

About P Systems with Symport/Antiport

Pierluigi FRISCO

Leiden Institute for Advanced Computer Science – LIACS
Leiden University, Niels Bohrweg 1
2333 CA Leiden, The Netherlands
E-mail: pier@liacs.nl

Abstract. It is proved that four membranes suffice to P systems with minimal symport/antiport to generate all recursively enumerable sets of numbers.

It is also proved that P systems with symport/antiport without maximal parallelism are equivalent to partially blind counter automata.

1 Introduction

In recent years observations of cells and their biochemical processes have been of inspiration for the creation of theoretical computational devices. One of these models looks at the structure of a cell as a set of (nested) compartments delimited by membranes. The chemicals present in a cell in each of its membrane compartments are seen as objects interacting, changing (evolving), and passing to other compartments. *Membrane systems* (also known as *P systems*) were introduced in [11] and since then they were the object of intense investigation (the latest news about P systems are present at <http://psystems.disco.unimib.it/>). A book [12] covering the subject of membrane computing has recently been published.

One of the most elegant variants of the general model was introduced in [10] under the name of membrane systems *with symport/antiport*. This variant models in a straightforward way the synchronized movement of chemicals present in a cell: specific groups of objects may pass together through a membrane either in the same or in opposite direction. In the former case we refer to *symport*, in the latter to *antiport*. There is no modification (evolution) of any sort of the objects (as present in other variants), communication is the only driving power of these systems.

Various variants of these systems compute all recursively enumerable sets of numbers; the result has been first obtained in [10], and then improved in complexity (reducing the number of membranes and the number of objects that are transported at the same time) in [7, 8, 4].

In [1] the authors show that P systems with symport/antiport passing at most one object per time can generate any recursively enumerable set of numbers; such systems have nine membranes. This result is improved in [6] reducing the number of membrane to six; later in [2] this result is further improved reducing the number of membrane to five. In Section 3 we prove that four membrane suffice for this variant of P systems to generate any recursively enumerable set of numbers.

In the area of P systems the lack of a global clock (or of synchronization) has not been broadly studied. The first research in this direction was presented in [13] where P systems with bi-stable catalyst are studied, and later in [3] where conformon-P systems are considered. In [12] the study of P systems without maximal parallelism is cited as an open problem. In Section 4 we study P systems with symport/antiport without maximal parallelism. We prove that the generative power of these systems is at most the one of partially blind counter automata.

2 Basic Definitions

We assume the reader to have familiarity with basic concepts of formal language theory [5], and in particular with the topic of membrane systems with symport/antiport [10, 7, 8], and [12, Chapter 4]. In this section we recall particular aspects relevant to our presentation.

We use $\mathbb{N}\cdot\text{RE}$ to denote the family of recursively enumerable sets of natural numbers (where $0 \in \mathbb{N}$).

Let V be a finite set of objects. With V^* we indicate the free monoid generated by V with the operation of concatenation; λ indicates the empty word.

A *multiset* (over V) is a function $M : V \rightarrow \mathbb{N} \cup \{+\infty\}$; for $a \in V$, $M(a)$ defines the *multiplicity* of a in the multiset M . We say that an element a of a multiset M has *infinite multiplicity* if $M(a) = +\infty$. In case the multiplicity of an element of a multiset is 1 we indicate just the element.

The *support* of a multiset M is the set $\text{supp}(M) = \{a \in V \mid M(a) > 0\}$. The *size* of a multiset is defined by the function $|\cdot| : (V \rightarrow \mathbb{N} \cup \{+\infty\}) \rightarrow \mathbb{N} \cup \{+\infty\}$, where for M multiset over V , $|M| = \sum_{a \in \text{supp}(M)} M(a)$. The symbol ϕ indicates the *empty multiset*, that is, the multiset whose support is the empty set.

Let $M_1, M_2 : V \rightarrow \mathbb{N} \cup \{+\infty\}$ be two multisets. The *union* of M_1 and M_2 is the multiset $M_1 \cup M_2 : V \rightarrow \mathbb{N} \cup \{+\infty\}$ defined by $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$, for all $a \in V$. The *difference* $M_1 - M_2$ is here defined only when M_2 is included in M_1 (which means that $M_1(a) \geq M_2(a)$ for all $a \in V$) and it is the multiset $M_1 - M_2 : V \rightarrow \mathbb{N} \cup \{+\infty\}$ given by $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$. Of course, if $M_1(a) = +\infty$ and $M_2(a)$ is finite, then $M_1(a) - M_2(a) = +\infty$. If $M_2(a) = +\infty$, then $M_1(a) - M_2(a) = 0$.

A *membrane system with symport/antiport* with maximal parallelism of *degree* m , $m \geq 1$, is a construct $\Pi = (V, \mu, L_0, L_1, \dots, L_m, R_1, \dots, R_m, \text{fin})$. A *membrane system with symport/antiport* without maximal parallelism of *degree* m , $m \geq 1$, is a construct $\Pi_{\neq} = (V, \mu, L_0, L_1, \dots, L_m, R_1, \dots, R_m, \text{ack}, \text{fin})$. The set V is a finite set of objects; $\mu = (N, E)$ is a *tree* underlying Π . The set $N \subset \mathbb{N}$ contains *vertices*, for simplicity we assume $N = \{0, 1, \dots, m\}$ where the vertex 0 is the *root* of the tree. Each vertex in N except the root defines a *membrane compartment* (in this paper referred simply as *membrane*) of the system Π ; the root 0 defines the *environment*; $\text{fin} \in N \setminus \{0\}$ is a leaf, that is a vertex with no children, which defines the *final membrane*; $\text{ack} \in N \setminus \{0\}$ is a distinguished membrane called *acknowledgment membrane*. In this paper we consider only P systems where the root has only one child; this last is called *skin membrane*. The set $E \subseteq N \times N$ defines directed *edges* between vertices. The set E is the ‘father of’ relation present in μ equivalent to the nesting of membranes normally used in the literature of P systems. The multisets L_0, L_1, \dots, L_m over V define the initial multisets of objects with the peculiarity that all the objects in L_0 have infinite multiplicity while the ones in L_1, \dots, L_m do not. The sets R_1, \dots, R_m contain a finite number of *rules* of the form: $(v, \text{in}), (v, \text{out})$ (called *symport*

rules), or $(v, \text{in}; w, \text{out})$ (called *antiport* rules), with v, w nonempty multisets over V with a finite support. Thus the symports (ϕ, in) and (ϕ, out) and the antiports $(a, \text{in}; b, \text{out})$ with $a = \phi$ or $b = \phi$ are not allowed.

A *configuration* of a membrane system with symport/antiport of degree m is given by the $(m + 1)$ -tuple $(M_0 - L_0, M_1, \dots, M_m)$ of multisets over V associated to the root and the membranes $\{1, \dots, m\}$, respectively. Note that the configuration does not record the objects in the environment that occur with infinite multiplicity as they are invariant to any configuration. The $(m + 1)$ -tuple (ϕ, L_1, \dots, L_m) is called *initial configuration*. If the system does not have maximal parallelism, then $L_{ack} = \emptyset$ in the initial configuration. For two configurations $(M_0 - L_0, M_1, \dots, M_m), (M'_0 - L_0, M'_1, \dots, M'_m)$ of Π we write $(M_0 - L_0, M_1, \dots, M_m) \Rightarrow (M'_0 - L_0, M'_1, \dots, M'_m)$ indicating a *transition* from $(M_0 - L_0, M_1, \dots, M_m)$ to $(M'_0 - L_0, M'_1, \dots, M'_m)$ that is the application of a multiset of rules associated to each membrane. The system is with *maximal parallelism* if the largest multiset of applicable rules is considered for each transition of the system. If more than one multiset of rules can be applied, then exactly one of them is nondeterministically chosen. All rules present in this multiset are applied in parallel. The system is *without maximal parallelism* if any multiset (obviously except the empty one) of applicable rules can be considered for each transition of the system. The reflexive and transitive closure of \Rightarrow is indicated by \Rightarrow^* .

The rules R_q associated to a membrane $q \in N \setminus \{0\}$ can change the multisets M_q and M_p of p father of q in μ .

- A multiset v included in M_p may be subtracted from M_p and may be united to M_q , if the symport rule (v, in) is present in R_q . In this case the multisets change from M_p and M_q to $M'_p = M_p - v$ and $M'_q = M_q \cup v$, respectively.
- A multiset v included in M_q may be subtracted from M_q and united to M_p if the symport rule (v, out) is present in R_q . The multisets change from M_p and M_q to $M'_p = M_p \cup v$ and $M'_q = M_q - v$, respectively.
- A multiset v included in M_p may be united to M_q while, at the same time, a multiset w included in M_q may be united to M_p if the antiport rule $(v, \text{in}; w, \text{out})$ is associated to membrane q . In this case the multisets of objects change from M_p and M_q to $M'_p = (M_p - v) \cup w$ and $M'_q = (M_q - w) \cup v$, respectively.

In general, if a multiset v is subtracted from M_p and united to M_q we say that v *passes* from membrane p to membrane q .

A *computation* is a finite sequence of transitions between configurations of a system Π starting from the initial configuration (ϕ, L_1, \dots, L_m) . If the system has maximal parallelism, then the result of a computation is given by the multiset of objects present in membrane *fin* when the computations *halts* (that is, when the multiset of applicable rules has empty support). If the system does not have maximal parallelism the result of a computation is given by the multiset of objects present in membrane *fin* when an object is in membrane *ack*.

The set $N(\Pi)$ denotes the set of numbers computed by a P system Π with symport/antiport with maximal parallelism; the set $N(\Pi_{\neq})$ denotes the set of numbers computed by a P system Π_{\neq} with symport/antiport without maximal parallelism.

The *weight* of a rule is given by $|v|$ in case of a symport (v, in) or (v, out) and by $\max(|v|, |w|)$ in case of an antiport $(v, \text{in}; w, \text{out})$.

The family of all sets $N(\Pi)$ computed by P systems with symport/antiport with maximal parallelism of degree at most m , using symports of weight at most x and antiports of weight at most y is denoted by $\mathbb{N}\text{-PP}_m(\text{sym}_x, \text{anti}_y)$. The family of all sets $N(\Pi_{\neq})$ computed by P systems with symport/antiport without maximal parallelism of degree at most m , using symports of weight at most x and antiports of weight at most y is denoted by $\mathbb{N}\text{-P}_{\neq}P_m(\text{sym}_x, \text{anti}_y)$. When m , x or y is not bounded it is replaced with $*$.

Counter automata were introduced by M.L. Minsky in [9] as finite state devices with a read-only input tape and equipped with additional external storage in the form of one or more counters. Each of these counters has an unbounded capacity recording a natural number. Simple operations can be performed on the counters: increment of one unit, decrement of one unit (when positive), and test for zero, while the input tape is scanned one cell per time. After each of these operations the automaton may change state. It is shown in [9] that counter automata with an input tape can simulate any Turing machine when equipped with (at least) two counters.

It is possible to consider counter automata without input tape but with an *output counter*. Reading input symbols is syntactically replaced by adding to the distinguished output counter; the output counter is never decremented nor tested for zero. Formally a counter automaton with n counters ($n \in \mathbb{N}$) with output counter is defined as $M = (S, C, R, s_0, f)$, where S is a finite set of *states*, $s_0, f \in S$ are respectively called the *initial* and *final* state; C is the set of counters; R is the set of *instructions* of the form $(s \rightarrow r, \iota)$, where $s \rightarrow r$ indicates the state change with $s, r \in S, s \neq f$, and ι is a counter operation, with $\iota \in \{A_+, A_-, A_{=0}, \epsilon\}$ for some $A \in C$.

The effect of each kind of instruction in R is described below:

- $(s \rightarrow r, A_+)$: if the automaton is in state s , it changes state to r , incrementing the content of counter A by one unit;
- $(s \rightarrow r, A_-)$: if the automaton is in state s and the content of counter A is bigger than 0, then it decreases the counter by one unit and changes state to r ;
- $(s \rightarrow r, A_{=0})$: if the automaton is in state s and the counter A has 0 as content, then it changes state to r ;
- $(s \rightarrow r, \epsilon)$: if the automaton is in state s , then it changes state to r .

The configuration of a counter automaton M with $n + 1$ counters with output counter is given by (s, x_0, \dots, x_n) , where $s \in S$ and $x_0, x_1, \dots, x_n \in \mathbb{N}$. The number x_0 indicates the value of the output counter, while the other elements indicate the value of the rest of the counters.

Given two configurations $(s, x_0, x_1, \dots, x_n), (r, y_0, y_1, \dots, y_n)$ of a counter automaton with output counter we define a *computational step* as the relation of configurations given by $(s, x_0, x_1, \dots, x_n) \vdash (r, y_0, y_1, \dots, y_n)$ as follows.

For instructions $(s \rightarrow r, \iota) \in R$ we have:

- if $\iota = A_+$, then $y_A = x_A + 1$ and $y_j = x_j$ for $1 \leq j \leq n, j \neq A$;
- if $\iota = A_-$ and $x_A > 0$, then $y_A = x_A - 1$ and $y_j = x_j$ for $1 \leq j \leq n, j \neq A$;
- if $\iota = A_{=0}$ and $x_A = 0$, then $y_j = x_j$ for $1 \leq j \leq n$;
- if $\iota = \epsilon$, then $y_j = x_j$ for $1 \leq j \leq n$.

The reflexive and transitive closure of \vdash is indicated as \vdash^* .

A *computation* is a finite sequence of computational steps of a counter automaton M starting from the initial configuration $(s_0, 1, 0, \dots, 0)$. If we consider an automaton M with output counter and $(s_0, 0, 0, \dots, 0) \vdash^* (f, y_0, y_1, \dots, y_n)$, then we say that M *accepts* the number y_0 (where 0 is the index of the output counter).

The language accepted by M is defined as $L(M) = \{y_0 \in \mathbb{N} \mid (s, 0, 0, \dots, 0) \vdash^* (f, y_0, y_1, \dots, y_n)\}$, where y_0 is the content of the output counter.

For every counter automaton it is possible to create another one accepting the same set of numbers, and having all counters empty (with the obvious exception of the output counter) in the final state. We will assume this normal form in the sequel.

Partially blind counter automata were introduced by M.L. Minsky also in [9] as counter automata without test on zero. In case the automaton tries to subtract from a counter having value zero it stops in a non final state. In [9] partially blind counter automata are proved to be strictly less powerful than counter automata.

A partially blind counter automaton with output counter M_{pb} is defined as a counter automaton M with output counter lacking of instructions of the kind $(s \rightarrow r, A_{=0})$.

3 Universality with Four Membranes

As already said in Section 1, P systems with symport/antiport with maximal parallelism and weight 1 for both symports and antiports have been studied by several authors. The obtained results show that nine, six, and then five membrane were sufficient to generate all recursively enumerable sets of numbers. In the following we prove that four membranes suffice for these systems to generate all recursively enumerable sets of numbers.

Theorem 3.1 $\mathbb{N}\cdot\text{RE} = \mathbb{N}\cdot\text{PP}_4(\text{sym}_1, \text{anti}_1)$.

Proof. We consider the inclusion $\mathbb{N}\cdot\text{RE} \subseteq \mathbb{N}\cdot\text{PP}_4(\text{sym}_1, \text{anti}_1)$, proving it constructing a P system with symport/antiport with maximal parallelism simulating a counter automaton with output counter. The inclusion $\mathbb{N}\cdot\text{PP}_4(\text{sym}_1, \text{anti}_1) \subseteq \mathbb{N}\cdot\text{RE}$ can be shown constructing a Turing machine simulating a P system with symport/antiport.

Let $M = (S, C, R, s_0, f)$ be a counter automaton as specified in Section 2. The P system with symport/antiport is defined by

$$\Pi = (V, \mu, L_0, L_1, L_2, L_3, L_4, R_1, R_2, R_3, R_4, 4),$$

where:

$$V = S \cup W \cup \{b_1, b_2, e, e', e'', e''', e_1, e_2, a_1, a_2, a_3, a_4, \dagger, k, k'_1, k'_2, k'_3, k'_4, k'_5, \infty_1, \infty_2\};$$

$$W = \{c_r \mid c \in C, r \in S \text{ and } (s \rightarrow r, c_+) \in S\} \cup$$

$$\{c'_r, d_{c,r} \mid c \in C, r \in S \text{ and } (s \rightarrow r, c_-) \in S\} \cup$$

$$\{c''_r, d''_{c,r} \mid c \in C, r \in S \text{ and } (s \rightarrow r, c_{=0}) \in S\};$$

$$\mu = (\{0, 1, 2, 3, 4\}, \{(i, i+1), (i+1, i) \mid 0 \leq i \leq 3\});$$

$$L_0 = S \cup W \cup \{e, e'', e'''\};$$

$$L_1 = b_1 b_2 \dagger;$$

$$L_2 = a_1 e_1 e_2 k k'_1 k'_2 k'_4 k'_5 \infty_1;$$

$$L_3 = a_2 a_3 k'_3 \infty_2;$$

$$L_4 = a_4 e';$$

$$R_i = R_3^\infty \cup R'_i \cup R''_i \cup R'''_i, \quad 1 \leq i \leq 4.$$

Each computation of M can be simulated by Π . The simulation can be divided in three main logical phases:

1. **initialization:** using the rules present in R'_i , $1 \leq i \leq 4$, an unbounded number of objects of the kind $c_r, d_{c,r}$ and $d''_{c,r}$ present in the environment may pass to other regions of the P system;
2. **simulation:** using the rules present in R''_i , $1 \leq i \leq 4$, the instruction of the counter automaton M are simulated. If M halts in a non final state, then the computation of Π never stops. If M halts in a final state, then Π can reach the third phase;
3. **termination:** using the rules present in R'''_i , $1 \leq i \leq 4$, the P system Π moves into the final membrane the objects defining the result of the computation and moves out from this membrane the objects that are irrelevant for the result of the computation. This process eventually leads to the end of the computation of Π .

The set $R_3^\infty = \{(\infty_1, \text{in}; \infty_2, \text{out}), (\infty_2, \text{in}; \infty, \text{out}), (\dagger, \text{in}), (\dagger, \text{out})\}$ contains rules forcing the system to an endless computation. The rules involving ∞_1 and ∞_2 are used during the initialization and simulation phase and part of the termination phase. The movement of the objects ∞_1 and ∞_2 between membrane 2 and membrane 3 can be interrupted only during the termination phase. These rules force the P system to an endless computation unless the counter automaton has been correctly simulated. The rules involving \dagger can start to be applied during the initialization or simulation phase, in this case the system never renders a result even if the termination phase is reached. The rules involving \dagger are present to avoid that the P system reaches a halting configuration after a computation that is not a simulation of the counter automaton. As indicated in the following, the rules involving \dagger can also be applied even if the P system was properly simulating the counter automaton.

The rules associated to the initialization phase are:

$$\begin{aligned}
R'_1 &= \{(X, \text{in}; b_1, \text{out}) \mid X \in \{r, d_{c,r}, d'_{c,r} \mid c \in C, r \in S\}\} \cup \{(b_1, \text{in}), (p_0, \text{in}; a_4, \text{out})\}; \\
R'_2 &= \{(X, \text{in}; b_2, \text{out}) \mid X \in \{r, d_{c,r}, d'_{c,r} \mid c \in C, r \in S\}\} \cup \\
&\quad \{(b_2, \text{in}), (a_4, \text{out}), (b_1, \text{in}; a_1, \text{out}), (a_1, \text{in}; a_2, \text{out}), (\dagger, \text{in}; b_2, \text{out}), (\dagger, \text{in}; a_2, \text{out})\}; \\
R'_3 &= \{(d, \text{in}; a_3, \text{out}) \mid d \in \{d_{c,r}, d'_{c,r} \mid c \in C, r \in S\}\} \cup \\
&\quad \{(a_3, \text{in}), (b_2, \text{in}; a_2, \text{out}), (b_1, \text{in}; a_4, \text{out})\}; \\
R'_4 &= \{(d, \text{in}; a_4, \text{out}) \mid d \in \{d_{c,r}, d'_{c,r} \mid c \in C, r \in S\}\} \cup \\
&\quad \{(a_4, \text{in}), (b_2, \text{in}; a_4, \text{out}), (a_3, \text{in}; b_2, \text{out})\}.
\end{aligned}$$

In this phase an unbounded number of objects $q \in S \subset E$ can pass from the environment to membrane 2 and an unbounded number of objects $d_{c,r}, d'_{c,r} \in E, c \in C, r \in S$ can pass from the environment to membrane 4. This process starts with the parallel application of one of the rules $(X, \text{in}; b_1, \text{out}) \in R'_1$ and the rule $(b_2, \text{in}) \in R'_2$. After this the rules $(b_1, \text{in}) \in R'_1$ and one of the rules $(X, \text{in}; b_2, \text{out}) \in R'_2$ can be applied. The configuration of the system is almost similar to the initial one: the only difference is that an object of the kind $r, d_{c,r}$ or $d'_{c,r}$ is in membrane 2. The process of moving this kind of objects from the environment to membrane 2 can be indeed repeated as b_1 and b_2 are in membrane 1 as in the initial configuration. Following a similar process, an object of the kind $d_{c,r}$ or $d'_{c,r}$ present in membrane 2 can pass to membrane 4. This is performed by the application of one of the rules $(d, \text{in}; a_3, \text{out}) \in R'_3$ followed by the parallel application of $(d, \text{in}; a_4, \text{out}) \in R'_4$ and $(a_3, \text{in}) \in R'_3$. Once in membrane 3 the object a_4 can pass to membrane 4 by the application of the rule $(a_4, \text{in}) \in R'_4$.

When the object b_2 (or one of the objects $d'_{c,r}$) is in membrane 2 the object \dagger can pass from membrane 1 to membrane 2 by the application of the rule $(\dagger, \text{in}; b_2, \text{out}) \in R'_2$ (or $(\dagger, \text{in}; d'_{c,r}, \text{out}) \in R''_2$, see the following of the proof). If this happens, then the P system will never render a result. The rule $(\dagger, \text{in}; b_2, \text{out})$ is present because, as we will see later in the proof, the objects of the kind $r, d_{c,r}$ and $d'_{c,r}$ are used (in phase 2) by the P system to simulate some specific rules of the counter automaton. Once in membrane 1 these symbols can pass to the environment by the applications of rules in R'_1 starting in this way the simulation of an instruction of the counter automaton. This could lead the P system to reach an halting configuration after a computation that was not a simulation of the counter automaton. To avoid this, if in the initialization phase an object of the kind $r, d_{c,r}$ and $d'_{c,r}$ passes from membrane 1 to the environment when b_2 is in membrane 2, then the rule $(\dagger, \text{in}; b_2, \text{out})$ has to be applied.

Also the rule $(b_2, \text{in}; a_2, \text{out}) \in R'_3$ can be applied when the object b_2 is in membrane 2. Also in this case the computation of the P system will never render a result as the rule $(\dagger, \text{in}; a_2, \text{out}) \in R'_2$ is then applied.

When in membrane 1 the object b_1 can pass to membrane 2 by the application of the rule $(b_1, \text{in}; a_1, \text{out}) \in R'_2$; parallel to this rule also the rule $(b_2, \text{in}) \in R'_2$ is applied. As the object b_1 is no longer in membrane 1 the rules $(X, \text{in}; b_1, \text{out}) \in R'_1$ cannot be applied: the passage of objects from the environment to other membranes stops. In the rest of this phases other objects pass to one membrane to another such that the systems can eventually start the simulation phase.

If the rule $(b_2, \text{in}; a_2, \text{out}) \in R'_3$ is applied when a_1 is in membrane 1, then the rules $(b_2, \text{in}; a_4, \text{out}) \in R'_4$ and $(a_1, \text{in}; a_2, \text{out}) \in R'_2$ can be applied in parallel and then the rules $(b_1, \text{in}; a_4, \text{out}) \in R'_3$ and $(a_3, \text{in}; b_2, \text{out}) \in R'_4$ can also be applied in parallel. When a_4 is in membrane 3 and b_1 in membrane 4, also the rule $(a_4, \text{in}) \in R'_4$ can be applied. In this case the system never reaches the simulation phase. On the other hand, once in membrane 2 the object a_4 can pass to membrane 1 by the application of $(a_4, \text{out}) \in R'_2$, then the rule $(s_0, \text{in}; a_4, \text{out}) \in R'_1$ is applied. In this way one occurrence of the object s_0 , related to the initial state of the simulated automaton, passes from the environment to membrane 1. The initialization phase ends and the simulation phase starts.

If no rule involving \dagger has been used during the initialization phase, then the simulation phase starts with the object a_4 in the environment, the objects $a_2 s_0$ and \dagger in membrane 1; with some objects $r \in S \subset E$, either ∞_1 or ∞_2 and $a_1 e_1 e_2 k k'_1 k'_2 k'_4 k'_5$ in membrane 2, with either ∞_1 or ∞_2 and $b_1 b_2 k'_3$ in membrane 3; and with $a_3 e'$ and some objects of the kind $d_{c,r}, d'_{c,r}$ in membrane 4.

The rules associated to the simulation phase are:

$$\begin{aligned}
R''_1 &= \{(r, \text{in}; s, \text{out}) \mid s, r \in S, (s \rightarrow r, \epsilon) \in R\} \cup \\
&\quad \{(c_r, \text{in}; s, \text{out}), (r, \text{in}; d_{c,r}, \text{out}) \mid r, s \in S, c \in C, (s \rightarrow r, c_+) \in R\} \cup \\
&\quad \{(c'_r, \text{in}; s, \text{out}), (r, \text{in}; d_{c,r}, \text{out}) \mid r, s \in S, c \in C, (s \rightarrow r, c_-) \in R\} \cup \\
&\quad \{(c''_r, \text{in}; s, \text{out}), (r, \text{in}; d'_{c,r}, \text{out}) \mid r, s \in S, c \in C, (s \rightarrow r, c_{=0}) \in R\}, \\
R''_2 &= \{(c_r, \text{in}; r, \text{out}) \mid r \in S, c \in C, (s \rightarrow r, c_+) \in R \text{ for a } s \in S\} \cup \\
&\quad \{(c'_r, \text{in}; k, \text{out}), (k, \text{in}; d_{c,r}, \text{out}) \mid r \in S, c \in C, (s \rightarrow r, c_-) \in R \text{ for a } s \in S\} \cup \\
&\quad \{(c''_r, \text{in}; k'_1, \text{out}), (\dagger, \text{in}; d'_{c,r}, \text{out}), (k'_5, \text{in}; d'_{c,r}, \text{out}) \mid r \in S, c \in C, \\
&\quad \quad \quad (s \rightarrow r, c_{=0}) \in R \text{ for a } s \in S\} \cup \\
&\quad \{(k'_1, \text{in}; k'_2, \text{out}), (k'_2, \text{in}; k'_3, \text{out}), (k'_3, \text{in}; k'_4, \text{out}), (k'_4, \text{in}; k'_5, \text{out})\}, \\
R''_3 &= \{(c'_r, \text{in}), (c_t, \text{in}; d_{c,r}, \text{out}) \mid r, t \in S, c \in C, (s \rightarrow r, c_-) \in R \text{ for a } s \in S\} \cup
\end{aligned}$$

$$\begin{aligned}
& \{(c_r'', \text{in}; k_3', \text{out}), (c_r'', \text{in}; d_{c,r}', \text{out}), (c_t, \text{in}; d_{c,r}', \text{out}) \mid r, t \in S, c \in C, \\
& \hspace{15em} (s \rightarrow r, c_{=0}) \in R \text{ for a } s \in S\}, \\
R_4'' = & \{(c_r', \text{in}; d_{c,r}, \text{out}) \mid r \in S, c \in C, (s \rightarrow r, c_-) \in R \text{ for a } s \in S\} \cup \\
& \{(c_r'', \text{in}; d'c, r, \text{out}) \mid r \in S, c \in C, (s \rightarrow r, c_-) \in R \text{ for a } s \in S\}.
\end{aligned}$$

During this phase only one object per time related to a state of the simulated automaton can be present in membrane 1. If more than one instruction can be applied when the counter automaton is in a certain state, then in the P system the object related to the state present in membrane 1 can be subject to more than one rule. The value of a counter c is represented with occurrences of $c_r, r \in S$ present in membrane 2.

The simulation of an instruction of the kind $(s \rightarrow r, \epsilon)$ is performed by the rule $(r, \text{in}; s, \text{out}) \in R_1''$

The simulation of an instruction of the kind $(s \rightarrow r, c_+)$ starts with the application of the rule $(c_r, \text{in}; s, \text{out}) \in R_1''$ followed by the application of the rule $(c_r, \text{in}; r, \text{out}) \in R_2''$. If no occurrence of r is present in membrane 2, then the system never reaches the termination phase.

The simulation of an instruction of the kind $(s \rightarrow r, c_-)$ starts with the application of the rule $(c_r', \text{in}; s, \text{out}) \in R_1''$. The object c_r' is used by the P system to let an occurrence of a $c_t, t \in S$, to pass from membrane 2 to membrane 3 (simulating in this way the subtraction of one unit from counter c) if such an occurrence is present; if not, then the P system never reaches the termination phase. Once an object of the kind c_r' is present in membrane 1 a sequence of two, or eventually three rules is applied: first $(c_r', \text{in}; k, \text{out}) \in R_2''$, then $(c_r', \text{in}) \in R_3''$ and then $(c_r', \text{in}; d_{c,r}, \text{out}) \in R_4''$ if one object of the kind $d_{c,r}$ is present in membrane 4 (if this last rule cannot be applied the termination phase is never reached). If all three rules in sequence are applied, then the object c_r' is in membrane 4, one occurrence of $d_{c,r}$ is in membrane 3 and k is in membrane 1. If no occurrence of c_t for any $t \in S$ is present in membrane 2, then the system never reaches the termination phase. If there is at least one occurrence of c_t for any $t \in S$, then two rules can be applied in sequence: $(c_t, \text{in}; d_{c,r}, \text{out}) \in R_3''$ (effectively simulating the decrement of one unit of the counter c) and $(k, \text{in}; d_{c,r}, \text{out}) \in R_2''$. The simulation of the instruction ends with the application of the rule $(r, \text{in}; d_{c,r}, \text{out}) \in R_1''$.

The simulation of an instruction of the kind $(s \rightarrow r, c_{=0})$ starts with the application of the rule $(c_r'', \text{in}; s, \text{out}) \in R_1''$. Once a c_r'' is in membrane 1 a sequence of rules is applied. First the rule $(c_r'', \text{in}; k_1', \text{out}) \in R_2''$, then in parallel the rules $(c_r'', \text{in}; k_3', \text{out}) \in R_3''$ and $(k_1', \text{in}; k_2', \text{out}) \in R_2''$, and then in parallel the rules $(k_2', \text{in}; k_3', \text{out}) \in R_2''$ and $(c_r'', \text{in}; d_{c,r}', \text{out}) \in R_3''$. If there is no object $d_{c,r}'$ in membrane 4, then this rule cannot be applied and the termination phase is never reached. In this phase once one occurrence of one object of the kind $d_{c,r}'$ is in membrane 3 two sequences of rules can be applied. If an occurrence of one $c_t, t \in S$ is present in membrane 2, then first the rules $(k_3', \text{in}; k_4', \text{out}) \in R_2''$ and $(c_t, \text{in}; d_{c,r}', \text{out}) \in R_3''$ and then the rules $(\dagger, \text{in}; d_{c,r}', \text{out}) \in R_2''$ and $(k_4', \text{in}; k_5', \text{out}) \in R_2''$ are applied. As already said, once in membrane 2 the object \dagger forces the system to an infinite computation. If when one occurrence of one object of the kind $d_{c,r}'$ is in membrane 3 no of $c_t, t \in S$ is present in membrane 2, then the applied rules are first $(k_3', \text{in}; k_4', \text{out}) \in R_2''$, and then, in parallel, $(k_4', \text{in}; k_5', \text{out}) \in R_2''$ and $(k_3', \text{in}; d_{c,r}', \text{out}) \in R_3''$. In this case either $(\dagger, \text{in}; d_{c,r}', \text{out}) \in R_2''$ or $(k_5', \text{in}; d_{c,r}', \text{out}) \in R_2''$ can be applied. If this last is the applied rule, then the application of the rule $(r, \text{in}; d_{c,r}', \text{out}) \in R_1''$ ends the proper simulation of the instruction. One can notice that all the objects $k_i, 1 \leq i \leq 5$, are in the same membranes as they were at the beginning of

the simulation of the instruction.

When the object f related to the final state of the counter automaton is present in membrane 1, then the simulation phase ends and the termination phase starts. The instructions related to this phase let all the occurrences of out , the object related to the output counter of the simulated automaton, to pass from membrane 2 to membrane 4, and let all the objects different from out to pass from membrane 4 to other membranes of the system. Eventually the continuous passage of ∞_1 and ∞_2 from membrane 2 to membrane 3 and vice versa, is interrupted. It is important to notice that even if this phase is reached no result can be rendered by the P system (as, for instance, the object \dagger can keep passing from membrane 1 to membrane 2 and vice versa). If no rule involving \dagger has been used until the begin of the termination phase, the configuration of the P system at the beginning of this phase is similar to the one at the begin of the simulation phase with the exception that membrane 2 and membrane 3 can contain occurrences of objects related to the counters (except the ones related to the output counter) of the simulated automaton.

The rules related to the termination phase are:

$$\begin{aligned}
R_1''' &= \{(e, \text{in}; f, \text{out})\} \cup \{(e'', \text{in}; e', \text{out}), (e''', \text{in}; a_3, \text{out})\}; \\
R_2''' &= \{(e'', \text{in}; c_t, \text{out}) \mid t \in S, t \neq out\} \cup \{(e'', \text{in}; a_3, \text{out}), (e, \text{in}), (e', \text{out}), (e''', \text{in})\}; \\
R_3''' &= \{(c_t, \text{in}; e, \text{out}) \mid q \in Q\} \cup \{(e, \text{in}), (e_1, \text{in}; e', \text{out}), (e_2, \text{in}; e_1, \text{out}), (e_1, \text{in}; a_3, \text{out}), \\
&\quad (e''', \text{in}; \infty_1, \text{out}), (e''', \text{in}; \infty_2, \text{out})\}; \\
R_4''' &= \{(e''', \text{in}; X, \text{out}) \mid X \in \{c'_t, d_{c,t}, d'_{c,t} \mid c \in C, t \in S\}\} \cup \\
&\quad \{(out, \text{in})\} \cup \{(e, \text{out}), (e_2, \text{out}), (e''', \text{out}), (e, \text{in}; e', \text{out}), (e_2, \text{in}; a_3, \text{out})\}.
\end{aligned}$$

In the first part of this phase all the objects of the kind c_q can pass from membrane 2 to membrane 3, the objects out , related to the output counter of the simulated automaton, pass from membrane 3 to membrane 4 by the application of the rules $(out, \text{in}) \in R_4'''$. In the second part of this phase all the objects of the kind $c'_t, c''_t, d_{c,t}$ and $d'_{c,t}$ pass from membrane 4 to membrane 3 only if there are no more objects of the kind c_t in membrane 2.

When the object f related to the final state of the counter automaton is present in membrane 1, then the rule $(e, \text{in}; f, \text{out}) \in R_1'''$ can be applied. Once in membrane 1 the object e passes to membrane 3 by the sequential application of $(e, \text{in}) \in R_2'''$ and $(e, \text{in}) \in R_3'''$. Once in membrane 3 the object e can be subject of two sequences of rules. One of the rules $(c_t, \text{in}; e, \text{out}) \in R_3'''$ (moving one occurrence of c_t from membrane 2 to membrane 3) can be applied and followed by $(e, \text{in}) \in R_3'''$. Otherwise the rule $(e, \text{in}; e', \text{out}) \in R_4'''$ is first applied and followed by $(e, \text{out}) \in R_4'''$ and $(e_1, \text{in}; e', \text{out}) \in R_3'''$ applied in parallel. This second sequence of rules can be applied only once in the system (as there is only one occurrence of e' in membrane 4), while the first can be applied continuously (until there are objects of the kind c_t in membrane 2). Once the object e' is in membrane 2 a sequence of rules can be applied in the P system. First in parallel $(e', \text{out}) \in R_2'''$ and $(e_2, \text{in}; e_1, \text{out}) \in R_3'''$, then in parallel $(e'', \text{in}; e', \text{out}) \in R_1'''$ and $(e_2, \text{in}; a_3, \text{out}) \in R_4'''$. When e'' is in membrane 1 two scenarios are possible: in membrane 2 there are objects of the kind c_t , so the rule $(e'', \text{in}; c_t, \text{out}) \in R_2'''$ can be applied for a $t \in S$; or there are no objects of the kind c_t in membrane 2. In either cases the rule $(e_1, \text{in}; a_3, \text{out}) \in R_3'''$ can be applied in parallel with $(e_2, \text{out}) \in R_4'''$. Once the object a_3 is in membrane 2 the rule $(a_3, \text{in}) \in R_3'$ can be applied, if this happens the P system never renders a result. If not, then also the rule $(e'', \text{in}; a_3, \text{out}) \in R_2'''$ can be applied. It is worth notice that a_3 may pass to membrane 1 only if no object of the kind c_t is in membrane 2.

When the object a_3 is in membrane 1 a sequence of rules can be applied: first $(e''', \text{in}; a_3, \text{out}) \in R_1'''$, then $(e''', \text{in}) \in R_2'''$, then either $(e''', \text{in}; \infty_1, \text{out})$ or $(e''', \text{in}; \infty_2, \text{out})$ (both in R_3''' , the application of one of these rules ends the continuous passage of ∞_1 and ∞_2 from membrane 2 to membrane 3 and vice versa), then one of the rules $(e''', \text{in}; X, \text{out}) \in R_4'''$ followed by $(e''', \text{out}) \in R_4'''$. The application of these two last rules let the objects of the kind $c'_t, c''_t, d_{c,t}$ and $d'_{c,t}$ to pass from membrane 4 to membrane 3. It is important to notice that if c'_t (or c''_t) passes to membrane 3 while an occurrence of $d_{c,t}$ (or $d'_{c,t}$) is in membrane 4, then the rule $(c'_r, \text{in}; d_{c,r}, \text{out}) \in R_4''$ (or $(c''_r, \text{in}; d'_{c,r}, \text{out}) \in R_4''$) can be applied without changing the output of the P system. The computation stops when none of the rules $(e''', \text{in}; X, \text{out}) \in R_4'''$ can be applied any more. In this case the result of the computation is given by the number of objects of the kind *out* present in membrane 4. \square

4 No Maximal Parallelism

A few variants of P systems without maximal parallelism have been studied. The following two theorems show that the generative power of P systems with symport/antiport without maximal parallelism is equivalent to the one of partially blind counter automata.

Theorem 4.1 $\mathbb{N} \cdot \text{P}_{\neq} \text{P}_*(\text{sym}_*, \text{anti}_*) \subseteq L(M_{pb})$.

Proof. In P systems with symport/antiport the number of membranes and the number of different objects is finite. So it is possible to construct a partially blind counter automaton having a finite number of counters of the form O_m indicating how many occurrences of the object O are present in membrane m . Such counters are not present for objects initially present (with infinite cardinality) in the environment on the simulated P systems.

The simulation is performed in two, or possibly three logical phases: *initialization*, *simulation*, and, eventually, *termination*.

In the initialization phase a deterministic sequence of instructions codes the initial configuration of the simulated P systems in the registers. If, for instance, the initial configuration of the simulated P system has the objects *aac* in membrane 1 and *bb* in membrane 2, then the sequence of instructions is: $(s_0 \rightarrow s_1, a_{1+}), (s_1 \rightarrow s_2, a_{1+}), (s_2 \rightarrow s_3, c_{1+}), (s_3 \rightarrow s_4, b_{2+}), (s_4 \rightarrow s_5, b_{2+})$. The state reached by the last of this sequence of instruction should be \bar{s} (so, in the previous example $s_5 = \bar{s}$). Once in this state, the partially blind counter automaton is in the simulation phase.

As the name suggests, during the simulation phase the rules of the P systems are simulated. As the maximal parallelism is not present, in the simulated system all its rules can be simulated in sequence. Each symport and antiport rule is simulated by a sequence of instructions all starting and ending in state \bar{s} . In this way any symport or antiport can be simulated at any moment during the computation. If the partially blind counter automaton tries to simulate a rule that cannot be applied in a particular situation of the simulated P system, then it halts in a non final state.

As a symport rule can be considered as an antiport rule with one of the two moved multisets of objects missing, in the following we are going to give a description of the simulation of a generic antiport rule. We consider that each rule of the simulated P system has a unique number associated to it. So, for instance, we consider the antiport rule $(a_1 a_2 \dots a_p, \text{in}; b_1 b_2 \dots b_q, \text{out})$ having index r present in R_j , with membrane i father of membrane j . The sequence of instructions simulating this rule are:

$$\begin{aligned}
&(\bar{s} \rightarrow \bar{s}_r^1, a_{1i-}), (\bar{s}_r^1 \rightarrow \bar{s}_r^2, a_{2i-}), \dots, (\bar{s}_r^{(p-1)} \rightarrow \bar{s}_r^p, a_{pi-}), \\
&(\bar{s}_r^p \rightarrow \bar{s}_r^{1'}, b_{1i-}), (\bar{s}_r^{1'} \rightarrow \bar{s}_r^{2'}, b_{2i-}), \dots, (\bar{s}_r^{(q-1)'} \rightarrow \bar{s}_r^{q'}, b_{qi-}), \\
&(\bar{s}_r^{q'} \rightarrow \bar{s}_r^{1''}, a_{1j+}), (\bar{s}_r^{1''} \rightarrow \bar{s}_r^{2''}, a_{2j+}), \dots, (\bar{s}_r^{(p-1)''} \rightarrow \bar{s}_r^{p''}, a_{pj+}), \\
&(\bar{s}_r^{p''} \rightarrow \bar{s}_r^{1'''}, b_{1j+}), (\bar{s}_r^{1'''} \rightarrow \bar{s}_r^{2'''}, b_{2j+}), \dots, (\bar{s}_r^{(q-1)'''} \rightarrow \bar{s}, b_{qj+}).
\end{aligned}$$

The sequence of instructions present in the top line decreases of one unit the counters related to the objects $a_1 a_2 \dots a_p$ present in membrane i ; the following line indicates the sequence of instructions decreasing the counters related to the objects $b_1 b_2 \dots b_q$ that pass to membrane i ; the following line indicates the sequence of instructions increasing the counters related to the objects $a_1 a_2 \dots a_p$ that pass to membrane j , while the bottom line indicates the sequence of instructions increasing the counters related to the objects $b_1 b_2 \dots b_q$ that pass to membrane i .

If membrane i is the environment and (some of) the objects $a_1 a_2 \dots a_p$ are initially present (in unbounded copies) in the environment, then in the previous sequence of instructions the top line is not present and $\bar{s}_r^p = \bar{s}$. If membrane i is the environment and (some of) the objects $b_1 b_2 \dots b_q$ are initially present (in unbounded copies) in the environment, then in the previous sequence of instructions the bottom line is not present and $\bar{s}_r^{p''} = \bar{s}$.

If any of the instructions tries to subtract one unit from a counter, then the partially blind counter automaton stops in a non final state. If the simulated rule concerns the passage of a (particular) object in the acknowledgment membrane, then the last state reached by the counter automaton by the simulation of the rule is the final state f .

It is possible to construct a partially blind counter automaton ending with all counter at zero with the exception of the output counter.

In this case the simulation of a rule letting a (particular) object to pass to the acknowledgment membrane would not bring the counter automaton to the final state f but to the non final state \bar{s} . In this case the instruction $(\bar{s} \rightarrow f, \epsilon)$, with f final state would be present in the counter automaton together with the instructions $(\bar{s} \rightarrow \bar{s}_c, \epsilon)$, $(\bar{s}_c \rightarrow \bar{s}, \epsilon)$ and $(\bar{s}_c \rightarrow \bar{s}_c, c_-)$ for each counter c different than the output counter. These last instructions decrease of a random value each counter different than the output one, the attempt of subtraction of an empty counter would stop the automaton in a non final state. The counter automaton accepts the number present in the output counter if when in the final state all the counters except the output one are empty. \square

Theorem 4.2 $L(M_{pb}) \subseteq \mathbb{N} \cdot \mathbb{P} \neq \mathbb{P}_2(\text{sym}_1, \text{anti}_2)$.

Proof. We consider that given any partially blind counter automaton it is possible to create another one that ends with all counters empty except the output counter.

The P system with symport/antiport without maximal parallelism has in its initial configuration one instance of the object s_0 , related to the initial state of the simulated counter automaton, in membrane 1. Objects related to the other states and to the counters of the simulated automaton are present in unbounded copies in the environment. Membrane 2, the acknowledgment membrane, is empty for all configurations except the final one. In this configuration it contains the object related to the final state of the simulated automaton. Membrane 1 is the final membrane.

The simulation of an instruction of the kind $(s \rightarrow r, \epsilon)$ is performed by the rule $(r, \text{in}; s, \text{out})$. The simulation of an instruction of the kind $(s \rightarrow r, c_+)$ is performed by the

rule $(rc, \text{in}; s, \text{out})$. The simulation of an instruction of the kind $(s \rightarrow r, c_-)$ is performed by the rule $(r, \text{in}; sc, \text{out})$.

The only rule associated to membrane 2 is (f, in) , where f is the final state of the simulated automaton. The computation of the P system halts when on occurrence related to a final state of the simulated automaton passes from membrane 1 to membrane 2. The number of objects present in membrane 1 in the final configuration is equal to the value of the output counter of the simulated automaton. \square

5 Final Remarks

This paper first gives a result concerning P systems with symport/antiport with maximal parallelism. It is proved that four membranes suffice to generate all recursively enumerable sets of numbers. The challenge to discover the family of sets of numbers generated by these systems with less than four membranes remains.

We also investigate the power of P systems with symport/antiport without maximal parallelism discovering that it is equivalent to the power of partially blind counter automaton. In this way we partially answer an open problem present in [12]

Acknowledgments. We thank F. Bernardini for the stimulating discussion related to what presented in Section 3.

References

- [1] F. Bernardini, M. Gheorghe. On the power of minimal symport/antiport. *PreProceedings Workshop on Membrane Computing, WMC-2003*, Tarragona, July 17-22, 2003, 2003. Technical Report, 28/03, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, 72–83.
- [2] F. Bernardini, A. Păun. Symport/antiport: Five membranes suffices. In A. Alhazov, C. Martín-Vide, and G. Păun, editors, *Workshop on Membrane Computing, WMC-2003, Tarragona*, pages 72–83, 2003.
- [3] P. Frisco. The conformon-P system: A molecular and cell biology-inspired comutability model. *Theoretical Computer Science*, 312(2-3):295–319, 2004.
- [4] P. Frisco, H.J. Hoogeboom. P systems with symport/antiport simulating counter automata. Submitted.
- [5] J.E. Hopcroft, D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [6] L. Kari, C. Martín-Vide, A. Păun. *Aspects of Molecular Computing: Essays Dedicated to Tom Head, on the Occasion of His 70th Birthday*, volume 2950 of *Lecture Notes in Computer Science*, chapter On the Universality of P Systems with Minimal Symport/Antiport Rules, pages 254–265. Springer-Verlag, Berlin, Heidelberg, New York, 2004.
- [7] C. Martín-Vide, A. Păun, G. Păun. On the power of P systems with symport rules. *The Journal of Universal Computer Science*, 8:317–331, 2002.

- [8] C. Martín-Vide, A. Păun, G. Păun, G. Rozenberg. Membrane systems with coupled transport: universality and normal forms. *Fundamenta Informaticae*, 49:1–15, 2002.
- [9] M.L. Minsky. Recursive unsolvability of Post’s problem of “tag” and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, November 1961.
- [10] A. Păun, G. Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20(3):295–306, 2002.
- [11] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 1(61):108–143, 2000.
- [12] G. Păun. *Membrane Computing, An Introduction*. Springer-Verlag, Berlin, Heidelberg, New York, 2002.
- [13] G. Păun, S. Yu. On synchronization in P systems. *Fundamenta Informaticae*, 34(4):397–410, 1999.