{i}_{\ell(n)-1}, \ldots, \mathtt{i}_0$ in that order; at the same time, it sets the charge of membrane $\mathtt{i}_j$ to negative, if the j-th least significant bit (counting from 0) of its subscript i is $\mathtt{0}$, and to positive if that bit is $\mathtt{1}$. The following rules are used in order to perform this process:

$$q_{i,w} \; [\,]^0_{\mathtt{i}_j} \to [q_{i,w}]^-_{\mathtt{i}_j} \qquad \text{if the } j\text{-th least significant bit of } i \text{ is } \mathtt{0} \tag{1}$$

$$q_{i,w} \; [\,]^0_{\mathtt{i}_j} \to [q_{i,w}]^+_{\mathtt{i}_j} \qquad \text{if the } j\text{-th least significant bit of } i \text{ is } \mathtt{1}. \tag{2}$$

These rules are replicated for all $q \in Q$, $0 \leq i < n$, $0 \leq w < s(n)$, $0 < j < \ell(n)$.

For the innermost membrane $\mathtt{i}_0$ instead we use the following rules, which add a binary counter of $\ell(n)$ bits (starting from 0) as a superscript to the state object:

$$q_{i,w}\ [\ ]^0_{\mathtt{i}_0} \to [q^0_{i,w}]^-_{\mathtt{i}_0} \qquad \text{if the least significant bit of } i \text{ is } \mathtt{0} \tag{3}$$

$$q_{i,w}\ [\ ]^0_{\mathtt{i}_0} \to [q^0_{i,w}]^+_{\mathtt{i}_0} \qquad \text{if the least significant bit of } i \text{ is } \mathtt{1}. \tag{4}$$

These rules are replicated for all $q \in Q$, $0 \le i < n$, $0 \le w < s(n)$.

When membrane $\mathtt{i}_0$ becomes non-neutral, the input objects a_i (for $0 \le i < n$) are sent out. Membranes $\mathtt{i}_0, \ldots, \mathtt{i}_{\ell(n)-1}$ behave as "filters" in the following sense: object a_i may pass through $\mathtt{i}_j$ only if the charge of the membrane corresponds to the j-th bit of i (where positive denotes a $\mathtt{1}$, and negative a $\mathtt{0}$). Exactly one input object will traverse all of them and reach the outermost membrane, namely, the object corresponding to the symbol under the tape head in the current configuration of M, whose position on the input tape is represented by the subscript i of the object $q_{i,w}$. Indeed, it is never the case that two or more input objects reach the outermost membrane, since the subscripts of the input symbols are unique (i.e., no two input objects a_{i_1}, a_{i_2} have identical bits in all $\ell(n)$ positions of their subscripts); moreover, one of them always does, since the simulated Turing machine, being a legitimate one, has a symbol under its input tape head at all times. The time required for the correct input object to reach the outermost membrane depends on the nondeterministic order in which the objects are sent out from the membranes $\mathtt{i}_0, \ldots, \mathtt{i}_{\ell(n)-1}$; in the following discussion we use a worst-case upper bound of $\frac{n}{2} + \ell(n) + 1$.

Formally, the required rules are:

$$[a_i]^-_{\mathtt{i}_j} \to [\ ]^-_{\mathtt{i}_j}\ a_i \qquad \text{if the } j\text{-th bit of } i \text{ is } \mathtt{0} \tag{5}$$

$$[a_i]^+_{\mathtt{i}_j} \to [\ ]^+_{\mathtt{i}_j}\ a_i \qquad \text{if the } j\text{-th bit of } i \text{ is } \mathtt{1}. \tag{6}$$

These rules are replicated for all $a \in \Sigma, 0 \le i < n$, $0 \le j < \ell(n)$.

The single object that reaches the outermost membrane $\mathtt{h}$ is then used in order to set to positive the charge of the corresponding membrane $a_{\mathtt{i}}$ (thus signalling that the symbol under the input tape head is a):

$$a_i\ [\ ]^0_{a_{\mathtt{i}}} \to [a_i]^0_{a_{\mathtt{i}}} \tag{7}$$

$$[a_i]^0_{a_{\mathtt{i}}} \to [\ ]^+_{a_{\mathtt{i}}}\ a_i \tag{8}$$

These rules are replicated for all $a \in \Sigma, 0 \le i < n$.

The number of steps required for these operations to be carried out (starting from the moment membrane $\mathtt{i}_0$ becomes non-neutral) is bounded by $\frac{n}{2} + \ell(n) + 1$. During this time, the head object waits inside $\mathtt{i}_0$ by using the following rules:

$$[q^t_{i,w} \to q^{t+1}_{i,w}\ ]^\alpha_{\mathtt{i}_0} \qquad \text{for } 0 \le t < \frac{n}{2} + \ell(n) + 1 \tag{9}$$

These rules are replicated for all $q \in Q$, $0 \le i < n$, $0 \le w < s(n)$, $\alpha \in \{+, -\}$. (See Fig. 3.)

When the superscript t reaches $\frac{n}{2}+\ell(n)+1$, the state object travels back to membrane $\mathtt{h}$ while resetting the charges of $\mathtt{i}_0,\ldots,\mathtt{i}_{\ell(n)-1}$ to neutral:

$$[q_{i,w}^{\frac{n}{2}+\ell(n)+1}]_{\mathtt{i}_0}^{\alpha} \to [\,]_{\mathtt{i}_0}^{0}\, q'_{i,w} \tag{10}$$

$$[q'_{i,w}]_{\mathtt{i}_j}^{\alpha} \to [\,]_{\mathtt{i}_j}^{0}\, q'_{i,w} \tag{11}$$

$$[q'_{i,w}]_{\mathtt{i}_{\ell(n)-1}}^{\alpha} \to [\,]_{\mathtt{i}_{\ell(n)-1}}^{0}\, q_{i,w}^{0} \tag{12}$$

These rules are replicated for all $q \in Q$, $0 \le i < n$, $0 \le w < s(n)$, $0 < j < \ell(n)-1$, $\alpha \in \{+,-\}$.

When membranes $\mathtt{i}_j$ revert to neutral, the input objects a_i are sent back in, all the way to the input membrane $\mathtt{h}_0$:

$$a_i\,[\,]_{\mathtt{i}_j}^{0} \to [a_i]_{\mathtt{i}_j}^{0} \tag{13}$$

These rules are replicated for all $0 \le i < n$, $0 \le j < \ell(n)$, $a \in \Sigma$.

Once again, the state object waits $\frac{n}{2}+\ell(n)+1$ steps (this time, inside membrane $\mathtt{h}$) for this process to complete:

$$[q_{i,w}^{t} \to q_{i,w}^{t+1}]_{\mathtt{h}}^{0} \qquad \text{for } 0 \le t < \frac{n}{2}+\ell(n)+1 \tag{14}$$

These rules are replicated for all $q \in Q$, $0 \le i < n$, $0 \le w < s(n)$.

The time required up to now is $O\big(\frac{n}{2}+\ell(n)\big) = O(n)$ steps. The remainder of the simulation of the current step of M will only require a constant number of steps. First, the state object $q_{i,w}^{\frac{n}{2}+\ell(n)+1}$ enters membrane $\mathtt{w}_w$ and changes its charge, thus causing the object a inside it to be sent out.

$$q_{i,w}^{\frac{n}{2}+\ell(n)+1}\,[\,]_{\mathtt{w}_w}^{0} \to [q''_{i,w}]_{\mathtt{w}_w}^{+} \tag{15}$$

$$[a]_{\mathtt{w}_w}^{+} \to [\,]_{\mathtt{w}_w}^{-}\, a \tag{16}$$

These rules are replicated for all $q \in Q$, $0 \le i < n$, $0 \le w < s(n)$, $a \in \Sigma$.

When the charge of $\mathtt{w}_w$ becomes negative, the state object is sent out to $\mathtt{h}$, while object a enters the corresponding membrane $a_{\mathtt{w}}$ and sets its charge to positive.

$$[q''_{i,w}]_{\mathtt{w}_w}^{-} \to [\,]_{\mathtt{w}_w}^{-}\, q''_{i,w} \tag{17}$$

$$a\,[\,]_{a_{\mathtt{w}}}^{0} \to [a]_{a_{\mathtt{w}}}^{+} \tag{18}$$

These rules are replicated for all $q \in Q$, $0 \le i < n$, $0 \le w < s(n)$, $a \in \Sigma$.

Now the configuration of Π_x (see Fig. 4) has the following properties:

- Exactly one membrane among $\mathtt{w}_0,\ldots,\mathtt{w}_{s(n)-1}$ is negatively charged (this is the membrane corresponding to the work tape cell currently scanned by M) while the others are neutral.
- Exactly one membrane $a_{\mathtt{i}}$ is positively charged (the one corresponding to the input tape symbol currently read by M), while $b_{\mathtt{i}}$ is neutral for all $b \in \Sigma - \{a\}$.

– Exactly one membrane $a_{\mathtt{w}}$ is positively charged (the one corresponding to the work tape symbol currently read by M), while $b_{\mathtt{w}}$ is neutral for all $b \in \Sigma - \{a\}$.

While the object a inside membrane $a_{\mathtt{w}}$ is deleted by the following rule, replicated for all $a \in \Sigma$:

$$[a \to \lambda]^{+}_{a_{\mathtt{w}}} \tag{19}$$

the state object can identify the symbols currently read by M by checking the charges of the corresponding membranes (resetting them to neutral), and store those symbols as superscripts:

$$q''_{i,w}\ [\,]^{+}_{a_{\mathtt{i}}} \to [q''_{i,w}]^{+}_{a_{\mathtt{i}}} \tag{20}$$

$$[q''_{i,w}]^{+}_{a_{\mathtt{i}}} \to [\,]^{0}_{a_{\mathtt{i}}}\ q^{a}_{i,w} \tag{21}$$

$$q^{a}_{i,w}\ [\,]^{+}_{b_{\mathtt{w}}} \to [q^{a}_{i,w}]^{+}_{b_{\mathtt{w}}} \tag{22}$$

$$[q^{a}_{i,w}]^{+}_{b_{\mathtt{w}}} \to [\,]^{0}_{b_{\mathtt{w}}}\ q^{a,b}_{i,w} \tag{23}$$

These rules are replicated for all $q \in Q$, $0 \le i < n$, $0 \le w < s(n)$, $a, b \in \Sigma$.

Now the state object possesses all the information needed in order to simulate the transition of M, namely, the state itself and the two symbols currently scanned by the Turing machine. Let

$$\delta \colon Q \times \Sigma^2 \to Q \times \Sigma \times \{+1, -1\}^2$$

be the transition function of M; here we assume δ is only defined for non-final states, and that the head movements are represented by ± 1. Assume that

$$\delta(q, a, b) = (r, c, d_1, d_2).$$

Then, the following rules produce the object representing the new work tape symbol that replaces a:

$$[q^{a,b}_{i,w} \to \hat{q}^{a,b}_{i,w}\ c']^{0}_{\mathtt{h}} \tag{24}$$

These rules are replicated for all $q \in Q$, $0 \le i < n$, $0 \le w < s(n)$, $a, b \in \Sigma$.

The object c' is sent to the membrane simulating the tape cell it is written on, i.e., the only negatively charged membrane $\mathtt{w}_w$, and it resets its charge to neutral (while losing the prime):

$$c'\ [\,]^{-}_{\mathtt{w}_w} \to [c]^{0}_{\mathtt{w}_w} \tag{25}$$

This rule is replicated for all $0 \le w < s(n)$, $c \in \Sigma$.

Simultaneously, the state object has to update three pieces of information (state and positions on the tapes) in order to complete the simulation of the current step of M:

$$[\hat{q}^{a,b}_{i,w} \to r_{i',w'}]^{0}_{\mathtt{h}} \qquad \text{where } i' = i + d_1,\ w' = w + d_2 \tag{26}$$

These rules are replicated for all $q \in Q$, $0 \leq i < n$, $0 \leq w < s(n)$, $a, b \in \Sigma$.

The configuration of Π_x now encodes the configuration of M after having simulated the step performed by the Turing machine in $O(n)$ time. The simulation may now proceed with the next step of M.

If M reaches an accepting state q, then the following rule is applied:

$$[q_{i,w}]^0_{\mathtt{h}} \to [\,]^0_{\mathtt{h}}\, \mathtt{yes} \tag{27}$$

while the following one is applied for a rejecting state:

$$[q_{i,w}]^0_{\mathtt{h}} \to [\,]^0_{\mathtt{h}}\, \mathtt{no} \tag{28}$$

These rules are replicated for all $0 \leq i < n$, $0 \leq w < s(n)$.

This completes the description of the family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ simulating M. Each P system Π_x only requires $O(\log |x|)$ membranes and objects besides the input objects (and these are not modified nor created during the computation). The time required by the simulation is $O\big(n \cdot t(n)\big)$, where $t(n)$ is the maximum number of steps performed by M on inputs of length n.

In order to prove Theorem 1 we still need to show that the family $\boldsymbol{\Pi}$ is indeed $(\mathbf{DLT}, \mathbf{DLT})$-uniform. Here we provide a proof sketch for this result.

Consider the mapping $x \mapsto w_x$, encoding each input string of M as a multiset over the alphabet of Π_n (with $n = |x|$): each symbol of x has to be subscripted with an index of $\ell(n)$ bits representing its position in x. The corresponding ENCODING predicate is

$$\textsc{encoding}(x, i, a_j) \iff j = i \wedge x_i = a.$$

It is easy to check in **DLOGTIME** if the predicate holds for each (x, i, a_j). First, we copy the portions of the input representing i and a_j (of length $O(\log n)$) on auxiliary work tapes and we check if the third argument is indeed of the form a_j for some $a \in \Sigma$ by simulating a finite state automaton. By scanning i and j we can ensure that $i = j$. Then, we extract the i-th symbol of x by copying i on the address tape of the machine, and we check if that symbol is a. Since symbol-by-symbol comparisons require linear time with respect to the length of the strings, the evaluation of ENCODING can be carried out in logarithmic time.

The alphabet of Π_n can be represented by using $O(\ell(n))$ bits, where the hidden constants also depend on the size of the alphabet Σ of M. For simplicity, we can use $k \cdot \ell(n)$ for some appropriate k as an upper bound, and set

$$\textsc{alphabet}(1^n, m) \iff m = k \cdot \ell(n).$$

This predicate can be checked in **DLOGTIME**, as multiplication by a constant can be implemented by repeated additions. The reasoning for the predicate LABELS is similar.

The membrane structure of Π_n (see Fig. 1 for an example with $n = 5$) is described as follows:

$$\begin{aligned}
\textsc{outermost}(1^n, h) &\iff h = \mathtt{h} \\
\textsc{inside}(1^n, h_1, h_2) &\iff (h_1 = \mathtt{i}_{\ell(n)-1} \wedge h_2 = \mathtt{h}) \vee \\
&\quad (h_1 = \mathtt{i}_j \wedge h_2 = \mathtt{i}_{j+1} \wedge 0 \leq j < \ell(n) - 1) \vee \\
&\quad (h_1 = \mathtt{w}_j \wedge h_2 = \mathtt{h} \wedge 0 \leq j < s(n)) \vee \\
&\quad (h_1 = a_{\mathtt{i}} \wedge h_2 = \mathtt{h} \wedge a \in \Sigma) \vee \\
&\quad (h_1 = a_{\mathtt{w}} \wedge h_2 = \mathtt{h} \wedge a \in \Sigma)
\end{aligned}$$

that is, by a disjunction of a constant number of conjuncts, each one consisting of a constant number of terms whose truth can be verified in **DLOGTIME** by executing comparisons or simple computations on numbers of $O(\log n)$ bits. The input membrane is identified by

$$\textsc{input}(1^n, h) \iff h = \mathtt{i}_0.$$

The initial multisets are described by

$$\begin{aligned}
\textsc{multiset}(1^n, h, i, a) &\iff (h = \mathtt{h} \wedge i = 0 \wedge a = \mathtt{q}_{0,0}) \vee \\
&\quad (h = \mathtt{w}_j \wedge i = 0 \wedge a = \sqcup \wedge 0 \leq j < s(n))
\end{aligned}$$

which is also decidable in **DLOGTIME**.

We shall not describe in detail the predicates for the rules of Π_x. As an example, consider the rules of kind (14) on page 10:

$$[q^t_{i,w} \to q^{t+1}_{i,w}]^0_{\mathtt{h}} \qquad\qquad \text{for } 0 \leq t < \frac{n}{2} + \ell(n) + 1$$

It is easy to see that

$$\#\textsc{evolution}(1^n, \mathtt{h}, 0, a, 1)$$

holds for $a = q^t_{i,w}$, $q \in Q$, $0 \leq i < n$, $0 \leq w < s(n)$, $0 \leq t < \frac{n}{2} + \ell(n) + 1$; this is one of the conjuncts of the full definition of $\#\textsc{evolution}$. The value $\frac{n}{2} + \ell(n) + 1$ can be computed from 1^n in **DLOGTIME**. The part of the predicate EVOLUTION dealing with rules of kind (14)

$$\textsc{evolution}(1^n, \mathtt{h}, 0, q^t_{i,w}, 0, j, b)$$

then holds when $j = 0$ and $b = q^{t+1}_{i,w}$, and this can be checked in **DLOGTIME** as described before.

The full definition of EVOLUTION (and of all the other predicates for the rules of Π_n) is a disjunction of a constant number of conjuncts (each one dealing with a different kind of evolution rules, depending on the elements on the left-hand side of the rule) where each conjunct can be checked in **DLOGTIME**. □

An immediate corollary of Theorem 1 is that the class of problems solved by logarithmic-space Turing machines is contained in the class of problems solved by $(\mathbf{DLT}, \mathbf{DLT})$-uniform, logarithmic-space P systems with active membranes.

Corollary 1. $\mathbf{L} \subseteq (\mathbf{DLT}, \mathbf{DLT})\text{-}\mathbf{LMCSPACE}_{\mathcal{AM}}$. □

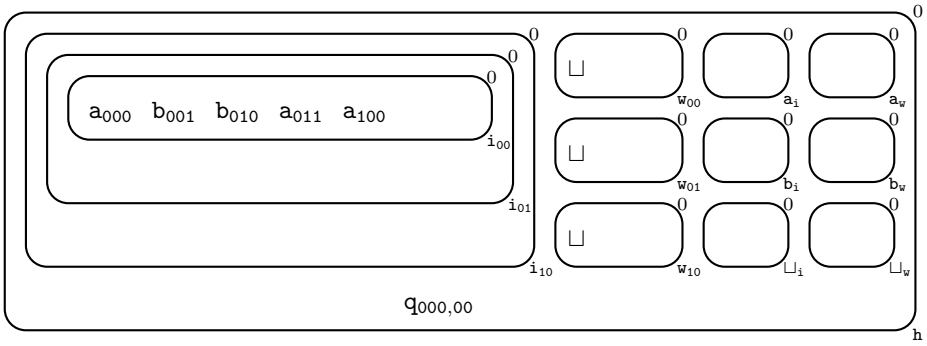

Fig. 1. The initial configuration of Π_x, that is Π_n with $n = 5$ and input $x = \mathtt{abbaa}$, assuming M uses $\log n$ space, has $\Sigma = \{\mathtt{a}, \mathtt{b}, \sqcup\}$ as its alphabet and $\mathtt{q}$ as its initial state.

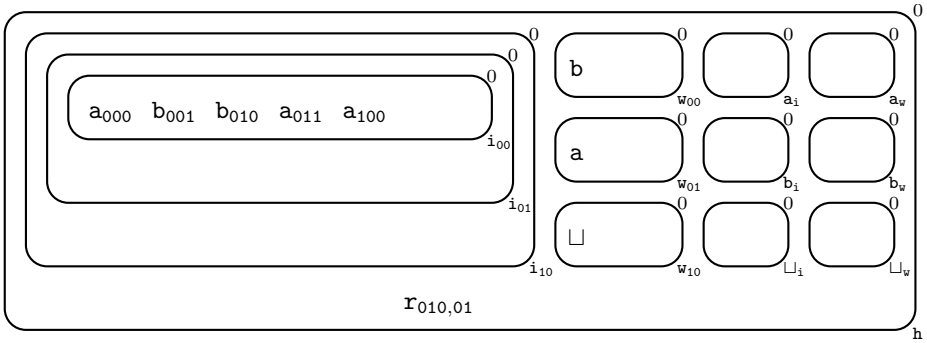

Fig. 2. A possible configuration of the P system Π_x (see Fig. 1) simulating the Turing machine M after a few computation steps have been simulated. Here the current state of M is $\mathtt{r}$, the work tape contains the string $\mathtt{ba}$, the input tape head is on cell 2 (binary 010), and the work tape head is on cell 1 (binary 01).

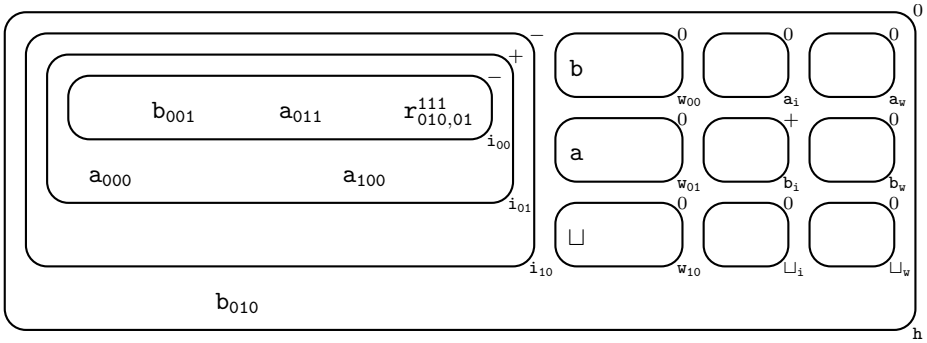

Fig. 3. Configuration of Π_x (from Fig. 1) after the object $\mathtt{b}_{010}$ (corresponding to the symbol under the input tape head) has set the charge of membrane $\mathtt{b}_\mathtt{i}$ to positive, allowing the state-object to identify it.

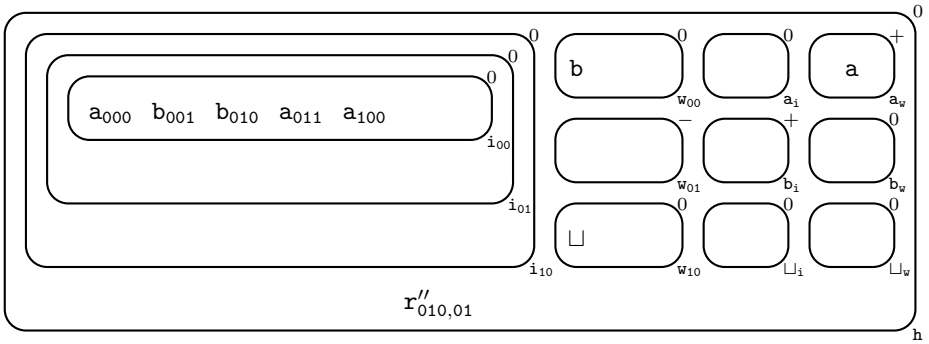

Fig. 4. Configuration of Π_x after the object a has set the charge of membrane $\mathtt{a}_w$ to positive, thus identifying the symbol under the work tape head.

5 Conclusions

In this paper we extended the definition of space complexity for P systems [6] in order to consider sublinear-space computations and compare them to logarithmic-space Turing machines.

To ensure that the P systems themselves perform the actual computation (as opposed to letting the uniformity machine solve the problem), we needed to weaken the usual polynomial-time uniformity condition (as $\mathbf{L} \subseteq \mathbf{P}$). We showed how a variant of a common uniformity condition for Boolean circuits, **DLOGTIME** uniformity, may also be used to define families of P systems with active membranes.

We were then able to define **DLOGTIME**-uniform families of P systems working in logarithmic space and simulating logarithmic-space Turing machines, thus showing that the former devices are at least as computationally powerful as the latter ones, in symbols $\mathbf{L} \subseteq (\mathbf{DLT}, \mathbf{DLT})\text{-}\mathbf{LMCSPACE}_{\mathcal{AM}}$.

Although the **DLOGTIME** uniformity condition we proposed, like the $\mathbf{AC}^0$ uniformity already considered in the literature [3], is weaker than the usual **P** uniformity, it nevertheless seems powerful enough to be applied to many already published results. Indeed, we conjecture that most previously defined **P**-uniform families of P systems can be adapted to **DLOGTIME** uniformity.

It remains to be established whether $(\mathbf{DLT}, \mathbf{DLT})\text{-}\mathbf{LMCSPACE}_{\mathcal{AM}} = \mathbf{L}$, or if that class includes harder problems like, for instance, those in **NL**.

Acknowledgements

The authors would like to thank Artiom Alhazov, Luca Manzoni, Niall Murphy, and Marco S. Nobile for the suggestions they provided. This work was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo per la Ricerca (FAR) 2011.

References

1. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: The computational power of exponential-space P systems with active membranes. In: Martínez-del-Amor, M.A., Păun, Gh., Pérez-Hurtado, I., Romero-Campero, F.J. (eds.) Proceedings of the Tenth Brainstorming Week on Membrane Computing, vol. I, pp. 35–60. Fénix Editora (2012)
2. Mix Barrington, D.A., Immerman, N., Straubing, H.: On uniformity within NC^1. Journal of Computer and System Sciences 41(3), 274–306 (1990)
3. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. Natural Computing 10(1), 613–632 (2011)
4. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
5. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2(3), 265–284 (2003)
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), 301–310 (2009)
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), 65–73 (2011)
8. Ruzzo, W.L.: On uniform circuit complexity. Journal of Computer and System Sciences 22(3), 365–383 (1981)