On a Powerful Class of Non-Universal P Systems with Active Membranes

Antonio E. Porreca, Alberto Leporati, and Claudio Zandron

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

Abstract. We prove that uniform and semi-uniform families of P systems with active membranes using only communication and nonelementary division rules are not computationally universal. However, they are powerful enough to solve exactly the problems solvable by Turing machines operating in time and space that are "tetrational" (i.e., bounded by a stack of exponentials of polynomial height) with respect to the size of the input.

1 Introduction

Membrane systems, usually called *P systems*, have been introduced in [4] as computing devices inspired by the internal working of biological cells. The main feature of P systems is a structure of membranes dividing the space into regions, inside which multisets of objects describe the molecular environment. A set of rules describe how the molecules (and often the membranes themselves) evolve during the computation; usually, the rules are applied in a maximally parallel way, i.e., each component of the P system *must* be subject to a rule during each computation step, if a suitable rule exists. When multiple rules may be applied to an object or membrane, one of them is nondeterministically chosen. The computation, starting from an initial configuration, proceeds until no further rule can be applied. For an introduction to membrane computing, we refer the reader to [7,8], and for the latest information to the P Systems Webpage [14], where an extensive bibliography on the topic can be found.

Families of P systems can be used as language recognizers, by associating with each input string (or with each input length) a P system; this association is subject to a uniformity condition (i.e., it must be computed by a Turing machine operating in polynomial time). The constructed P systems can then accept or reject, thus deciding the membership of strings to the language. The computational complexity of recognizer P systems with *active membranes* [5], where the membranes themselves play an important role during the computation, has been subject to extensive investigation, due to their ability of solving **NP**-complete [5] and even **PSPACE**-complete problems [13] in polynomial time: this efficiency is due to the possibility of creating in polynomial time an exponential

number of membranes, which then evolve in parallel, using *membrane division* (a process found in nature, e.g., in cell mitosis).

From the computability standpoint, this class of P systems is known to be equivalent to Turing machines [6]: this result, however, is proved by using *object evolution* rules, which can increase the number of objects an arbitrary number of times. In this paper, we analyze the case in which only *communication* rules (which move an object from a region to another) and *nonelementary division* rules (which only apply to membranes containing other membranes) are allowed, and prove that the resulting P systems are not computationally universal; nonetheless, they are very powerful, as they characterize the class of languages decidable by Turing machines using time (or, equivalently, space) bounded by an exponential function iterated polynomially many times (known as *tetration*).

The paper is organized as follows. In Section 2 we define P systems with active membranes, their use as language recognizers and their space complexity; we also recall the notion of tetration and introduce a class of languages having that kind of complexity. In Section 3 we prove that P systems with active membranes using only communication and nonelementary division rules are not universal, by showing how they can be simulated by tetrational-space bounded Turing machines. In Section 4 we introduce a P system module which will then be used in Section 5 in order to prove that the reverse simulation is also possible, thus characterizing exactly the power of our class of P systems. Section 6 concludes the paper and describes possible future research.

2 Definitions

Let us start by recalling the definition of the model of P systems we will use in this paper.

Definition 1. *A* P system with active membranes *with polarizations (using only communication and nonelementary division rules) of degree $m \geq 1$, is a tuple*

$$\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_m, R)$$

where:

- *Γ is a finite alphabet of symbols, also called* objects*;*
- *Λ is a finite set of labels for the membranes;*
- *μ is a membrane structure (i.e., a rooted unordered tree) consisting of m membranes enumerated by $1, \ldots, m$. Furthermore, each membrane is labeled by an element of Λ, non necessarily in an injective way;*
- *$w_1, \ldots, w_m$ are strings over Γ, describing the multisets of objects placed in the m initial regions of μ;*
- *R is a finite set of rules.*

The membrane corresponding to the root of μ is called the skin*, the ones corresponding to the leaves* elementary membranes*, whereas the others are called*

nonelementary. Each membrane possesses a further attribute, named polarization *or* electrical charge, *which is either neutral (represented by 0), positive (+) or negative (−).*

The rules are of the following kinds:

– 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 polarization 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 polarization of h is changed to β.
– Send-out communication rules, *of the form* $[a]_h^\alpha \to [\;]_h^\beta\,b$
  They can be applied to a membrane labeled by h, having polarization 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 polarization of h is changed to β.
– Non-elementary division rules, *of the form*

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

  They can be applied to a membrane labeled by h, having polarization 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 polarization β and γ, respectively; the positive children are placed inside the former, their polarizations changed to δ, while the negative ones are placed inside the latter, their polarizations changed to ε. Any neutral membrane inside h is duplicated (together with the objects and membranes it contains) and placed inside both copies.

A *configuration* in a P system with active membranes is described by its membrane structure, together with its polarizations and the multisets of objects contained in its regions. The initial configuration is given by μ, all membranes having polarization 0 and the initial contents of the membranes being $w_1, \ldots, w_m$. A *computation step* changes the current configuration according to the following principles:

– Each object and membrane can be subject to at most one rule per computation step.
– The rules are applied in a *maximally parallel way*: each object which appears on the left-hand side of applicable communication rules must be subject to exactly one of them; the same holds for each membrane which can be involved in a communication or nonelementary division rule. The only objects and membranes which remain unchanged are those associated with no rule, or with unapplicable rules.
– When more than one rule can be applied to an object or membrane, the actual rule to be applied is nondeterministically chosen; hence, in general, multiple configurations can be reached from the current one.
– Every object which is sent out from the skin membrane cannot be brought in again.

A halting computation $\mathcal{C}$ of a P system Π is a finite sequence of configurations $(\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration of Π, every $\mathcal{C}_{i+1}$ can be reached from $\mathcal{C}_i$ according to the principles just described, and no further configuration can be reached from $\mathcal{C}_k$ (i.e., no rule can be applied). P systems might also perform non-halting computations; in this case, we have infinite sequences $\mathcal{C} = (\mathcal{C}_i : i \in \mathbb{N})$ of successive configurations.

We can use families of P systems with active membranes as language recognizers, thus allowing us to solve decision problems [9].

Definition 2. *A* recognizer P system with active membranes Π *has an alphabet containing two distinguished objects* yes *and* no*, used to signal acceptance and rejection respectively; every computation of Π is halting and exactly one object among* yes*,* no *is sent out from the skin membrane during the last step of each computation.*

In what follows we will only consider *confluent* recognizer P systems with active membranes, in which all computations starting from the initial configuration agree on the result.

Definition 3. *Let $L \subseteq \Sigma^\star$ be a language and let $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recognizer P systems. We say that $\boldsymbol{\Pi}$ decides L, in symbols $L(\boldsymbol{\Pi}) = L$, when for each $x \in \Sigma^\star$, the result of Π_x is acceptance iff $x \in L$.*

Usually some uniformity condition, inspired by those applied to families of Boolean circuits, is imposed on families of P systems. Two different definitions are used:

Definition 4. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* polynomially semi-uniform *when the mapping $x \mapsto \Pi_x$ can be computed in polynomial time, with respect to $|x|$, by a deterministic Turing machine.*

Definition 5. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* polynomially uniform *when there exist two polynomial-time Turing machines M_1 and M_2 such that, for each $n \in \mathbb{N}$ and each $x \in \Sigma^n$*

- *M_1, on input 1^n (the unary representation of the length of x), outputs the description of a P system Π_n with a distinguished input membrane;*
- *M_2, on input x, outputs a multiset w_x (an encoding of x);*
- *Π_x is Π_n with w_x added to the multiset located inside its input membrane.*

In other words, the P system Π_x associated with string x consists of two parts; one of them, Π_n, is common for all strings of length $|x| = n$ (in particular, the membrane structure and the set of rules fall into this category), and the other (the input multiset w_x for Π_n) is specific to x. The two parts are constructed independently and, only as the last step, w_x is inserted in Π_n.

In this paper we denote by $\mathbf{MC}_\mathcal{D}$ (resp., $\mathbf{MC}_\mathcal{D}^\star$) the class of languages that can be decided by uniform (resp., semi-uniform) families of confluent P systems of type $\mathcal{D}$ (e.g., $\mathcal{AM}$ denotes P systems with active membranes) without any restriction on time complexity. The space complexity of P systems [10] is defined as follows:

Definition 6. *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 $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a halting computation of Π, then the space required by $\mathcal{C}$ is defined as*

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

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

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

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

The space required by Π itself is then

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

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

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

Next, we recall the definition of (deterministic) register machines.

Definition 7. *A deterministic n-register machine is a construct $R = (n, I, m)$, where $n > 0$ is the number of registers, I is a finite sequence of instructions (program) bijectively labeled with the elements of the set $\{1, 2, \ldots, m\}$, 1 is the label of the first instruction to be executed, and m is the label of the last instruction of I. Registers contain non-negative integer values. The instructions of I have the following forms:*

- *"i: INC$(r), j$" with $i \in \{1, 2, \ldots, m\}$, $j \in \{1, 2, \ldots, m+1\}$ and $r \in \{1, 2, \ldots, n\}$. This instruction, labeled with i, increments the value contained in register r, and then jumps to instruction j.*
- *"i: DEC$(r), j, k$" with $i \in \{1, 2, \ldots, m\}$, $j, k \in \{1, 2, \ldots, m + 1\}$ and $r \in \{1, 2, \ldots, n\}$. If the value contained in register r is positive then decrement it and jump to instruction j. If the value of r is zero then jump to instruction k, without altering the contents of the register.*

Computations start by executing the first instruction of I (labeled with 1), and halt when the instruction currently executed tries to jump to label $m + 1$.

Formally, a *configuration* is a $(n{+}1)$-tuple whose components are the contents of the n registers, and the label of the next instruction of I to be executed. In the *initial* configuration this label is set to 1, whereas it is equal to $m{+}1$ in any *final*

configuration. A *computation* starts in the initial configuration and proceeds by performing computation steps, i.e., executing the current instruction, modifying accordingly the contents of the affected register and the pointer to the next instruction. A computation halts if it reaches a final configuration. The contents of the registers in the initial configuration are regarded as the input, and those in the final one as the output of the computation. A non-halting computation does not produce a result.

We will also use the operation of *tetration* (iterated exponentiation) to describe upper bounds on time and space complexity:

Definition 8. *For $n \in \mathbb{N}$, let tetration be defined by*

$$^{n}b = \begin{cases} 1 & \text{if } n = 0 \\ b^{(^{n-1}b)} & \text{if } n > 0 \end{cases}, \qquad \text{that is,} \qquad {}^{n}b = \underbrace{b^{b^{b^{\cdot^{\cdot^{\cdot^{b}}}}}}}_{n \ times}$$

We denote by **PTETRA** *the class of languages decidable by deterministic Turing machines in time $^{p(n)}2$ for some polynomial p.*

This class is trivially closed under union, intersection, complement and polynomial time reductions; furthermore, since a TM operating in space $f(n)$ can be simulated in time $2^{O(f(n))}$, **PTETRA** is also the class of languages decidable in *space* $^{p(n)}2$ for some polynomial p (we will make use of this characterization later). It is important to notice that **PTETRA**, despite being a very large class, does not contain all recursively enumerable (or even recursive) languages, as a consequence of the hierarchy theorems [3] for Turing machines.

3   Proof of Non-Universality

We begin by providing an upper bound on the size that can be reached by the membrane structure of a P system not using elementary division. If we ignore labels and polarizations, any membrane structure of size m is a subtree of the complete m-ary tree of depth $m - 1$.[1] For the sake of finding the upper bound, we can thus assume that the membrane structure we are considering has exactly this shape. A distinctive feature of this tree is *uniformity* [1], that is, the number of descendants of each node (hence the size and shape of the subtree rooted in it) only depends on its level. Since we only deal with uniform trees in this section, we denote them by $T(n_1, \ldots, n_{k-1})$ as in [1], where n_i is the number of nodes on level i (the 0-th level implicitly contains only the root node). The complete tree we just described is then denoted by $T(m, m^2, \ldots, m^{m-2}, m^{m-1})$.

Since nonelementary division rules separate the positively and negatively charged children membranes (explicitly listed on the left-hand side of the rules themselves) while duplicating the neutral ones, it should be clear that the number

---

[1] Smaller trees containing all subtrees of m nodes are known to exist, but none of them is substantially (i.e., exponentially) smaller than this one [1].

of newly created membranes can be maximized by using only one positive and one negative membrane, as follows:

$$\left[\,[\,]_{h_1}^{+}\,[\,]_{h_2}^{-}\,\right]_{h}^{\alpha} \to \left[\,[\,]_{h_1}^{\delta}\,\right]_{h}^{\beta}\,\left[\,[\,]_{h_2}^{\epsilon}\,\right]_{h}^{\gamma}$$

Suppose this rule is applied to a membrane h having $k \geq 2$ children (among these, a positively charged instance of h_1 and a negatively charged instance of h_2, while all the other children membranes are neutral). As a result, membrane h is doubled, and each of the two resulting instances of h possesses $k-1$ children (one of the two charged ones, and a copy of all the neutral ones). For the rest of this section, we ignore polarizations and labels, and assume that nonelementary division rules (all of the above form) are always applicable when needed, as this is the worst-case scenario. Under these assumptions, a membrane can divide as long as it has at least two children. Notice that communication rules play no role here, as they cannot increase the number of membranes.

The number of membranes created during the computation is maximized when division rules are applied in a bottom-up fashion, that is, by first applying all of them to membranes on the lowest possible level of the tree, until no further division is possible, and only then proceeding to the upper level. Indeed, so doing the membranes on the upper level will have more children, and so will be, in turn, susceptible to more division rules. Notice that membranes on the same level are, instead, independent as far as nonelementary division is concerned: we can thus apply division rules to all membranes of the current level simultaneously, in a maximally parallel way, in order to simplify the calculations.

According to the principles described thus far, each node on level $m-2$ of the tree

$$T(m, m^2, \ldots, m^{m-2}, m^{m-1}),$$

which is the lowest where nonelementary division can be applied, is doubled $m-1$ times by applying every possible division rules in a maximally parallel way; the first few steps are

$$T\big(m, m^2, \ldots, 2 \cdot m^{m-2}, 2 \cdot m^{m-2}(m-1)\big)$$
$$T\big(m, m^2, \ldots, 2^2 \cdot m^{m-2}, 2^2 \cdot m^{m-2}(m-2)\big)$$
$$T\big(m, m^2, \ldots, 2^3 \cdot m^{m-2}, 2^3 \cdot m^{m-2}(m-3)\big)$$

and the resulting tree is

$$T\big(m, m^2, \ldots, m^{m-3}, 2^{m-1} \cdot m^{m-2}, 2^{m-1} \cdot m^{m-2}\big).$$

Repeating the whole process at level $m-3$ yields

$$T\big(m, m^2, \ldots, m^{m-4}, 2^{2^{m-1} \cdot m - 1} \cdot m^{m-3}, 2^{2^{m-1} \cdot m - 1} \cdot m^{m-3}, 2^{2^{m-1} \cdot m - 1} \cdot m^{m-3}\big).$$

In general, the number of nodes of level $m-1-k$ of tree $T(m, m^2, \ldots, m^{m-1})$ after applying all division rules on the lowest k levels (excluding the leaves) is

$m^{m-1-k} \cdot N(k)$ where

$$N(k) = \begin{cases} 1 & \text{if } k = 0 \\ 2^{N(k-1)m-1} & \text{if } m-2 \geq k > 0. \end{cases}$$

After all possible division rules have been applied on level k, the number of nodes is also $m^{m-1-k} \cdot N(k)$ in all levels below it. Hence the final tree, obtained by applying all division rules on all $m-2$ levels below the root (in a bottom-up fashion), has a total of $1 + (m-1) \cdot m \cdot N(m-2)$ nodes, and it can be proved that this amount is $^{O(m^2)}2$. As a consequence, the following statement holds.

Lemma 1. *Let Π be a P system that does not use elementary division rules, with an initial membrane structure of size m. The maximum number of membranes of Π, in any of its computations, is bounded by $^{O(m^2)}2$.* □

Now suppose that only communication and nonelementary division rules are allowed. The only way to create new objects is, then, dividing a membrane, thus duplicating all objects inside it. An upper bound on the maximum number of objects which can be generated by a P system of this kind can be computed as follows.

Lemma 2. *Let Π be a P system using only communication and nonelementary division rules, with an initial membrane structure of size m, and k objects in its initial configuration. The maximum number of objects of Π, in any of its computations, is bounded by $^{O(m^2)}2 \cdot k$.*

Proof. Suppose that, before *each* application of a nonelementary division rule to a membrane, all objects are moved inside that membrane, so that they are doubled when it divides: this hypothetical situation is the worst case. Clearly, the number of applications of division rules which may occur during the whole computation is bounded by the final number of membranes, which is, in turn, bounded by $^{O(m^2)}2$ (Lemma 1). Then, the number of objects is bounded by $2^{(^{O(m^2)}2)} \cdot k$, that is, by $^{O(m^2)}2 \cdot k$. □

Lemmata 1 and 2 allow us to prove that recognizer P systems using only communication and nonelementary division rules are not universal, as they can be simulated by Turing machines operating in tetrational space.

Theorem 1. *Let $\mathcal{D}$ be the class of recognizer P systems using only communication and nonelementary division rules. Then $\mathbf{MC}^\star_\mathcal{D} \subseteq \mathbf{PTETRA}$.*

Proof. Let $L \in \mathbf{MC}^\star_\mathcal{D}$. Then L is decided by a semi-uniform family $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of P systems of type $\mathcal{D}$. Since there exists a polynomial-time Turing machine constructing each Π_x, there is a polynomial p such that each P system Π_x has a description of size at most $p(n)$, where $n = |x|$. But then both the size of the initial membrane structure and the initial number of objects in each P system of $\boldsymbol{\Pi}$ are bounded by $p(n)$. Hence, by Lemmata 1 and 2, the family $\boldsymbol{\Pi}$ requires $^{O(p(n)^2)}2 \cdot p(n)$ space; since a family of P systems can be simulated by a Turing machine using asymptotically the same space [12], the statement of the theorem follows. □

4 A Useful Gadget

The absence of object evolution rules (i.e., rules such as $[a \to w]_h^\alpha$ which transform an object into a multiset of objects without using the membrane structure) from the class of P systems we are considering makes the process of designing membrane algorithms a daunting task. In this section we introduce a gadget that can be adapted for several uses, in particular as a timer (to generate a specific object after a certain number of time steps) and as a generator of exponentially or even tetrationally many copies of an object (which is useful to increase the polynomial amount of objects available in the initial configuration), thus partially fulfilling the role of evolution rules. These features will be needed in Section 5 in order to simulate a register machine using tetrational space.

The gadget is a P system "module" $\Pi_{d,w}$ with a membrane structure having a tree representation with $w + d - 1$ nodes; $d - 1$ of them are arranged in a chain, while the remaining w are children of the lowest node in the chain, thus they correspond to the innermost membranes. The membranes are labeled by $h_1, \ldots, h_d$ according to the level of the tree they belong to. We give here just an informal description of the inner working of the gadget; the appendix contains a complete description, as well as a number of purely technical details that have to be addressed (for instance, a constant number of extra membranes per level must be added in order to provide a way of delaying the action of objects, without using object evolution rules).

Two objects named ℓ and r, initially located inside h_{d-1}, enter two of the membranes labeled by h_d and set their charge to positive and negative respectively; at this point, a nonelementary division rule can be applied to h_{d-1}, which splits in two copies containing all the neutral membranes h_d and one of the two charged membranes; the objects ℓ and r are also duplicated. Both copies of h_{d-1} are now in a configuration analogous to the initial one (only with one less child), and the process is repeated in parallel inside both of them. The procedure stops when each of the 2^{w-1} resulting copies of h_{d-1} has a unique child, at which point an object c is sent to h_{d-2} to signal that the operations on level $d-1$ have been completed; object c activates the objects ℓ and r of the level above, and the whole process is repeated there. When an object c finally reaches h_1 (the root of the membrane structure), another object e is produced, thus signaling the completion of the division procedure on all the levels.

Choosing $w = 3$ is sufficient to produce at least $^{d-2}2$ membranes on the level immediately below the root (and on every level below it). This requires at least $^{d-3}2$ steps of nonelementary divisions on that level: as a consequence, the object e is only produced after this amount of time steps have passed. By choosing an appropriate value of d, we can use $\Pi_{d,3}$ as a clock, when we need a certain object e to appear only after some other operations have been carried out.

Similarly, by adding a child membrane containing a copy of the object a to each membrane labeled by h_d in $\Pi_{d,3}$ we can obtain at least $^{d-2}2$ copies of a, located inside the elementary membranes. When the object e is produced, it can be used to release the copies of a (by changing the polarization of the membranes that contain them), which can then travel upwards and be sent out

from h_1. Notice that, by modifying $\Pi_{3,w}$ this way instead, we obtain *exactly* 2^{w-1} copies of object a.

5 Solving PTETRA Problems

We defined the complexity class **PTETRA** in terms of Turing machines, operating in time or space $^{p(n)}2$, but the class remains the same when register machines are used. Indeed, register machines can simulate TMs by using asymptotically the same space (understood as the number of bits needed to represent the values of the registers) by interpreting the tape of the TM as two binary integers corresponding to the portions of the tape on the left and on the right of the head, respectively [2]. A register machine recognizes a set of natural numbers by computing its characteristic function. Since strings may have meaningful leading 0s, before feeding them as input to the register machines we prefix them by 1 (this operation is, obviously, a bijection between $\{0,1\}^\star$ and $1\{0,1\}^\star$). By simulating these register machines via P systems, using only communication and nonelementary division rules, we can prove the reverse inclusion of Theorem 1.

It is known [11] that a register machine can be simulated by a uniform family of P systems using only communication rules as long as the values stored in the registers can be represented in unary using a polynomial number of symbols. Each register r is represented by a membrane having label r and containing a number of occurrences of object a equal to the value of the register; increment and decrement operations are performed by sending in or out another occurrence of a when the polarization of r is changed via a program-counter object (with a subscript denoting the current instruction).

In our case, however, the unary encoding of an input string $x \in 1\{0,1\}^n$ requires exponentially many objects; furthermore, tetrationally many auxiliary objects are required during the computation. We can solve both these problems by using the gadget described in the previous section, only requiring the addition of nonelementary division rules.

Let $L \in$ **PTETRA**, and let R be a register machine accepting all integers with binary encoding $1x$ for $x \in L$, and rejecting integers with binary encoding $1x$ for $x \notin L$; assume that R works in space $^{p(n)}2$ for some polynomial p. We simulate R via a uniform family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \{0,1\}^\star\}$.

The membrane structure of all systems Π_x with $|x| = n$ is the following one:

$$\mu_n = \left[\left[\mu_0' \cdots \mu_n'\right]_1^0 \left[\,\right]_2^0 \cdots \left[\,\right]_k^0 \mu'' \mu''' \left[\,\right]_z^0\right]_0^0$$

where k is the number of registers of R, 1 is the input register, μ'' is the membrane structure of a P system module sending out $^{p(n)}2$ copies of object a (thus providing the auxiliary objects needed to carry out the computation), μ''' is the membrane structure of a module generating object p_1 after $O(^{p(n)}2)$ steps, and z is an auxiliary membrane (used to simulate "waiting" without using object evolution rules [11]). Membrane μ_i' (for $0 \le i \le n$) sends out 2^i copies of object a inside the input membrane 1 if and only if is activated by an object y_i entering

its outermost membrane; this process is used to convert the binary input into unary notation. The membrane structure μ_n, together with the associated set of rules and initial objects, can be constructed from 1^n by a Turing machine operating in polynomial time.

The encoding of $x = x_{n-1} \cdots x_0$, to be placed inside the input membrane 1, is constructed as follows: for all $i \in \{0, \ldots, n-1\}$, object y_i is part of the input multiset iff $x_i = 1$; furthermore, y_n is also part of the input multiset. Clearly, this encoding can be computed from x by a Turing machine working in polynomial time; hence, the family of P systems $\boldsymbol{\Pi}$ is uniform.

Each P system Π_x of $\boldsymbol{\Pi}$ operates as follows. As described above, the object y_i enters the membrane structure μ_i' which activates and sends out 2^i copies of object a to membrane 1. When the process is completed for all y_i, the multiplicity of a inside membrane 1 is exactly the same as the number having binary representation $1x$. Simultaneously, inside μ'' we produce enough copies of a to carry out the simulation of the register machine, which are then sent out to the skin membrane one at a time. Finally, we use μ''' to delay the appearance of the program counter object p_1 until all the instances of object a (both those encoding the input and the auxiliary ones) have been generated and moved to their initial position (membrane 1 and the skin, respectively).

When p_1 is sent out to the skin membrane, the real simulation (which is described in detail in [11]) begins. The object p_i, where i is the currently simulated instruction, moves between regions and changes the polarizations of the membranes, thus activating rules that bring in or send out a copy of object a to one of the register membranes, i.e., performing increment and decrement operations. For instance, the following rules simulate the instruction "$i : \text{INC}(r), j$":

$$p_i \, [\,]_r^0 \rightarrow [p_i']_r^+ \qquad\qquad a \, [\,]_r^+ \rightarrow [a]_r^0 \qquad\qquad [p_i']_r^0 \rightarrow [\,]_r^0 \, p_j$$

Decrement instructions "$i : \text{DEC}(r), j, k$" are implemented similarly, only with the difference that the program counter object p_i must wait at least one step in order to let a copy of object a (that possibly occurs inside r) be sent out, thus changing the charge of membrane r; now p_i is sent out of r as p_j or p_k depending on whether an a was actually sent out (i.e., the value of register r was nonzero).

The computation halts when the object *yes* (resp., *no*) is sent out from the skin membrane of Π_x if the simulated register machines accepts (resp., rejects), that is, if $x \in L$ (resp., $x \notin L$). Thus, the family $\boldsymbol{\Pi}$ recognizes L.

Since L is an arbitrary **PTETRA** language, and given the space requirements of $\boldsymbol{\Pi}$ and the result of Theorem 1, we proved the following statement:

Theorem 2. *Let $\mathcal{D}$ be the class of P systems with active membranes using only communication and elementary division rules. Then* **PTETRA** $=$ $\mathbf{MC}_{\mathcal{D}} =$ $\mathbf{MC}_{\mathcal{D}}^{\star}$. $\qquad\qquad\square$

6 Conclusions

We proved that P systems with active membranes using only communication and nonelementary division rules are computationally weaker than Turing machines.

In particular, they characterize the class **PTETRA** of languages decided by Turing machines whose resources are bounded by $^{p(n)}2$ for some polynomial p. This statement is proved by extending a recently published simulation of register machines [11].

The fact that this kind of P systems, despite their non-universality, are able to decide such a large class of languages suggests some topics for future research. Are there any other classes of P systems (and, more generally, of classic or bio-inspired computing devices) that exhibit a similar behavior? If this is not the case, how can universal computing devices be restricted (in a different way than simply imposing a resource bound) or enriched in order to achieve it?

References

1. Chung, F.R.K., Graham, R.L., Coppersmith, D.: On trees containing all small trees. In Chartrand, G. (ed.) The Theory of Applications of Graphs, pp. 265–272. Wiley (1981)
2. Fischer, P.C., Meyer, A.R., Rosenberg, A.L.: Counter machines and counter languages. Mathematical Systems Theory 2(3), 265–283 (1968)
3. Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Transactions of the American Mathematical Society 117, 285–306 (1965)
4. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences, 1(61), 108–143 (2000).
5. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
6. Păun, G.: Active membranes. In Păun, G., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 282–301. Oxford University Press (2010)
7. Păun, G., Rozenberg, G.: A guide to membrane computing. Theoretical Computer Science 287, 73–100 (2002)
8. Păun, G., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
9. Pérez-Jiménez, M.J., Romero Jiménez, A., Sancho Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2, 265-285 (2003)
10. 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)
11. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes: Trading time for space. Natural Computing, to appear
12. Porreca, A.E., Mauri, G., Zandron, C.: Complexity classes for membrane systems. RAIRO Theoretical Informatics and Applications 40(2), 141–162 (2006)
13. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
14. The P Systems Webpage, `http://ppage.psystems.eu`

A Appendix

A.1 Upper Bound on the Number of Membranes

In Section 3 we claim that $1 + (m-1) \cdot m \cdot N(m-2) = {}^{O(m^2)}2$; this statement can be proved as follows. First of all, notice that for all k we have $N(k) \leq M(k)$ where

$$M(k) = \begin{cases} 1 & \text{if } k = 0 \\ 2^{M(k-1)m} & \text{if } m-2 \geq k > 0. \end{cases}$$

By induction we prove $M(k) \leq {}^{k(m+1)}2$. Indeed, $M(0) = 1 = {}^{0(m+1)}2$; furthermore

$$M(k+1) = 2^{M(k)m} \leq 2^{\left({}^{k(m+1)}2\right)m} \leq 2^{\left({}^{k(m+1)+m}2\right)} = {}^{k(m+1)+m+1}2 = {}^{(k+1)(m+1)}2$$

where the third step is justified by the fact that $({}^b2)^a \leq {}^{a+b}2$ (which holds for $a \geq 1$). Hence, we obtain $N(m-2) \leq M(m-2) \leq {}^{(m-1)(m+1)}2$. Then

$$\begin{aligned} 1 + (m-1) \cdot m \cdot N(m-2) &\leq 1 + (m-1) \cdot m \cdot {}^{(m-1)(m+1)}2 \\ &\leq 1 + {}^{(m-1)(m+1)+(m-1)m}2 \\ &\leq {}^{(m-1)(m+1)+(m-1)m+1}2 \\ &= {}^{O(m^2)}2 \end{aligned}$$

where the second and third steps are justified by $a \cdot {}^b2 \leq {}^{a+b}2$ and $a + {}^b2 \leq {}^{a+b}2$.

A.2 Details of the Gadget of Section 4

In Section 4 we gave an informal idea on how to generate a membrane structure having the shape of a tree with tetrationally many nodes. The technical details of the solution are discussed here.

The initial membrane structure of the P system module $\Pi_{d,w}$ is the following:

$$\overbrace{\Big[\Big[\cdots\Big[\Big[}^{d-1 \text{ levels}} \underbrace{\big[\,\big]_{h_d}\cdots\big[\,\big]_{h_d}}_{w \text{ copies}} \big[\big[\,\big]_\ell\big]_\ell\Big]_{h_{d-1}} \big[\big[\,\big]_\ell\big]_\ell\big[\,\big]_c\Big]_{h_{d-2}} \cdots \big[\big[\,\big]_\ell\big]_\ell\big[\,\big]_c\Big]_{h_2} \big[\,\big]_e\Big]_{h_1}$$

that is, besides the membranes labeled by h_i (described in Section 4) we have other $4(d-2)$ auxiliary ones:

- The nested membranes $\big[\big[\,\big]_\ell\big]_\ell$ occur inside each membrane $h_2, \ldots, h_{d-1}$; these membranes are traversed back and forth by the object ℓ when we need to "wait" for the nonelementary membrane division of h_i to occur.
- Membrane c occurs inside each membrane $h_2, \ldots, h_{d-2}$; these membranes host the object c, which is released whenever all possible nonelementary divisions of the membrane labeled by h_{i+1} have occurred.

– Membrane e, placed inside h_1, performs the same role as the membranes labeled by c, with the difference that object e is sent out when all possible nonelementary divisions have occurred on *each* level.

This membrane structure initally contains the following objects:

– Each membrane $h_1, \ldots, h_{d-2}$ contains the objects ℓ and r.
– Membranes $h_2, \ldots, h_{d-2}$ contain the object c', while h_{d-1} contains c and h_1 contains e'.

During the first computation step, all objects c' enter the membrane labeled by c near them by using the communication rule $c' \, [\,]_c^0 \to [c]_c^+$, thus also changing its charge to positive; the same happens to e', which uses the rule $e' \, [\,]_e^0 \to [e]_e^+$.

We can now discuss how the level-by-level division procedure is performed. Let $i \in \{3, \ldots, d-1\}$, and assume that all divisions of membranes labeled by $h_{i-1}, \ldots, h_{d-1}$ have been carried out and the object c has been released; let k be the number of children of membrane h_i at this moment. The configuration is then the following one (where only the relevant membranes and objects are displayed, and those not currently involved are omitted):

$$\mathcal{C}_0 = \ell r \Big[c \underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k \text{ copies}} \big[[\,]_\ell^0 \big]_\ell^0 \Big]_{h_i}^0 [c]_c^+$$

Object c exits h_i via $[c]_{h_i}^0 \to [\,]_{h_i}^+ \#$, where $\#$ denotes a "junk" object (i.e., it does not appear on the left-hand side of any rule; as such, it is irrelevant and omitted from the following configurations):

$$\mathcal{C}_1 = \ell r \Big[\underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k \text{ copies}} \big[[\,]_\ell^0 \big]_\ell^0 \Big]_{h_i}^+ [c]_c^+$$

When h_i is positive, object ℓ enters via $\ell \, [\,]_{h_i}^+ \to [\ell_1]_{h_i}^-$:

$$\mathcal{C}_2 = r \Big[\ell_1 \underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k \text{ copies}} \big[[\,]_\ell^0 \big]_\ell^0 \Big]_{h_i}^- [c]_c^+$$

When h_i is negative, object r can enter via $r \, [\,]_{h_i}^- \to [r_1]_{h_i}^+$ and, simultaneously, ℓ_1 enters one of the copies of h_{i+1} and sets its charge to positive, using the rule $\ell_1 \, [\,]_{h_{i+1}} \to [\ell_2]_{h_{i+1}}^+$:

$$\mathcal{C}_3 = \Big[r_1 [\ell_2]_{h_{i+1}}^+ \underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k-1 \text{ copies}} \big[[\,]_\ell^0 \big]_\ell^0 \Big]_{h_i}^+ [c]_c^+$$

The evolution of the computation from this point on depends on whether there exists another neutral membrane labeled by h_{i+1} (i.e., on whether $k > 1$). First,

assume that this is indeed the case. While ℓ_2 exists its membrane by using the rule $[\ell_2]_{h_{i+1}}^+ \to [\,]_{h_{i+1}}^+ \ell_3$, object r_1 enters another one via $r_1 [\,]_{h_{i+1}}^0 \to [r_2]_{h_{i+1}}^0$:

$$\mathcal{C}_4 = \Big[\ell_3 [\,]_{h_{i+1}}^+ [r_2]_{h_{i+1}}^0 \underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k-2 \text{ copies}} \big[[\,]_\ell^0\big]_\ell^0 \Big]_{h_i}^+ [c]_c^+$$

Now r_2 exits h_{i+1}, via $[r_2]_{h_{i+1}}^0 \to [\,]_{h_{i+1}}^+ r_3$, setting its charge to negative; in the mean time, ℓ_3 enters the outermost membrane labeled by ℓ using $\ell_3 [\,]_\ell^0 \to [\ell_4]_\ell^0$:

$$\mathcal{C}_5 = \Big[r_3 [\,]_{h_{i+1}}^+ [\,]_{h_{i+1}}^- \underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k-2 \text{ copies}} \big[\ell_4 [\,]_\ell^0\big]_\ell^0 \Big]_{h_i}^+ [c]_c^+$$

Now h_i contains both a negative and a positive membrane, and it can divide using $\big[[\,]_{h_{i+1}}^+ [\,]_{h_{i+1}}^-\big]_{h_i}^+ \to \big[[\,]_{h_{i+1}}^0\big]_{h_i}^0 \big[[\,]_{h_{i+1}}^0\big]_{h_i}^0$; simultaneously, ℓ_4 moves one membrane deeper via $\ell_4 [\,]_\ell^0 \to [\ell_5]_\ell^0$.

$$\mathcal{C}_6 = \Big[r_3 \overbrace{\underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k-1 \text{ copies}} \big[[\ell_5]_\ell^0\big]_\ell^0}^{2 \text{ copies}} \Big]_{h_i}^0 [c]_c^+$$

In two steps, via $[\ell_5]_\ell^0 \to [\,]_\ell^0 \ell_6$ and $[\ell_6]_\ell^0 \to [\,]_\ell^0 \ell_7$, the two instances of ℓ_7 are moved back to h_i:

$$\mathcal{C}_8 = \Big[\ell_7 r_3 \overbrace{\underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k-1 \text{ copies}} \big[[\,]_\ell^0\big]_\ell^0}^{2 \text{ copies}} \Big]_{h_i}^0 [c]_c^+$$

The objects ℓ_7 now exit h_i via $[\ell_7]_{h_i}^0 \to [\,]_{h_i}^- \ell_8$, followed by r_3 via the rule $[r_3]_{h_i}^- \to [\,]_{h_i}^+ r_4$. Hence, in two steps we obtain

$$\mathcal{C}_{10} = \ell_8 r_4 \Big[\overbrace{\underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k-1 \text{ copies}} \big[[\,]_\ell^0\big]_\ell^0}^{2 \text{ copies}} \Big]_{h_i}^+ [c]_c^+$$

Now ℓ_8 goes back to h_i via $\ell_8 [\,]_{h_i}^+ \to [\ell_1]_{h_i}^-$; during the next step, r_4 follows via $r_4 [\,]_{h_i}^- \to [r_1]_{h_i}^+$ and ℓ_1 uses the rule $\ell_1 [\,]_{h_{i+1}} \to [\ell_2]_{h_{i+1}}^+$ as described earlier:

$$\mathcal{C}_{12} = \Big[r_1 [\ell_2]_{h_{i+1}}^+ \overbrace{\underbrace{[\,]_{h_{i+1}}^0 \cdots [\,]_{h_{i+1}}^0}_{k-2 \text{ copies}} \big[[\,]_\ell^0\big]_\ell^0}^{2 \text{ copies}} \Big]_{h_i}^+ [c]_c^+$$

Notice that the two instances of ℓ_8 and r_4 do not necessarily re-enter the same membrane h_i from which they came, due to nondeterminism; however, this makes no real difference as the two copies of h_i are identical, and the rules involving the two objects are applied in parallel.

Configuration $\mathcal{C}_{12}$ is completely analogous to $\mathcal{C}_3$, only with *two* occurrences of h_i, each one containing *one less* occurrence of h_{i+1}. Hence, we have entered a cycle which repeats in parallel the whole process of division of h_i, until each of these membranes only remains with one child. When this event occurs, the configuration becomes

$$\mathcal{C}_3' = \Big[r_1 \big[\ell_2\big]_{h_{i+1}}^{+} \overbrace{\big[\,[\,]_\ell^0\big]_\ell^0\big]^{+}}^{2^{k-1}\text{ copies}} \Big]_{h_i}^{+} [c]_c^+$$

Object r_1 has no neutral membrane h_{i+1} to enter, and remains stuck. The other object exits h_{i+1} and travels through the two membranes labelled by ℓ, but no division occurs (since we do not have negative membranes inside h_i); after five steps we obtain

$$\mathcal{C}_8' = \Big[\ell_7 r_1 \big[\,\big]_{h_{i+1}}^{+} \overbrace{\big[\,[\,]_\ell^0\big]_\ell^0\big]^{+}}^{2^{k-1}\text{ copies}} \Big]_{h_i}^{+} [c]_c^+$$

By noticing the positive (instead of neutral) charge of h_i, we may deduce that no division occurred and, by construction, that each copy of h_i only has one children labeled by h_{i+1}. The objects ℓ_7 are sent out of h_i and renamed to ℓ_9 via $[\ell_7]_{h_i}^{+} \rightarrow [\,]_{h_i}^{0} \ell_9$; this rule also sets h_i to neutral.

$$\mathcal{C}_9' = \ell_9 \Big[r_1 \big[\,\big]_{h_{i+1}}^{+} \overbrace{\big[\,[\,]_\ell^0\big]_\ell^0\big]^{+}}^{2^{k-1}\text{ copies}} \Big]_{h_i}^{+} [c]_c^+$$

One of the instances of ℓ_9 enters membrane c via $\ell_9\,[\,]_c^{+} \rightarrow [\ell_{10}]_c^0$; by setting its charge to neutral, it blocks all the other copies of ℓ_9 forever. Object c is now free to exit to membrane h_{i-1} via $[c]_c^0 \rightarrow [\,]_c^0\,c$.

During the next step, object c is sent out again via $[c]_{h_{i-1}}^0 \rightarrow [\,]_{h_{i-1}}^{+} \#$, thus changing the polarization to positive and activating the objects ℓ and r one level above (see configurations $\mathcal{C}_0$ and $\mathcal{C}_1$). The whole process of division is then restarted, this time involving membrane h_{i-1}.

The division of membrane h_{d-1}, the first to which the procedure is applied, is started by the object c located inside it (recall that h_{d-1} contains this object instead of c') by using the rule $[c]_{h_{d-1}}^0 \rightarrow [\,]_{h_{d-1}}^{+} \#$, as it happens on the other levels. On the other hand, on the outermost level (membrane h_1) it is the object e to be released, instead of c, in order to signal that *every* level has completed the division process.

By considering only the subtree consisting of the membranes labeled by $h_1, \ldots, h_d$, and ignoring the auxiliary ones (ℓ and c), we obtain a tree structure usable for constructing the gadget of Section 4.