Space complexity equivalence of P systems with active membranes and Turing machines

Artiom Alhazov[a], Alberto Leporati[b,*], Giancarlo Mauri[b], Antonio E. Porreca[b], Claudio Zandron[b]

[a] *Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, Academiei 5, Chişinău, MD-2028, Moldova*

[b] *Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336/14, 20126 Milano, Italy*

Abstract

We prove that arbitrary single-tape Turing machines can be simulated by uniform families of P systems with active membranes with a cubic slowdown and quadratic space overhead. This result is the culmination of a series of previous partial results, finally establishing the equivalence (up to a polynomial) of many space complexity classes defined in terms of P systems and Turing machines. The equivalence we obtained also allows a number of classic computational complexity theorems, such as Savitch's theorem and the space hierarchy theorem, to be directly translated into statements about membrane systems.

Keywords: P systems with active membranes, Turing machines, computational complexity

1. Introduction

P systems with active membranes have been introduced in [6] 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 in polynomial time [14, 3]. The class **PSPACE** is also an *upper* bound to the complexity of polynomial-time P systems with active membranes [15], even when they employ an exponential workspace.

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 [17, 8]. More recently, 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 [10]. On the other hand, if division of membranes is not allowed, then the efficiency apparently decreases: no **NP**-complete problem can be solved in polynomial time without using division rules unless $\mathbf{P} = \mathbf{NP}$ holds [17].

A measure of space complexity for P systems has also been introduced, in order to analyse the time-space trade-off exploited when P systems are used to efficiently solve computationally hard problems [9]. The space required by a P system is the maximal size it can reach during any legal computation, defined as the sum of the number of membranes and the number of objects. A uniform family $\mathbf{\Pi}$ of P systems is said to solve a problem in space $s\colon \mathbb{N} \to \mathbb{N}$ if no P system in $\mathbf{\Pi}$ associated to an input string of length n requires

*Corresponding author

Email addresses: artiom@math.md (Artiom Alhazov), alberto.leporati@unimib.it (Alberto Leporati), mauri@disco.unimib.it (Giancarlo Mauri), porreca@disco.unimib.it (Antonio E. Porreca), zandron@disco.unimib.it (Claudio Zandron)

more than $s(n)$ space. Under this notion of space complexity, in [11] 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 equivalence has been subsequently extended to exponential space [2], by proving that the class $\mathbf{EXPMCSPACE}_{\mathcal{AM}}$ of problems solvable in exponential space by P systems with active membranes coincides with **EXPSPACE**. Both these results have been obtained by mutual simulation of P systems and Turing machines.

The techniques used in [10, 11, 2] to simulate Turing machines via uniform families of P systems do not seem to apply when the space bound is super-exponential. This is due to the fact that the membranes are identified by binary numbers; when dealing with a super-exponential number of different membrane labels, such numbers would be made of a super-polynomial number of digits, 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 [8].

By using different techniques, in this paper we show that arbitrary single-tape Turing machines can be simulated by uniform families of P systems with a cubic slowdown and a quadratic space overhead; hence, the classes of problems solvable by P systems with active membranes and by Turing machines coincide up to a polynomial with respect to space complexity. This equivalence allows us to adapt to P systems existing theorems about the space complexity of Turing machines, such as Savitch's theorem and the space hierarchy theorem.

The rest of the paper is organised as follows. In Section 2 we recall some definitions concerning P systems with active membranes and their space complexity. In Section 3 we recall how P systems with active membranes can simulate register machines, and how this can be exploited for implementing subroutines. In Section 4 we describe how Turing machine configurations can be encoded as P system configurations. In Section 5 we show how to simulate a computation step of the Turing machine, and in Section 6 how the P system is initialised. Section 7 contains an analysis of the resources (time and space) needed to perform this simulation, our main result concerning the relationships between complexity classes for P systems and Turing machines, and corollaries obtained by re-interpreting some classic statements about Turing machines in the membrane computing framework. Finally, Section 8 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 [7], 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 labelled 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 polarisation), 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 labelled by h, having charge α and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by 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 labelled by h, having charge α and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the charge of h is changed to β.

- *Send-out communication rules*, of the form $[a]_h^\alpha \to [\,]_h^\beta\, b$

  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the charge of h is changed to β.

- *Dissolution rules*, of the form $[a]_h^\alpha \to b$

  They can be applied to a membrane labelled by h, having charge α and containing an occurrence of the object a; the membrane h is dissolved and its contents are left in the surrounding region unaltered, except that an occurrence of a becomes b.

- *Elementary division rules*, of the form $[a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$

  They can be applied to a membrane labelled by h, having charge α, containing an occurrence of the object a but having no other membrane inside (an *elementary membrane*); the membrane is divided into two membranes having label h and 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

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

  They can be applied to a membrane labelled 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.

A *configuration* of a P system with active membranes is given by the current membrane structure, including the electrical charges, together with the multisets located in the corresponding regions. In particular, the initial configuration is given by the membrane structure μ, where all membranes are neutral, and the initial contents $w_1, \ldots, w_d$ of its membranes.

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 these rules in a time step (unless the current charge of the membrane prohibits it). The same reasoning applies to each membrane that can be involved in 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 more than one rule can be applied to an object in a timestep, a single rule is chosen nondeterministically from the set of applicable rules. This nondeterministic choice occurs independently for each copy of the object in the membrane. This implies that 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.

Besides general P systems with active membranes, denoted by $\mathcal{AM}$, in this paper we also use a restricted variant without dissolution and elementary 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 $\mathcal{AM}(-\mathrm{d}, -\mathrm{ne})$.

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

P systems can be used as language *recognisers* by employing two distinguished objects *yes* and *no*; exactly one of these must be sent out from the outermost membrane in the last step of each computation, in order to signal acceptance or rejection, respectively; we also assume that all computations are halting. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*. If this is not necessarily the case, then we have a *non-confluent* P system, and the overall result is established as for nondeterministic Turing machines: it is acceptance iff an accepting computation exists.

In order to solve decision problems (i.e., decide languages), we use *families* of recogniser P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x that decides the membership of x in the language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for each input length [4]; in particular, in this paper we employ *logarithmic space* as a complexity bound.

Definition 3. A family of P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be $(\mathbf{L}, \mathbf{L})$-*uniform* if the mapping $x \mapsto \Pi_x$ can be computed by two deterministic logarithmic-space Turing machines F (for "family") and E (for "encoding") as follows:

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

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 [4] 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 [9].

Definition 4. 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 $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a halting computation of Π, then the *space required by* $\vec{\mathcal{C}}$ is defined as

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

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

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

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

The *space required by* Π itself is then

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

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

Definition 5. Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function. We denote by $(\mathbf{L}, \mathbf{L})$-$\mathbf{MC}_{\mathcal{D}}(f(n))$ the class of languages decidable by $(\mathbf{L}, \mathbf{L})$-uniform families of P systems of type $\mathcal{D}$ (in this paper we use $\mathcal{D} = \mathcal{AM}$ or $\mathcal{D} = \mathcal{AM}(-\mathrm{d}, -\mathrm{ne})$) within *time* bound f. By $(\mathbf{L}, \mathbf{L})$-$\mathbf{MCSPACE}_{\mathcal{D}}(f(n))$ we denote the corresponding *space* complexity class. The corresponding classes for non-confluent P systems are $(\mathbf{L}, \mathbf{L})$-$\mathbf{NMC}_{\mathcal{D}}(f(n))$ and $(\mathbf{L}, \mathbf{L})$-$\mathbf{NMCSPACE}_{\mathcal{D}}(f(n))$.

These definitions can be generalised to sets of functions $\mathcal{F}$ by union; for instance,

$$(\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{D}}(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{D}}(f(n))$$

Usually the complexity classes will be defined in terms of

$$\begin{aligned}
\mathbf{poly} &= \{f\colon \mathbb{N} \to \mathbb{N} \mid f(n) \leq p(n) \text{ for some polynomial } p, \text{ for large enough } n\} \\
2^{\mathbf{poly}} &= \{f\colon \mathbb{N} \to \mathbb{N} \mid f(n) \leq 2^{p(n)} \text{ for some polynomial } p, \text{ for large enough } n\} \\
\underbrace{2^{\cdot^{\cdot^{2^{\mathbf{poly}}}}}}_{k \text{ times}} &= \left\{f\colon \mathbb{N} \to \mathbb{N} \mid f(n) \leq \underbrace{2^{\cdot^{\cdot^{2^{p(n)}}}}}_{k \text{ times}} \text{ for some polynomial } p, \text{ for large enough } n\right\}
\end{aligned}$$

that is,

$$\begin{aligned}
(\mathbf{L}, \mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{D}} &= (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{D}}(\mathbf{poly}) \\
(\mathbf{L}, \mathbf{L})\text{-}\mathbf{EXPMCSPACE}_{\mathcal{D}} &= (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{D}}(2^{\mathbf{poly}}) \\
(\mathbf{L}, \mathbf{L})\text{-}k\mathbf{EXPMCSPACE}_{\mathcal{D}} &= (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{D}}\left(\underbrace{2^{\cdot^{\cdot^{2^{\mathbf{poly}}}}}}_{k \text{ times}}\right).
\end{aligned}$$

For the formal definitions and properties of Turing machines and, in particular, the space complexity classes $\mathbf{TIME}(t(n))$, $\mathbf{SPACE}(s(n))$, **PSPACE** and **EXPSPACE**, we refer the reader to [5].

3. Simulation of register machines

Let us recall how P systems with active membranes can simulate register machines [7], probably the simplest way to prove that P systems are able to compute every recursive function. Given a register machine R, we can construct a P system having the following features:

- one membrane for each register of R (a *register-membrane*), having the name of the register as its label and containing as many "unit" objects (denoted by $\bullet$ in the following) as the value of the corresponding register;
- a *program counter object*, representing the instruction of R currently being simulated, initially located outside the membranes associated with the registers.

An increment instruction p: INC$(r), q$, is executed when the program counter value is p; it increments the value of register r and then jumps to instruction q. Such INC instruction is simulated by sending the object p to the membrane r, where it creates another copy of $\bullet$ and is sent back out as q:

$$p\ [\ ]_r^0 \to [p]_r^0 \qquad\qquad [p \to p'\bullet]_r^0 \qquad\qquad [p']_r^0 \to [\ ]_r^0\ q$$

A decrement instruction p: DEC$(r), q_1, q_2$ decreases the value of register r and jumps to q_1 if the value of r is positive, whereas it jumps to q_2 without changing r if the register is zero. It is simulated by sending p to r, changing the charge of the membrane to negative and thus allowing a unit object (if one is present) to be sent out while resetting the charge to neutral. The program counter object waits one step, and then checks the charge in order to establish whether a unit object existed; it is then sent out and rewritten according to the result:

$$\begin{array}{lll} p\ [\ ]_r^0 \to [p]_r^- & [p \to p']_r^- & [\bullet]_r^- \to [\ ]_r^0\ \# \\ [p']_r^0 \to [\ ]_r^0\ q_1 & [p']_r^- \to [\ ]_r^0\ q_2 & \end{array}$$

Here # is a "junk" object, which can be erased by having an evolution rule $[\# \to \lambda]_z^0$ in the outermost membrane z of the P system.

Finally, a halting instruction p: HALT, which stops the computation of the register machine, is simulated by not having any rule involving p on the left-hand side.

Notice how the register-membranes are always neutrally charged, unless the program counter object is located inside one of them. This allows us to make the following key observation: we can use *any set of neutrally charged membranes having different labels and containing a single type of object, together with an extra object located outside them*, as a register machine simulator. This simply requires writing a set of rules simulating the required instructions and generating the program counter object corresponding to the initial instruction when that particular computation is needed. Thus, we can compute arbitrary recursive functions on the values (i.e., the multiplicities of the objects) of the designated register-membranes, attributing to each of them the role of input, output, or auxiliary register. We shall exploit this "subroutine" mechanism in Section 5.

4. Encoding Turing machine configurations

Let M be a Turing machine having tape alphabet $\Sigma = \{\sigma_0, \ldots, \sigma_{K-1}\}$, where σ_0 denotes a blank cell, and set of states $Q = \{q_0, \ldots, q_{L-1}\}$, where q_0 is the initial state, q_{L-2} the rejecting state, and q_{L-1} the accepting state. In the following we will often identify the name of a symbol or state with its index, e.g., writing $q = 3$ for $q = q_3$, or $\Sigma = \{0, \ldots, K-1\}$.

Suppose the current configuration of M is the following:

- the state of the machine is $q \in \{0, \ldots, L-1\}$;
- the visited portion of the tape has length m;
- the string on the visited portion of the tape is $w = w_1 \cdots w_m \in \Sigma^m$;
- the head is placed on tape cell $p \in \{1, \ldots, m\}$.

Then, the configuration of M can be encoded as a configuration of a P system as follows:

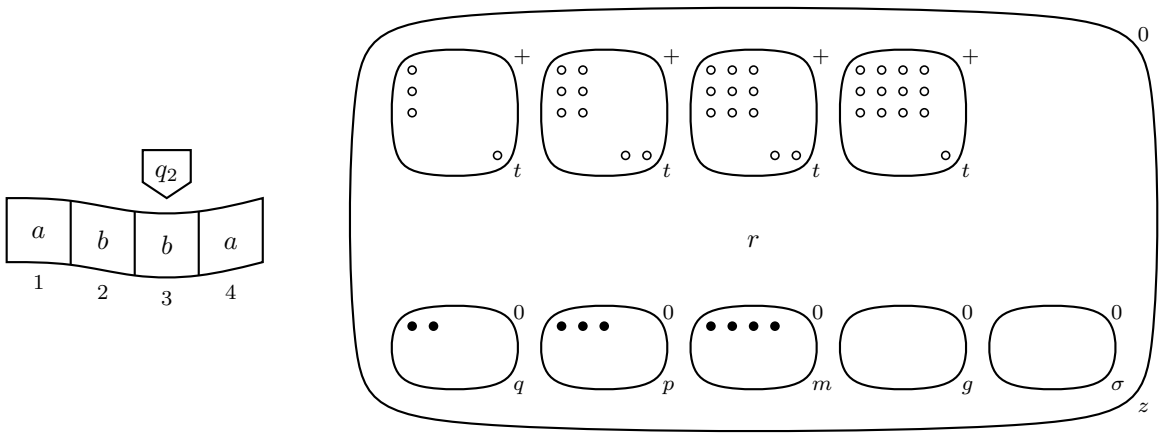

Figure 1: A Turing machine, with $\Sigma = \{\sigma_0, \sigma_1 = a, \sigma_2 = b\}$, and a P system encoding its configuration.

- there are membranes labelled by q, p, and m containing, respectively, the unary encoding of q, p, and m, i.e., as many copies of the "unit" object $\bullet$ as the corresponding value;
- there are m membranes labelled by t, one for each visited tape cell; each one contains the number $K \times i + j$, where i is the index of the corresponding tape cell, and σ_j is the symbol written on it, encoded in unary as the number of copies of the unit object $\circ$.

Notice that i and j can be easily recovered from $K \times i + j$ simply by taking the quotient of the integer division by K and the reminder of that division, respectively.

In the following, we will often refer to the "value of h" (for some membrane label h) to denote the number of unit objects contained inside membrane h.

5. Simulating a computation step of the Turing machine

The major difficulty in simulating a computation step of the Turing machine M consists in identifying, among all membranes labelled by t, the one corresponding to the cell located under the tape head of M. Indeed, these membranes are externally indistinguishable (i.e., with respect to a single application of active membrane rules), and differ only in the unary value contained in it. Hence, we will resort to *nondeterministically guessing* a membrane, then checking whether it is indeed the right one; if this is not the case, then the membrane will be marked, and the process repeated until we eventually find the correct one, ensuring the confluence of the simulation.[1]

Specifically, the guessing phase works as follows. First, the object r enters one membrane labelled by t and changes its charge to neutral; in the next step, r is sent out as r_0 while the unit objects $\circ$ are rewritten to $\bullet$. The corresponding rules are

$$r\ [\ ]_t^+ \to [r]_t^0 \qquad [r]_t^0 \to [\ ]_t^0\ r_0 \qquad [\circ \to \bullet]_t^0 \tag{R_1}$$

As we can observe in Figure 2, no pair of neutrally charged membranes of Π_x share the same label, and they all contain a single kind of object (namely, the unit object $\bullet$). Hence they can be employed, together with object r_0, representing the label of an instruction, as a register machine, as mentioned in Section 3. The complete pseudocode for the register machine is described in Figure 3 and analysed below.

[1] Note that if we consider a non-confluent family, where a single accepting computation is sufficient, then Π_x could just reject on guessing the wrong membrane. This would speed up the simulation by $O(s(n))$, where $s(n) = |\Pi_x|$.

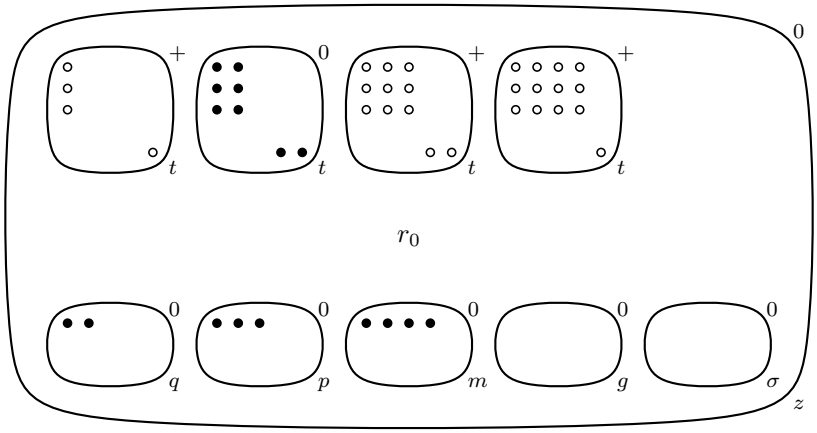

Figure 2: Configuration of the P system after having nondeterministically chosen a membrane labelled by t.

Remark 6. Although the pseudocode for the register machine only mentions registers having the same names as the membranes of the P system, its translation into actual register machine instructions requires a number of auxiliary registers. For instance, copying a register x into y usually employs an auxiliary register z: register x is decremented until it reaches 0, while both y and z are simultaneously incremented; finally, z is copied back to x. These auxiliary registers will be implemented as another set of membranes in the P system. However, since their number is constant (at most equal to the number of register machine instructions) they will not be mentioned explicitly in the rest of the paper; we will also assume that appropriate register machine instructions will reset to 0 the auxiliary registers after their use, in order to avoid conflicts with later instructions.

The first task of the register machine is to check whether the membrane that was chosen corresponds to the correct tape cell, i.e., whether the quotient of t and K equals the value of p. The corresponding pseudocode is

r_0: $g \coloneqq g + 1$
if t div $K \neq p$ **then**
r_1: **halt**
end

The register machine also increments the value of membrane g, denoting the number of guesses made until now. This value will be used later in order to correctly reset the membranes marked as "wrong guesses".

If the P system made a wrong guess, then the execution reaches the line labelled by r_1, corresponding to a **halt** instruction. The object r_1 now appears in the outermost membrane of Π_x, and is used to mark the chosen membrane by changing its charge to negative, while simultaneously resetting all $\bullet$ objects to $\circ$. The corresponding rules are

$$r_1\ [\ ]_t^0 \to [r_1]_t^+ \qquad [r_1]_t^+ \to [\ ]_t^-\ r \qquad [\bullet \to \circ]_t^+ \qquad (R_2)$$

Now the P system will once again apply the rules in R_1, choosing another membrane labelled by t.

Eventually, the correct membrane will be chosen, the instruction r_1 will be skipped, and the register machine simulation will proceed as follows:

$\sigma \coloneqq t \bmod K$
$t \coloneqq t - \sigma$

The first instruction extracts the index of the symbol in the scanned tape cell and sets the contents of membrane σ to that value. The second instruction erases the symbol from the tape cell.

Now the transition function $\delta\colon ((Q-\{q_{L-2}, q_{L-1}\})\times\Sigma) \to (Q\times\Sigma\times\{-1,0,+1\})$ of M is simulated. This can be implemented by a series of selection statements over all possible pairs of values of $q \in \{0,\ldots,L-3\}$ and $\sigma \in \{0,\ldots,K-1\}$. If the current pair is (q_i,σ_j), also written as (i,j), in the code we denote the next state, symbol to be written, and direction of the motion of the tape head as $Q_{i,j}$, $\Sigma_{i,j}$, and $D_{i,j}$ respectively (notice that these are *constants* in the program, rather than variables).

if $q = 0$ and $\sigma = 0$ **then**
  $q \coloneqq Q_{0,0}$
  $\sigma \coloneqq \Sigma_{0,0}$
  $p \coloneqq p + D_{0,0}$
else if $q = 0$ and $\sigma = 1$ **then**
  $q \coloneqq Q_{0,1}$
  $\sigma \coloneqq \Sigma_{0,1}$
  $p \coloneqq p + D_{0,1}$
else if $\cdots$ **then**
  $\vdots$
else if $q = L-3$ and $\sigma = K-1$ **then**
  $q \coloneqq Q_{L-3,K-1}$
  $\sigma \coloneqq \Sigma_{L-3,K-1}$
  $p \coloneqq p + D_{L-3,K-1}$
end

The values of q and p are now up-to-date, while the symbol σ still has to be copied to t. However, we first check whether by moving the tape head we exceeded the last visited tape cell, thus requiring a new membrane t. This is accomplished by the following instructions:

if $p > m$ **then**
  $m \coloneqq m + 1$
r_2: **halt**
end

If instruction r_2 is reached, the register machine simulation is suspended, and the object r_2 is used in order to create a new membrane t by elementary division. First r_2 enters the neutral membrane t, which corresponds to the rightmost tape cell visited up to now, then it causes its division, creating a new copy of t with positive charge (causing the rewriting of $\bullet$ into $\circ$) and incrementing by K its value.

$$r_2\,[\,]_t^0 \to [r_2]_t^0 \qquad [r_2]_t^0 \to [\#]_t^0\,[r_2']_t^+ \qquad [r_2' \to r_2''\underbrace{\circ\cdots\circ}_{K \text{ copies}}]_t^+ \qquad [r_2'']_t^+ \to [\,]_t^+\,r_3 \qquad (R_3)$$

On the other hand, if $p \leq m$, the simulation of the register machine is not suspended, and we reach instruction r_3 directly.

With the appearance of object r_3, the simulated register machine finally writes the symbol σ in t, while resetting the contents of membrane σ to 0:

r_3: $t \coloneqq t + \sigma$
  $\sigma \coloneqq 0$
r_4: **halt**

The current configuration of Π_x contains as many non-positive membranes t as the value of g. These are reset to positive by means of the unit objects contained in g, which are sent out one by one by using the following rules:

$$\begin{array}{llll} r_4\,[\,]_g^0 \to [r_4]_g^+ & [\bullet]_g^+ \to [\,]_g^0\,\bullet & [r_4 \to r_4']_g^+ & \\ \bullet\,[\,]_t^0 \to [\#]_t^+ & \bullet\,[\,]_t^- \to [\#]_t^+ & [r_4']_g^0 \to [\,]_g^0\,r_4 & [r_4']_g^+ \to [\,]_g^0\,r_5 \end{array} \qquad (R_4)$$

Although the membranes t are reset to positive in a nondeterministic order, the configuration reached at the end of the resetting phase is unique (i.e., the computation is confluent in this phase).

When the object r_5 appears, a final set of register machine instructions are simulated: these check whether the new state of the Turing machine is the rejecting state q_{L-2} (which produces the object *no* in the outermost membrane of Π_x), the accepting state q_{L-1} (producing the object *yes*), or any other state (producing the object r, which triggers the simulation of another step of M).

r_5: **if** $q = L - 2$ **then**
no: **halt**
else if $q = L - 1$ **then**
yes: **halt**
else
r: **halt**
end

Notice that, while all three branches of these selection statements lead to a **halt** instruction, the label of the instruction (which is the way the register machine simulator communicates with the rest of the P system) is different.

If a final state of the Turing machine was reached, the corresponding object is sent out of the outermost membrane as the result of the computation of the P system:

$$[no]_z^0 \to [\,]_z^0\, no \qquad\qquad [yes]_z^0 \to [\,]_z^0\, yes \qquad (R_5)$$

When this happens, no more rules are applicable and the P system halts.

We can erase the "junk" objects # from the configuration of the P system, in order not to increase its space complexity beyond necessary, by using the following evolution rules:

$$[\# \to \lambda]_z^0 \qquad [\# \to \lambda]_t^- \qquad [\# \to \lambda]_t^0 \qquad [\# \to \lambda]_t^+ \qquad (R_6)$$

6. Initialisation of the P system

In order to set up a configuration of Π_x encoding the initial configuration of the simulated Turing machine we proceed as follows.[2] The initial configuration of Π_n contains n empty membranes labelled by t, and the input of the Turing machine $x = x_1 \cdots x_n \in \Sigma^n$ is encoded as a multiset w_x (without repeated objects) over the alphabet $\{\sigma_{j,i} : 1 \le i \le n, 0 \le j < K\}$ as follows:

$$w_x = \{\sigma_{j,i} : \sigma_j = x_i\}$$

The multiset w_x is placed inside the outermost membrane of Π_n. Figure 4 shows an example.

The first two computation steps of Π_n on input w_x move the input objects to membranes labelled by t and create the correct amount of unit objects $\circ$ describing the contents of the tape cells of M:

$$\sigma_{j,i}\,[\,]_t^0 \to [\sigma_{j,i}]_t^+ \qquad [\sigma_{j,i} \to \underbrace{\circ \cdots \circ}_{Ki+j \text{ copies}}\,]_t^+ \qquad \text{for } 1 \le i \le n,\ 0 \le j < K \qquad (R_7)$$

In the mean time, object r'' evolves to r' and then to r, ensuring that the configuration of the P system after two steps correctly encodes the initial configuration of M:

$$[r'' \to r']_z^0 \qquad\qquad [r' \to r]_z^0 \qquad (R_8)$$

[2]Here we assume that multiple membranes having the same label are allowed in the initial configuration of Π_x. If this is not the case, then these membranes can be simply renamed as $t_1, \ldots, t_n$, and each rule involving t is repeated for each $t_1, \ldots, t_n$. The number of rules only increases polynomially in this variant.

```
r_0 :  g ≔ g + 1
       if t div K ≠ p then
r_1 :      halt
       end
       σ ≔ t mod K
       t ≔ t − σ

       if q = 0 and σ = 0 then
           q ≔ Q_{0,0}
           σ ≔ Σ_{0,0}
           p ≔ p + D_{0,0}
       else if q = 0 and σ = 1 then
           q ≔ Q_{0,1}
           σ ≔ Σ_{0,1}
           p ≔ p + D_{0,1}
       else if ··· then
           ⋮
       else if q = L − 3 and σ = K − 1 then
           q ≔ Q_{L−3,K−1}
           σ ≔ Σ_{L−3,K−1}
           p ≔ p + D_{L−3,K−1}
       end

       if p > m then
           m ≔ m + 1
r_2 :      halt
       end
r_3 :  t ≔ t + σ
       σ ≔ 0
r_4 :  halt
r_5 :  if q = L − 2 then
no :       halt
       else if q = L − 1 then
yes :      halt
       else
r :        halt
       end
```

Figure 3: Complete pseudocode for the register machine simulating a computation step of the Turing machine.

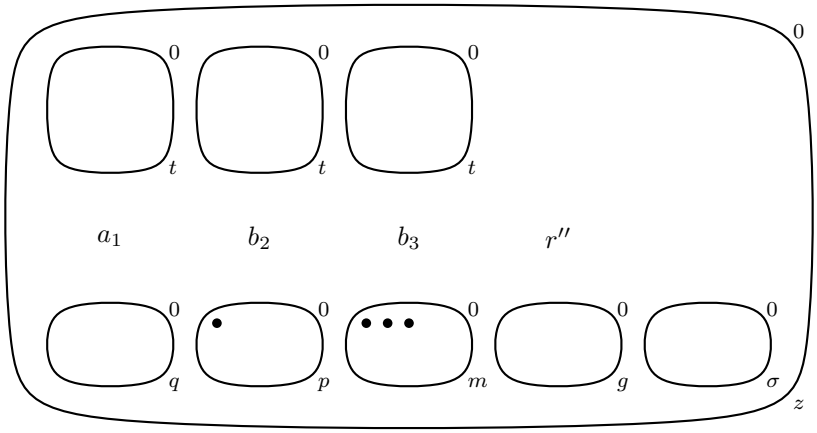

Figure 4: Initial configuration of a P system simulating a Turing machine on input $x = abb$ where $n = |x| = 3$.

7. Main result and consequences

In order to obtain a meaningful simulation of Turing machines by means of families of P systems, we need to provide a weak enough uniformity condition on them, to avoid "cheating" by hiding most of the computation in the construction of the P systems.

Lemma 7. *The family of P systems* $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *described above is* $(\mathbf{L}, \mathbf{L})$*-uniform.*

Proof. Notice that the only rules among $R_1, \ldots, R_8$ actually depending on the length n of the input are those of type R_7, and these are easy to output by using two nested loops with variables ranging over the length of the input and the size of the alphabet (which is a constant), hence in logarithmic space. The initial membrane structure of Π_n is also fixed, except for the number of membranes labelled by t and the number of unit objects $\bullet$ inside membrane m, both equal to n; these can also be output in logarithmic space using a single loop. Hence, the mapping $1^n \mapsto \Pi_n$ is computable in logarithmic space. The mapping $x \mapsto w_x$ can be also computed in logarithmic space, by simply indexing the input symbols with their position in the string x. □

We are now finally able to prove our main result.

Theorem 8. *Let M be a single-tape deterministic Turing machine working in time $t(n)$ and space $s(n)$, including the space required for its input. Then there exists an $(\mathbf{L}, \mathbf{L})$-uniform family $\mathbf{\Pi}$ of confluent P systems with restricted elementary active membranes working in time $O(t(n)s(n)^2) \subseteq O(t(n)^3)$ and space $O(s(n)^2)$ such that $L(M) = L(\mathbf{\Pi})$.*

Proof. The family $\mathbf{\Pi}$ described above works in quadratic space with respect to $s(n)$, since each P system Π_x with $|x| = n$ has $s(n)$ membranes labelled by t, and a constant number of membranes with other labels (including any auxiliary membrane, as mentioned in Remark 6); each membrane contains at most $O(s(n))$ unit objects, as many as the index of the rightmost tape cell of M.

Each simulated step of M requires at most $s(n)$ guesses before locating the correct membrane with label t, and checking whether a membrane corresponds to the position p can be performed in linear time with respect to the sizes of the numbers involved, once again bounded by $s(n)$. For the same reason, the configuration can be updated in $O(s(n))$ time. Since $s(n) \in O(t(n))$ for Turing machines, the slowdown of the simulation is at most cubic.

The family of P systems is confluent since the only two nondeterministic phases (guessing the membrane corresponding to the cell under the tape head of M, and resetting all charges of the membranes labelled by t to positive) are confluent, as mentioned in Section 5. The rest of the simulation is strictly deterministic.

The construction ensures that both devices have the same acceptance result, and Lemma 7 that the family is $(\mathbf{L}, \mathbf{L})$-uniform. □

Notice that the uniformity condition for $\mathbf{\Pi}$ ensures that the P systems themselves carry out the simulation of the Turing machine M, as opposed to the machines constructing them, whenever the problem they solve is outside $\mathbf{L}$, e.g., in the case of **PSPACE**-hard problems. Theorem 8 can be also compared to the previously known simulations of polynomial-space P systems [16, 10], requiring only $O(t(n))$ time and $O(s(n))$ space, and exponential-space P systems [2], requiring $O(t(n)^2 \log t(n))$ time and $O(s(n) \log s(n))$ space, which are more efficient in those restricted cases.

The simulation of Theorem 8 can be made faster, and also generalised to nondeterministic Turing machines, by using non-confluent P systems.

Theorem 9. *Let M be a single-tape, possibly nondeterministic Turing machine working in time $t(n)$ and space $s(n)$, including the space required for its input. Then there exists an $(\mathbf{L}, \mathbf{L})$-uniform family $\mathbf{\Pi}$ of non-confluent P systems with restricted elementary active membranes working in time $O(t(n)s(n)) \subseteq O(t(n)^2)$ and space $O(s(n)^2)$ such that $L(M) = L(\mathbf{\Pi})$.*

Proof sketch. The register machine simulation of Section 3 can be easily extended to include nondeterministic increment instructions of the form $p\colon \text{INC}(r), q_1, q_2$, which can be simulated by conflicting evolution rules by the P system. Nondeterministic increment can be used to implement unconditional nondeterministic jumps by using a scratch register.

The simulation of a step of the Turing machine (Section 5) is modified by deleting the rules of type R_2 and replacing them by $[r_1]^0_z \to [\ ]^0_z\ no$. This way, when a wrong membrane is guessed, the P system just halts by rejecting instead of marking the membrane and making another guess.

If the correct membrane t is guessed, and the symbol σ has been read from it, the simulated register machine chooses one among all possible transitions of M from (q, σ) by using nondeterministic jumps, and updates the configuration, including creating a new membrane t when needed, as before.

The resulting P systems behave as follows. Every accepting computation of M on input x has a corresponding computation of Π_x, where the correct membrane t is always guessed during each simulated step, and the nondeterministic choices made by M are mimicked by the register machine instructions. Beside these accepting computations, Π_x always rejects, either because a wrong membrane t is guessed, or because the nondeterministic choices of M simulated by it lead to rejection. Hence, the machine M and the family $\mathbf{\Pi}$ decide the same language.

The space required by this simulation is the same as Theorem 8, i.e., $O(s(n)^2)$ space, but the time required is only $O(t(n)s(n)) \subseteq O(t(n)^2)$, since only one membrane t per simulated step is guessed. □

From Theorem 8 we obtain inclusions of complexity classes for Turing machines and P systems when the space bound is at least linear (since we are dealing with single-tape Turing machines).

Theorem 10. *For every function $f(n) \in \Omega(n)$ the following inclusions hold:*

$$\begin{array}{lcllcl}
\mathbf{TIME}(f(n)) & \subseteq & (\mathbf{L},\mathbf{L})\text{-}\mathbf{MC}_{\mathcal{AM}(-\mathrm{d},-\mathrm{ne})}(O(f(n)^3)) & \subseteq & (\mathbf{L},\mathbf{L})\text{-}\mathbf{MC}_{\mathcal{AM}}(O(f(n)^3)) \\
\mathbf{SPACE}(f(n)) & \subseteq & (\mathbf{L},\mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}(-\mathrm{d},-\mathrm{ne})}(O(f(n)^2)) & \subseteq & (\mathbf{L},\mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(f(n)^2)).
\end{array}$$

Let us now recall how P systems may be simulated by Turing machines with a polynomial space overhead [11].

Theorem 11. *Let $\mathbf{\Pi}$ be a $(\mathbf{L}, \mathbf{L})$-uniform confluent (resp., non-confluent) family of recogniser P systems with active membranes working in space $s(n)$; let $t(n) \in \mathbf{poly}$ be the time complexity of the Turing machine F computing the mapping $1^n \mapsto \Pi_n$. Then, $\mathbf{\Pi}$ can be simulated by a deterministic (resp., nondeterministic) Turing machine working in space $O(s(n)t(n) \log s(n))$.*

Proof sketch. Since F works in logarithmic space, hence polynomial time, its output must have polynomial size, including both the initial configuration of Π_x and its set of rules. Hence, the polynomial $t(n)$ is an upper bound on the length of the description of Π_x, to the number of labels $|\Lambda|$ and to the number of types of objects $|\Gamma|$ of Π_x.

Storing the membrane structure of Π_x as a tree data structure requires $O(s(n)t(n)\log s(n))$ space: there are $O(s(n))$ nodes, each having a label of $O(\log t(n))$ bits, $O(1)$ bits for the electrical charge, $O(t(n)\log s(n))$ space for storing multisets as vectors of binary numbers, and $O(\log s(n))$ space for pointers to adjacent membranes.

Simulating one computation step of Π_x is performed by first selecting (nondeterministically, in the non-confluent case) a maximal multiset of rules to be applied in the current configuration[3]; then, the rules are applied in a bottom-up way, from the elementary membranes (the leaves of the tree) towards the outermost membrane (corresponding to the root). The auxiliary data structures needed in order to simulate a step, i.e., a copy of the previous configuration and a stack for performing a depth-first search of the tree structure, do not exceed $O(s(n)t(n)\log s(n))$ space. □

By combining Theorem 8 and Theorem 11, we can prove *equality* between space complexity classes for P systems and Turing machines under some (not very restrictive) assumptions on the set of space bounds we are interested in.

Theorem 12. *Let $\mathcal{F}$ be a class of functions $\mathbb{N} \to \mathbb{N}$ such that*

- *$\mathcal{F}$ contains the identity function $n \mapsto n$;*
- *If $s(n) \in \mathcal{F}$ and $p(n)$ is a polynomial, then there exists some $f(n) \in \mathcal{F}$ with $f(n) \in \Omega(p(s(n)))$.*

Then $\mathbf{SPACE}(\mathcal{F}) = (\mathbf{L},\mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(\mathcal{F})$. *In particular, we have the following equalities:*

$$\mathbf{PSPACE} = (\mathbf{L},\mathbf{L})\text{-}\mathbf{PMCSPACE}_{\mathcal{AM}} \qquad \mathbf{EXPSPACE} = (\mathbf{L},\mathbf{L})\text{-}\mathbf{EXPMCSPACE}_{\mathcal{AM}}$$
$$2\mathbf{EXPSPACE} = (\mathbf{L},\mathbf{L})\text{-}2\mathbf{EXPMCSPACE}_{\mathcal{AM}} \qquad k\mathbf{EXPSPACE} = (\mathbf{L},\mathbf{L})\text{-}k\mathbf{EXPMCSPACE}_{\mathcal{AM}}.$$

Proof. Let $L \in \mathbf{SPACE}(\mathcal{F})$; then, there exists a single-tape Turing machine M deciding L in space $s(n)$ for some non-decreasing $s(n) \in \mathcal{F}$. By Theorem 8, there exists an $(\mathbf{L},\mathbf{L})$-uniform family $\mathbf{\Pi}$ of P systems simulating M and deciding L in space $g(n) \in O(s(n)^2)$. Let $p(n) = n^3$; then, there exists $f(n) \in \mathcal{F}$ with $f(n) \in \Omega(p(s(n))) = \Omega(s(n)^3) \subseteq \Omega(g(n))$, hence $L \in (\mathbf{L},\mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(f(n)) \subseteq (\mathbf{L},\mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(\mathcal{F})$.

Conversely, let $L \in (\mathbf{L},\mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(\mathcal{F})$. Then, L is decided by an $(\mathbf{L},\mathbf{L})$-uniform family $\mathbf{\Pi}$ of P systems working in space $s(n) \in \mathcal{F}$; let $t(n)$ be a polynomial upper bound to the time complexity of the Turing machine F computing the mapping $1^n \mapsto \Pi_n$. By Theorem 11, there exists a Turing machine M simulating $\mathbf{\Pi}$ in space $g(n) \in O(s(n)t(n)\log s(n))$. If $t(n) \in O(s(n))$, then we have $g(n) \in O(f(n))$ for some $f(n) \in \Omega(s(n)^3)$ with $f(n) \in \mathcal{F}$ by the hypotheses on $\mathcal{F}$, hence $L \in \mathbf{SPACE}(f(n)) \subseteq \mathbf{SPACE}(\mathcal{F})$. On the other hand, if $t(n) \in \Omega(s(n))$, then we have $g(n) \in O(t(n)^2 \log t(n)) \subseteq O(p(n))$ for some polynomial p. By hypothesis, we have $f(n) \in \Omega(p(n))$ for some $f(n) \in \mathcal{F}$, hence $L \in \mathbf{SPACE}(f(n)) \subseteq \mathbf{SPACE}(\mathcal{F})$.

It is easy to check that the classes $\mathbf{poly}$, $2^{\mathbf{poly}}$, $2^{2^{\mathbf{poly}}}$ and so on satisfy the hypotheses on $\mathcal{F}$; these classes cover most common complexity classes encountered in the literature. □

Another consequence of the possibility of P systems to simulate Turing machines with a polynomial overhead and vice versa is that we are now able to translate theorems about the space complexity of Turing machines into theorems about P systems. As an example, the following two corollaries can be proved almost immediately for large enough space complexity bounds[4].

Corollary 13 (Savitch's theorem for P systems)**.** *For each function $s(n)$ growing faster than every polynomial we have* $(\mathbf{L},\mathbf{L})\text{-}\mathbf{NMCSPACE}_{\mathcal{AM}}(s(n)) \subseteq (\mathbf{L},\mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(s(n)^8 \log^4 s(n)))$.

[3]The specific algorithm employed in order to select maximal multisets of rules is irrelevant, as long as it can be executed in $O(s(n)t(n)\log s(n))$ space; even a brute-force enumeration of all possible multisets of rules of cardinality up to $s(n)$, followed by a maximality check, can be performed within such space bound. We refer the reader to the original paper [11] for further details.

[4]Corollaries 13 and 14 can be proved, in the restricted polynomial and exponential space cases, with tighter space bounds by using the simulations in [10] and [2] respectively (see also [16]).

Proof. By Theorem 11, $s(n)$-space families of P systems can be simulated in $O(s(n)^2 \log s(n))$ space by Turing machines, since $t(n) \in o(s(n))$; hence

$$(\mathbf{L}, \mathbf{L})\text{-}\mathbf{NMCSPACE}_{\mathcal{AM}}(s(n)) \subseteq \mathbf{NSPACE}(O(s(n)^2 \log s(n))).$$

By the original Savitch's theorem [13] we have

$$\mathbf{NSPACE}(O(s(n)^2 \log s(n))) \subseteq \mathbf{SPACE}(O(s(n)^4 \log^2 s(n))).$$

Finally, Theorem 10 proves that the latter class is included in $(\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(s(n)^8 \log^4 s(n)))$. □

As a consequence, for all classes of functions $\mathcal{F}$ satisfying the hypotheses of Theorem 12 we have

$$(\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(\mathcal{F}) = (\mathbf{L}, \mathbf{L})\text{-}\mathbf{NMCSPACE}_{\mathcal{AM}}(\mathcal{F}).$$

Corollary 14 (Space hierarchy theorem for P systems)**.** *Let $s(n)$ be a function growing faster than every polynomial, and let $f(n)$ be a space-constructible function such that $O(s(n)^2 \log s(n)) \subseteq o(f(n))$. Then*

$$(\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(s(n)) \subsetneq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(f(n)^2)).$$

Proof. By Theorem 11 and assuming $t(n) \in o(s(n))$ we have

$$(\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(s(n)) \subseteq \mathbf{SPACE}(O(s(n)^2 \log s(n))).$$

Using the hypotheses on $f(n)$ we obtain

$$\mathbf{SPACE}(O(s(n)^2 \log s(n))) \subseteq \mathbf{SPACE}(o(f(n))).$$

The space hierarchy theorem for Turing machines [13] proves that

$$\mathbf{SPACE}(o(f(n))) \subsetneq \mathbf{SPACE}(O(f(n)))$$

Finally, by Theorem 10 we obtain

$$\mathbf{SPACE}(O(f(n))) \subseteq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(f(n)^2))$$

and the thesis follows. □

For instance, if $s(n)$ grows faster than every polynomial and is space-constructible we have

$$(\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(s(n)) \subsetneq (\mathbf{L}, \mathbf{L})\text{-}\mathbf{MCSPACE}_{\mathcal{AM}}(O(s(n)^{4+\epsilon}))$$

for each $\epsilon > 0$. Indeed, the function $f(n) = s(n)^{2+\frac{\epsilon}{2}}$ is space-constructible for each rational $\epsilon > 0$ [13] and we have $O(s(n)^2 \log s(n)) \subseteq o(s(n)^{2+\frac{\epsilon}{2}})$.

8. Conclusions

We proved that single-tape Turing machines can be simulated by $(\mathbf{L}, \mathbf{L})$-uniform families of P systems with restricted elementary active membranes with a cubic slowdown and a quadratic space overhead. This leads to inclusions between complexity classes for Turing machines and P systems, and equalities as long as the sets of space bounds satisfies some reasonable properties. In particular, complexity classes defined in terms of polynomial, exponential, double exponential, ..., n-fold exponential space coincide for the two kinds of device. We are also able to translate results involving relationships between space complexity classes for Turing machines to P systems: in fact, we can first simulate the P systems by means of Turing machines, then apply the theorem we want to translate, and finally simulate the resulting Turing machines by P systems. As an example, we proved a "Savitch's theorem" and a space hierarchy theorem for P systems.

Let us note that if membrane creation [1] is used instead of membrane division, then the simulation of deterministic Turing machines by P systems may be straightforward and faster (the slowdown would only be linear). 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 construction also generalises to arbitrarily large space bounds. However, with membrane division only, the depth of the membrane hierarchy cannot increase during the computation, and it is at most polynomial under reasonable uniformity condition.

As a direction for future research, it might also be interesting to characterise the behaviour of families of P systems with active membranes working in sublinear space. A first step in this direction has been made in [12], where it was proved that uniform families of P systems with active membranes working in logarithmic space can simulate logarithmic-space deterministic Turing machines. Note, however, that there are two major issues to be considered in this case: first, we have to 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 logarithmic-space Turing machine). Furthermore, the notion of uniformity used to define the families of P systems has to be weakened [4, 12], since logarithmic-space Turing machines constructing the families might be able to solve the problems altogether by themselves.

Acknowledgements

This work was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo (FA) 2012, and by the Lombardy region project NEDD. We would like to thank the anonymous referees for their suggestions on a previous version of this paper.

References

[1] A. Alhazov, R. Freund, A. Riscos-Núñez, Membrane division, restricted membrane creation and object complexity in P systems, International Journal of Computer Mathematics 83 (2006) 529–547. URL: http://dx.doi.org/10.1080/00207160601065314.

[2] A. Alhazov, A. Leporati, G. Mauri, A.E. Porreca, C. Zandron, The computational power of exponential-space P systems with active membranes, in: M.A. Martínez-del-Amor, Gh. Păun, I. Pérez-Hurtado, F.J. Romero-Campero (Eds.), Proceedings of the Tenth Brainstorming Week on Membrane Computing, volume I, Fénix Editora, 2012, pp. 35–60. URL: http://www.gcn.us.es/icdmc2012_proceedings.

[3] A. Alhazov, C. Martín-Vide, L. Pan, Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes, Fundamenta Informaticae 58 (2003) 67–77. URL: http://iospress.metapress.com/content/99n72anvn6bkl4mm/.

[4] N. Murphy, D. Woods, The computational power of membrane systems under tight uniformity conditions, Natural Computing 10 (2011) 613–632. URL: http://dx.doi.org10.1007/s11047-010-9244-7.

[5] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1993.

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

[7] Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, 2010.

[8] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Complexity classes in models of cellular computing with membranes, Natural Computing 2 (2003) 265–284. URL: http://dx.doi.org/10.1023/A:1025449224520.

[9] A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, Introducing a space complexity measure for P systems, International Journal of Computers, Communications & Control 4 (2009) 301–310. URL: http://www.journal.univagora.ro/?page=article_details&id=375.

[10] A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, P systems simulating oracle computations, in: M. Gheorghe, Gh. Păun, A. Salomaa, G. Rozenberg, S. Verlan (Eds.), Membrane Computing, 12th International Conference, CMC 2011, volume 7184 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 346–358. URL: http://dx.doi.org/10.1007/978-3-642-28024-5_23.

[11] A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, P systems with active membranes working in polynomial space, International Journal of Foundations of Computer Science 22 (2011) 65–73. URL: http://dx.doi.org/10.1142/S0129054111007836.

[12] A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, Sublinear-space P systems with active membranes, in: E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, G. Vaszil (Eds.), Membrane Computing, 13th International Conference, CMC 2012, volume 7762 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 342–357. URL: http://dx.doi.org/10.1007/978-3-642-36751-9_23.

[13] M. Sipser, Introduction to the Theory of Computation, Second Edition, Thomson Course Technology, 2006.
[14] P. Sosík, The computational power of cell division in P systems: Beating down parallel computers?, Natural Computing 2 (2003) 287–298. URL: http://dx.doi.org/10.1023/A:1025401325428.
[15] P. Sosík, A. Rodríguez-Patón, Membrane computing and complexity theory: A characterization of PSPACE, Journal of Computer and System Sciences 73 (2007) 137–152. URL: http://dx.doi.org/10.1016/j.jcss.2006.10.001.
[16] A. Valsecchi, A.E. Porreca, A. Leporati, G. Mauri, C. Zandron, An efficient simulation of polynomial-space Turing machines by P systems with active membranes, in: Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing, 10th International Workshop, WMC 2009, volume 6501 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 461–478. URL: http://dx.doi.org/10.1007/978-3-642-11467-0_31.
[17] C. Zandron, C. Ferretti, G. Mauri, Solving NP-complete problems using P systems with active membranes, in: I. Antoniou, C.S. Calude, M.J. Dinneen (Eds.), Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference, Springer, 2001, pp. 289–301. URL: http://dx.doi.org/10.1007/978-1-4471-0313-4_21.