Towards a Characterization of P Systems with Minimal Symport/Antiport and Two Membranes *

Artiom Alhazov[1,2] and Yurii Rogozhin[1]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
{artiom,rogozhin}@math.md
[2] Research Group on Mathematical Linguistics
Rovira i Virgili University, Tarragona, Spain
artiome.alhazov@estudiants.urv.cat

Abstract. We prove that any set of numbers containing zero generated by symport/antiport P systems with two membranes and minimal cooperation is finite (for both symport/antiport P systems and for purely symport P systems). On the other hand, one additional object in the output membrane allows symport/antiport P systems with two membranes and minimal cooperation generate any recursively enumerable sets of natural numbers without zero. The same question for symport P systems with two membranes and minimal cooperation (is only one additional object in the output membrane sufficient in order to get universality?) is still open.

1 Introduction

P systems with *symport/antiport* rules, i.e., P systems with *pure communication rules assigned to membranes*, first were introduced in [18]; symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions. These operations are very powerful, i.e., P systems with symport/antiport rules have universal computational power with only one membrane, e.g., see [9], [13], [10].

A comprehensive overview of the most important results obtained in the area of P systems and tissue P systems with *symport/antiport* rules, especially with respect to the development of computational completeness results improving descriptional complexity parameters as the number of membranes and cells, respectively, and the weight of the rules as well as the number of objects can be found in [1].

* The work of the first author is partially supported by the project TIC2003-09319-C03-01 from Rovira i Virgili University, and the second author is supported by the project 06.411.03.04P from Consiliul Suprem Pentru Ştiinta şi Dezvoltare Tehnologică al Academiei de Ştiinte a Moldovei.

At first we show that if some P system with two membranes and with minimal cooperation, i.e., a P system with symport/antiport rules of weight one or a P system with symport rules of weight two, generates a set of numbers *containing zero*, then this set is **finite**. After that we prove that P systems with symport/antiport rules of weight one can generate any *recursively enumerable* set of natural numbers without zero (i.e., they are computationally complete with just **one superfluous object** remaining in the output membrane at the end of a halting computation). Thus we improved the result from [1] for symport/antiport P systems with two membranes and minimal cooperation from three objects down to one object and showed the optimality of this result.

The same question for symport P systems with two membranes and minimal cooperation (is only one additional object in the output membrane sufficient in order to get universality?) is still open. Notice that symport/antiport P systems with three membranes and minimal cooperation can generate any recursively enumerable sets of natural numbers without using superfluous objects in the output membrane [3]. The question about precise characterization of computational power of symport/antiport P systems with two membranes and minimal cooperation is still open.

2 Basic Notations and Definitions

For the basic elements of formal language theory needed in the following, we refer to [23]. We just list a few notions and notations: $\mathbb{N}$ denotes the set of natural numbers (i.e., of non-negative integers). $V*$ is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; by $\mathbb{N}RE$, $\mathbb{N}REG$, and $\mathbb{N}FIN$ we denote the family of recursively enumerable sets, regular sets, and finite sets of natural numbers, respectively. For $k \geq 1$, by $\mathbb{N}_k RE$ we denote the family of recursively enumerable sets of natural numbers excluding the initial segment 0 to $k-1$. Equivalently, $\mathbb{N}_k RE = \{k + L \mid L \in \mathbb{N}RE\}$, where $k + L = \{k + n \mid n \in L\}$. Particularly, $\mathbb{N}_1 RE = \{N \in \mathbb{N}RE \mid 0 \notin N\}$. We will also use the next notations: $\mathbb{N}_{\ni 0} FIN = \{N \in \mathbb{N}FIN \mid 0 \in N\}$ and $\mathbb{N}_{\ni 0} SEG_1 = \{\{k \in \mathbb{N} \mid k < n\} \mid n \geq 0\}$.

The families of recursively enumerable sets of vectors of natural numbers are denoted by $PsRE$.

2.1 Counter Automata

A non-deterministic *counter automaton* (see [8], [1]) is a construct

$$M = (d, Q, q_0, q_f, P)\,, \text{ where}$$

- d is the number of counters, and we denote $D = \{1, ..., d\}$;
- Q is a finite set of states, and without loss of generality, we use the notation $Q = \{q_i \mid 0 \leq i \leq f\}$ and $F = \{0, 1, ..., f\}$,
- $q_0 \in Q$ is the initial state,
- $q_f \in Q$ is the final state, and

– P is a finite set of instructions of the following form:
  1. $(q_i \to q_l, k+)$, with $i, l \in F,\ i \neq f,\ k \in D$ (" increment" -instruction). This instruction increments counter k by one and changes the state of the system from q_i to q_l.
  2. $(q_i \to q_l, k-)$, with $i, l \in F,\ i \neq f,\ k \in D$ (" decrement" -instruction). If the value of counter k is greater than zero, then this instruction decrements it by 1 and changes the state of the system from q_i to q_l. Otherwise (when the value of counter k is zero) the computation is blocked in state q_i.
  3. $(q_i \to q_l, k = 0)$, with $i, l \in F,\ i \neq f,\ k \in D$ (" test for zero" -instruction). If the value of counter k is zero, then this instruction changes the state of the system from q_i to q_l. Otherwise (the value stored in counter k is greater than zero) the computation is blocked in state q_i.
  4. *halt*. This instruction stops the computation of the counter automaton, and it can only be assigned to the final state q_f.

A transition of the counter automaton consists in updating/checking the value of a counter according to an instruction of one of the types described above and by changing the current state to another one. The computation starts in state q_0 with all counters being equal to zero. The result of the computation of a counter automaton is the value of the first k counters when the automaton halts in state $q_f \in Q$ (without loss of generality we may assume that in this case all other counters are empty). A counter automaton thus (by means of all computations) generates a set of k-vectors of natural numbers.

It is known that any set of k-vectors of natural numbers from $PsRE$ can be generated by a counter automaton with $k+2$ counters where only " increment" -instructions are needed for the first k counters. We will use this fact in our proofs.

2.2 P Systems with Symport/Antiport Rules

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [20]; comprehensive information can be found in the P systems web page, [27].

A *P system with symport/antiport rules* is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_k, E, R_1, \ldots, R_k, i_0), \text{ where}$$

1. O is a finite alphabet of symbols called *objects*;
2. μ is a *membrane structure* consisting of k membranes that are labelled in a one-to-one manner by $1, 2, \ldots, k$;
3. $w_i \in O^*$, for each $1 \leq i \leq k$, is a finite multiset of objects associated with the region i (delimited by membrane i);
4. $E \subseteq O$ is the set of objects that appear in the environment in an infinite number of copies;

5. R_i, for each $1 \leq i \leq k$, is a finite set of symport/antiport rules associated with membrane i; these rules are of the forms (x, in) and (y, out) *(symport rules)* and $(y, out; x, in)$ *(antiport rules)*, respectively, where $x, y \in O^+$;
6. i_0 is the label of an elementary membrane of μ that identifies the corresponding output region.

A P system with symport/antiport rules is defined as a computational device consisting of a set of k hierarchically nested membranes that identify k distinct regions (the membrane structure μ), where to each membrane i there are assigned a multiset of objects w_i and a finite set of symport/antiport rules R_i, $1 \leq i \leq k$. A rule $(x, in) \in R_i$ permits the objects specified by x to be moved into region i from the immediately outer region. Notice that for P systems with symport rules the rules in the skin membrane of the form (x, in), where $x \in E^*$, are forbidden. A rule $(x, out) \in R_i$ permits the multiset x to be moved from region i into the outer region. A rule $(y, out; x, in)$ permits the multisets y and x, which are situated in region i and the outer region of i, respectively, to be exchanged. It is clear that a rule can be applied if and only if the multisets involved by this rule are present in the corresponding regions. The weight of a symport rule (x, in) or (x, out) is given by $|x|$, while the weight of an antiport rule $(y, out; x, in)$ is given by $max\{|x|, |y|\}$.

As usual, a computation in a P system with symport/antiport rules is obtained by applying the rules in a non-deterministic maximally parallel manner. Specifically, in this variant, a computation is restricted to moving objects through membranes, since symport/antiport rules do not allow the system to modify the objects placed inside the regions. Initially, each region i contains the corresponding finite multiset w_i, whereas the environment contains only objects from E that appear in infinitely many copies.

A computation is successful if starting from the initial configuration, the P system reaches a configuration where no rule can be applied anymore. The result of a successful computation is a natural number that is obtained by counting all objects (only the terminal objects as it done in [2], if in addition we specify a subset of O as the set of terminal symbols) present in region i_0. Given a P system Π, the set of natural numbers computed in this way by Π is denoted by $N(\Pi)$ (or $N(\Pi)_T$ if the terminal symbols are distinguished). If the multiplicity of each (terminal) object is counted separately, then a vector of natural numbers is obtained, denoted by $Ps(\Pi)$, see [20].

By

$$\mathbb{N}OP_m(sym_s, anti_t)$$

we denote the family of sets of natural numbers that are generated by a P system with symport/antiport rules having at most $m > 0$ membranes, symport rules of size at most $s \geq 0$, and antiport rules of size at most $t \geq 0$. By

$$\mathbb{N}_kOP_m(sym_s, anti_t)$$

we denote the corresponding families of natural numbers without initial segment $\{0, 1, \ldots, k-1\}$ generated by P systems. If we replace numbers by vectors, then

in the notations above $\mathbb{N}$ is replaced by Ps. When any of the parameters m, s, t is not bounded, it is replaced by $*$. If $t = 0$, then we may even omit $anti_t$.

3 The Garbage is Unavoidable

Theorem 1. *If* $M \in \mathbb{N}OP_2(sym_1, anti_1)$, *then* $0 \in M \Rightarrow M \in \mathbb{N}FIN$.

Proof. Consider an arbitrary P system Π with two membranes and symport/antiport rules of weight one,

$$\Pi = (O, [_1\ [_2\ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2).$$

For Π, consider some computation $\mathcal{C}$ generating 0: $\mathcal{C}$ ends in some configuration C with nothing in membrane 2, $u_1 \in (O-E)^*$ and $u_e \in E^*$ in membrane 1 and $u_0 \in (O-E)^*$ in the environment. Finally, consider an arbitrary halting computation $\mathcal{C}'$ of Π: $\mathcal{C}'$ ends in some configuration C' with $v_2 \in (O-E)^*$ and $v_f \in E^*$ in membrane 2, with $v_1 \in (O-E)^*$ and $v_e \in E^*$ in membrane 1 and $v_0 \in (O-E)^*$ in the environment. We are claiming that $|v_2 v_f| + |v_1 v_e| \leq |w_2| + |w_1|$ (i.e., the total number of objects in the system cannot grow without starting an infinite computation, and thus Π cannot generate numbers greater than the initial number of objects inside it).

Let us assume the contrary. Since the number of objects inside the system can only increase by symport rules, some rule $p_0 : (s_0, in) \in R_1$ had to be applied at some step of $\mathcal{C}'$ (by definition $s_0 \in O - E$). This implies that s_0 has been brought to the environment. We can assume that rules $p_i : (s_i, out; s_{i-1}, in) \in R_1$, $1 \leq i < n$, have been applied ($n \geq 0$), $s_i \in O - E$, $1 \leq i \leq n$. Suppose also that n is maximal (s_n was not brought to the environment by antiport with another object from $O - E$). Thus R_1 contains either a rule $p : (s_n, out) \in R_1$, or $p' : (s_n, out; a, in) \in R_1$, $a \in E$.

Now let us examine the final configuration C of the computation generating 0. Recall that since region 2 is empty, we cannot "hide" anything there. If s_0 is in u_0, then p_0 can be applied, hence C is not final. Therefore (region 2 is empty) s_0 is in u_1. For all $1 \leq i \leq n$, given $s_{i-1} \in w_1$, if s_i is in u_0, then p_i can be applied, hence C is not final. Consequently (region 2 is empty), s_i is in w_1 as well. By induction, we obtain that s_n is in w_1. However, this implies that either $p \in R$ and p can be applied, or some $p' \in R$ and p' can be applied, therefore C is not final.

This implies that if a system may generate 0, then any computation where the number of objects inside the output membrane is increased cannot halt. Therefore, Π cannot generate infinite sets containing 0. □

The corresponding result also holds for systems with symport of weight at most two, but the proof is more difficult.

Theorem 2. *If* $M \in \mathbb{N}OP_2(sym_2)$, *then* $0 \in M \Rightarrow M \in \mathbb{N}FIN$.

Proof. Consider an arbitrary P system Π with two membranes and symport rules of weight at most two, $\Pi = (O, [_1\ [_2\ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2)$; without restricting generality we may assume that the objects that compose w_1 and w_2 are disjoint from the objects in E. For Π, consider some computation $\mathcal{C}$ generating 0: $\mathcal{C}$ ends in some configuration C with nothing in membrane 2, $u_1 \in (O-E)^*$ and $u_e \in E^*$ in membrane 1 and $u_0 \in (O-E)^*$ in the environment. Finally, consider an arbitrary halting computation $\mathcal{C}'$ of Π: $\mathcal{C}'$ ends in some configuration C' with $v_2 \in (O-E)^*$ and $v_f \in E^*$ in membrane 2, with $v_1 \in (O-E)^*$ and $v_e \in E^*$ in membrane 1 and $v_0 \in (O-E)^*$ in the environment. We are claiming that $|v_2 v_f| + |v_1 v_e| \leq |w_2| + |w_1|$ (i.e., the total number of objects in the system cannot grow without starting an infinite computation, and thus Π cannot generate numbers greater than the initial number of objects inside it).

Let us assume the contrary. Denote by I_0 the set of objects from $O-E$ that we know must be in the environment in order for Π to halt with region 2 being empty; start with $I_0 = \emptyset$. Since bringing from the environment some object $a \in E \cup I_0$ is necessary (though not sufficient) to increase the number of objects inside the system, some rule $(ab, in) \in R_1$ had to be applied at some step of $\mathcal{C}'$ (if $a \in E$, by definition $b \in O-E$; if $a \in I_0$, then also b must be in $O-E$, otherwise rule (ab, in) would be applicable in C, which is supposed to be a halting configuration).

Clearly, object b was originally in region 1, so it has been brought to the environment by some rule $(b, out) \in R_1$ or $(bc, out) \in R_1$. In the first case, the system cannot halt without "hiding" object b in region 2 (contradiction with the assumption on C). In the second case, we have a few possibilities. If $c = a' \in E$, then by application of rules $(a'b, out)$, (ab, in) we have simply exchanged a' by a in region 1; since a' has been brought in the region 1 beforehand, we can repeat the same reasoning taking a' instead of a (this may only happen a finite number of times since we examine the computation $\mathcal{C}$ backwards). Finally, if $c = b' \in O-E$, then by application of rules $(a'b, out)$, (ab, in) we have exchanged b' by a in region 1. This will not increase the number of objects unless b' does not stay in the environment. Notice also that in configuration C object b' has to be in the environment. Add b' to I_0 and repeat the same reasoning.

The argument above implies that if a system can increase the number of objects inside it, then it cannot halt without any objects in region 2. Therefore, Π cannot generate infinite sets containing 0. □

4 Universality

Theorem 3. $\mathbb{N}OP_2(sym_1, anti_1) = \mathbb{N}_1 RE \cup F$, *where* $\mathbb{N}_{\exists 0} SEG_1 \subseteq F \subseteq \mathbb{N}_{\exists 0} FIN$.

Proof. We give here only sketch of a proof. The full variant of the proof is rather long and contains a lot of cases and will be prepared separately. While the upper bound of F results from Theorem 1, the lower bound of F is satisfied even by one-membrane constructions, see [4]. In what follows, we deal with proving $\mathbb{N}_1 OP_2(sym_1, anti_1) = \mathbb{N}_1 RE$.

We simulate a counter automaton $M = (d, Q, q_0, q_f, P)$ which starts with empty counters. We also suppose that all instructions from P are labelled in a one-to-one manner with elements of $\{1, \ldots, n\} = I$; I is the disjoint union of $\{n\}$ as well as I_+, I_-, and $I_{=0}$ where by I_+, I_-, and $I_{=0}$ we denote the set of labels for the " increment" -, " decrement" -, and " test for zero" -instructions, respectively. We use also the next notations: $C = \{c_k\}, k \in D$, $Q' = \{q'_i\}, q_i \in Q$.

We construct the P system Π_1 as follows:

$$
\begin{aligned}
\Pi_1 &= (O, [_1\ [_2\ \ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2),\\
O &= E \cup \{I_c, M, S, T_1, T_2, J_1, J_2\} \cup \{b_j, d_j \mid j \in I\},\\
E &= Q \cup Q' \cup C \cup \{a_j, a'_j \mid j \in I\} \cup \{J_0, F_1, F_2, F_3, F_4, F_5\},\\
w_1 &= J_1 J_2,\\
w_2 &= T_1 T_2 M S \prod_{j \in I} b_j \prod_{j \in I} d_j,\\
R_i &= R_{i,s} \cup R_{i,r} \cup R_{i,f}, \quad i = 1, 2.
\end{aligned}
$$

The functioning of this system may be split into three stages:

1. preparation of the system for the computation.
2. simulation of instructions of the counter automaton.
3. terminating the computation.

We code the counter automaton as follows:

Region 1 will hold the current state of the automaton, represented by a symbol $q_i \in Q$; region 2 will hold the value of all counters, represented by the number of occurrences of symbols $c_k \in C$, $k \in D$, where $D = \{1, ..., d\}$.

We split our proof into several parts that depend on the logical separation of the behavior of the system. We will present the rules and the initial symbols for each part, but we remark that the system we present is the union of all these parts. The rules R_i are given by three phases:

1. START;
2. RUN;
3. END.

The parts of the computations illustrated in the following describe different stages of the evolution of the P system. For simplicity, we focus on explaining a particular stage and omit the objects that do not participate in the evolution at that time. Each rectangle represents a membrane, each variable represents a copy of an object in a corresponding membrane (symbols outside of the outermost rectangle are found in the environment). In each step, the symbols that will evolve (will be moved) are written in **boldface**. The labels of the applied rules are written above the symbol $\Rightarrow$.

1. START. During the first stage we bring from the environment an arbitrary number of symbols c_k, $k \in D$, into region 1. We suppose that we have enough symbols c_k in region 1 to perform the computation. Otherwise, the computation will never stop. We also use the following idea: in our system we have a symbol M which moves from region 2 to region 1 and back in an infinite loop. This loop may be stopped only if all stages completed correctly.

$$R_{1,s} = \{\texttt{1s1} : (I_c, in), \texttt{1s2} : (I_c, out; c_k, in), \texttt{1s3} : (S, out; q_0, in) \mid c_k \in C\}$$
$$R_{2,s} = \{\texttt{2s1} : (M, out), \texttt{2s2} : (M, in), \texttt{2s3} : (S, out; I_c, in)\}$$

Symbol I_c brings symbols c_k from the environment into region 1 (rules `1s1`, `1s2`), where they may be used during the simulation of the " increment" instruction and then moved to region 2.

We illustrate the beginning of the computation as follows:

$$\mathbf{c_{k_1}} c_{k_2} \ldots c_{k_t} q_0 \boxed{\mathbf{I_c} \boxed{\mathbf{M} S}} \Rightarrow^{\texttt{1s2,2s1}} \mathbf{I_c} c_{k_2} \ldots c_{k_t} q_0 \boxed{c_{k_1} \mathbf{M} \boxed{S}} \Rightarrow^{\texttt{1s1,2s2}}$$
$$\mathbf{c_{k_2}} \ldots c_{k_t} q_0 \boxed{\mathbf{I_c} c_{k_1} \boxed{\mathbf{M} S}} \Rightarrow^{\texttt{1s2,2s1}} \ldots \mathbf{I_c} q_0 \boxed{c_{k_1} c_{k_2} \ldots c_{k_t} \mathbf{M} \boxed{S}} \Rightarrow^{\texttt{1s1,2s2}}$$
$$q_0 \boxed{\mathbf{I_c} c_{k_1} c_{k_2} \ldots c_{k_t} \boxed{\mathbf{MS}}} \Rightarrow^{\texttt{2s1,2s3}} \mathbf{q_0} \boxed{\mathbf{S} c_{k_1} c_{k_2} \ldots c_{k_t} \mathbf{M} \boxed{I_c}} \Rightarrow^{\texttt{1s3,2s2}}$$
$$S \boxed{\mathbf{q_0} c_{k_1} c_{k_2} \ldots c_{k_t} \boxed{\mathbf{M} I_c}}$$

I_c is eventually exchanged with S, which in turn brings q_0 into region 1, and the simulation of the register machine begins. Symbol I_c is then situated in region 2 and can be used during the second stage as a "trap" symbol, i.e., in order to organize an infinite computation.

Notice that some rules are never executed during a correct simulation: applying them would lead to an infinite computation. To help the reader, we will underline the labels of such rules in the description below.

2. RUN.

$$
\begin{aligned}
R_{1,r} = {} & \{\mathtt{1r1} : (q_i, out; a_j, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, =0\}\} \\
& \cup \{\mathtt{1r2} : (b_j, out; a'_j, in), \mathtt{1r3} : (a_j, out; J_0, in), \\
& \quad \mathtt{1r4} : (J_1, out; b_j, in) \mid j \in I\} \cup \{\mathtt{1r5} : (J_0, out; J_1, in)\} \\
& \cup \{\mathtt{1r6} : (a'_j, out; d_j, in) \mid j \in I_- \cup I_{=0}\} \\
& \cup \{\mathtt{1r7} : (a'_j, out; J_0, in),\ \mathtt{1r8} : (J_0, out; d_j, in) \mid j \in I_+\} \\
& \cup \{\underline{\mathtt{1r9}} : (J_2, out; d_j, in) \mid j \in I_+ \cup I_-\} \\
& \cup \{\mathtt{1r10} : (a'_j, out; J_1, in) \mid j \in I_{=0}\} \\
& \cup \{\mathtt{1r11} : (d_j, out; q'_l, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, =0\}\} \\
& \cup \{\mathtt{1r12} : (q'_j, out, q_j, in), \mathtt{1r13} : (b_j, out), \mathtt{1r14} : (d_j, out) \mid j \in I\}. \\
R_{2,r} = {} & \{\mathtt{2r1} : (b_j, out; a_j, in) \mid j \in I\} \\
& \cup \{\mathtt{2r2} : (c_k, out; a'_j, in) \mid (j : q_i \to q_l, c_k = 0) \in P\} \\
& \cup \{\mathtt{2r3} : (c_k, out; b_j, in) \mid (j : q_i \to q_l, c_k-) \in P\} \\
& \cup \{\underline{\mathtt{2r4}} : (a'_j, out; J_1, in) \mid j \in I_{=0} \cup I_+\} \\
& \cup \{\mathtt{2r5} : (a'_j, out) \mid j \in I_-\} \\
& \cup \{\mathtt{2r6} : (a'_j, out; c_k, in) \mid j : q_i \to q_l, c_k+) \in P\} \\
& \cup \{\underline{\mathtt{2r7}} : (a_j, out; J_1, in) \mid j \in I\} \\
& \cup \{\mathtt{2r8} : (a_j, out; J_2, in) \mid j \in I\} \\
& \cup \{\mathtt{2r9} : (d_j, out; b_j, in) \mid j \in I_{=0} \cup I_+\} \\
& \cup \{\mathtt{2r10} : (d_j, out; a'_j, in) \mid j \in I_-\} \\
& \cup \{\mathtt{2r11} : (q'_l, out; d_j, in),\ \mathtt{2r12} : (q'_l, out; J_1, in), \\
& \quad \mathtt{2r13} : (q'_l, in) \mid (j : q_i \to q_l, c_k+) \in P\} \\
& \cup \{\mathtt{2r14} : (J_2, out; d_j, in) \mid j \in I_{=0} \cup I_-\} \\
& \cup \{\mathtt{2r15} : (J_2, out; a'_j, in) \mid j \in I_+\} \\
& \cup \{\underline{\mathtt{2r16}} : (I_c, out; a_j, in) \mid j \in I\} \\
& \cup \{\underline{\mathtt{2r17}} : (I_c, out; a'_j, in), \mid j \in I_+ \cup I_-\} \\
& \cup \{\underline{\mathtt{2r18}} : (I_c, out; J_0, in)\}
\end{aligned}
$$

Let us informally explain some details of the simulation design: The state symbol q_i brings into the system the instruction symbol a_j, which "activates" another instruction symbol b_j. The latter, in turn, brings symbol a'_j into the system. While symbols a_j, a'_j return to the environment and b_j returns to region 2, yet another instruction symbol, d_j, is activated, which will also return to region 2. Somewhere in the process, c_k is either moved to region 2 (increment), or it is removed from region 2 (decrement), or it is not present in region 2 (zero test); the next state symbol q'_l is brought into the system, which will exchange with q_l and the next instruction can be simulated.

Symbols a_j and a'_j travel from the environment to region 2 and back (except for a'_j during the zero test), while b_j and d_j travel from region 2 to the environment and back. The most difficult situations are when either of these symbols is in region 1. Symbols J_0, J_1 and J_2 guide this process.

For instance, a_j brings J_2 to region 2, so that it helps a'_j come to region 2 (increment) or it helps d_j return to region 2 (decrement or zero test).

Symbol a_j can "legally" return to the environment bringing J_0 to region 1 only if symbol b_j brings J_1 in the environment, so that it would return J_0 to the environment (otherwise, `2r18` is executed). On the other hand, a_j can only come to region 2 when b_j is there.

Likewise, in the increment case, a'_j can "legally" return to the environment bringing J_0 to region 1 only if symbol d_j returns J_0 to the environment (otherwise, `2r18` is executed). On the other hand, a'_j can only come to region 2 with the help of J_2 (increment), or when d_j is there (decrement).

If b_j does not come to the environment when it "should", it would lead J_0 (activated by a_j) without J_1, causing `2r18` to be executed. On the other hand, the case when b_j "should" come to region 2 is handled by Lemma 1.

If d_j comes to region 2 "too soon", then q'_l would never be brought into the system; then the next instructions will not be simulated, and the infinite computation would be enforced by M. On the other hand, the case when d_j "should" come to region 2 is handled by Lemma 2.

We will now consider the "main" line of computation.

" Increment" -instruction:

(i) *There is some c_k in region 1:*

$$q_l q'_l \mathbf{a_j} a'_j J_0 \boxed{\mathbf{q_i} c_k J_1 J_2 \boxed{b_j d_j I_c}} \Rightarrow^{\texttt{1r1}} q_i q_l q'_l a'_j J_0 \boxed{\mathbf{a_j} c_k J_1 J_2 \boxed{\mathbf{b_j} d_j I_c}} \Rightarrow^{\texttt{2r1}}$$

$$q_i q_l q'_l \mathbf{a'_j} J_0 \boxed{\mathbf{b_j} c_k J_1 \mathbf{J_2} \boxed{\mathbf{a_j} d_j I_c}} \Rightarrow^{\texttt{1r2,2r8}} q_i q_l q'_l \mathbf{b_j J_0} \boxed{\mathbf{a'_j} c_k \mathbf{J_1 a_j} \boxed{\mathbf{J_2} d_j I_c}} \Rightarrow^{\texttt{1r3,1r4,2r15}}$$

$$q_i q_l q'_l a_j \mathbf{J_1} \boxed{\mathbf{c_k J_0 b_j} J_2 \boxed{\mathbf{a'_j d_j} I_c}} \Rightarrow^{\texttt{1r5,2r6,2r9}} q_i q_l \mathbf{q'_l} a_j \mathbf{J_0} \boxed{\mathbf{a'_j} J_1 \mathbf{d_j} J_2 \boxed{c_k b_j I_c}} \Rightarrow^{\texttt{1r7,1r11}}$$

$$q_i q_l \mathbf{d_j} a_j a'_j \boxed{\mathbf{J_0} J_1 \mathbf{q'_l} J_2 \boxed{c_k b_j I_c}} \Rightarrow^{\texttt{1r8,2r13}} q_i q_l a_j a'_j J_0 \boxed{\mathbf{d_j} J_1 J_2 \boxed{\mathbf{q'_l} c_k b_j I_c}} \Rightarrow^{\texttt{2r11}}$$

$$q_i \mathbf{q_l} a_j a'_j J_0 \boxed{\mathbf{q'_l} J_1 J_2 \boxed{d_j c_k b_j I_c}} \Rightarrow^{\texttt{1r12}} q_i q'_l a_j a'_j J_0 \boxed{\mathbf{q_l} J_1 J_2 \boxed{d_j c_k b_j I_c}}$$

In that way, q_i is replaced by q_l and c_k is moved from region 1 into region 2. Notice that symbols a_j, b_j, a'_j, d_j, J_2, J_1, J_0 have returned to their original positions.

(ii) *There is no c_k in region 1:*

$$q_l q_l' \mathbf{a_j} a_j' J_0 \boxed{\mathbf{q_i} J_1 J_2 \boxed{b_j d_j I_c}} \Rightarrow^{\texttt{1r1}} q_i q_l q_l' a_j' J_0 \boxed{\mathbf{a_j} J_1 J_2 \boxed{\mathbf{b_j} d_j I_c}} \Rightarrow^{\texttt{2r1}}$$
$$q_i q_l q_l' \mathbf{a_j'} J_0 \boxed{\mathbf{b_j} J_1 \mathbf{J_2} \boxed{\mathbf{a_j} d_j I_c}} \Rightarrow^{\texttt{1r2,2r8}} q_i q_l q_l' \mathbf{b_j J_0} \boxed{\mathbf{a_j'} \mathbf{J_1} \mathbf{a_j} \boxed{\mathbf{J_2} d_j I_c}} \Rightarrow^{\texttt{1r3,1r4,2r15}}$$
$$q_i q_l q_l' a_j \mathbf{J_1} \boxed{\mathbf{J_0} \mathbf{b_j} J_2 \boxed{a_j' \mathbf{d_j} I_c}} \qquad \text{(A)}$$

Now there are two possibilities: we may either apply
a) rule `2r9` or
b) rule `1r2`.

Consider **case a)**:

$$q_i q_l q_l' a_j \mathbf{J_1} \boxed{\mathbf{J_0} \mathbf{b_j} J_2 \boxed{a_j' \mathbf{d_j} I_c}} \Rightarrow^{\texttt{1r5,2r9}} q_i q_l \mathbf{q_l'} a_j J_0 \boxed{\mathbf{J_1} \mathbf{d_j} J_2 \boxed{\mathbf{a_j'} b_j I_c}} \Rightarrow^{\texttt{1r11,2r7}}$$
$$q_i q_l \mathbf{d_j} a_j \mathbf{J_0} \boxed{\mathbf{a_j'} \mathbf{q_l'} \mathbf{J_2} \boxed{J_1 b_j I_c}} \Rightarrow^{\texttt{1r7,1r9,2r13 or 1r12}} q_i q_l a_j a_j' J_2 \boxed{\mathbf{J_0} \mathbf{d_j} \boxed{\mathbf{q_l'} J_1 b_j \mathbf{I_c}}} \text{ or}$$
$$q_i q_l' a_j a_j' J_2 \boxed{\mathbf{J_0} \mathbf{d_j} \mathbf{q_l} \boxed{J_1 b_j \mathbf{I_c}}}$$

After that rule `2r18` will eventually be applied, object I_c will be moved to region 1 and then applying rules `1s1,1s2` leads to an infinite computation.

It is easy to see that **case b)** also leads to an infinite computation:

$$q_i q_l q_l' a_j \mathbf{a_j'} \mathbf{J_1} \boxed{\mathbf{J_0} \mathbf{b_j} J_2 \boxed{a_j' d_j I_c}} \Rightarrow^{\texttt{1r2,1r5}} q_i q_l q_l' a_j \mathbf{b_j J_0} \boxed{\mathbf{J_1} \mathbf{a_j'} J_2 \boxed{\mathbf{a_j'} d_j I_c}}$$

Now there are two possibilities, we can apply rule `1r4` (and we get the previous configuration (A)) or rule `2r7`. Consider the last case:

$$q_i q_l q_l' a_j b_j \mathbf{J_0} \boxed{\mathbf{J_1} \mathbf{a_j'} J_2 \boxed{\mathbf{a_j'} d_j I_c}} \Rightarrow^{\texttt{1r7,2r7}} q_i q_l q_l' a_j a_j' b_j \mathbf{J_0} \boxed{\mathbf{J_0} \mathbf{a_j'} J_2 \boxed{J_1 d_j \mathbf{I_c}}}$$

Thus rule `2r18` will be applied eventually (that leads to an infinite computation). So in the case if we have not enough symbols c_k in region 1 it leads to an infinite computation.

" Decrement" -instruction:

(i) *There is some c_k in region 2:*

$$q_l q_l' a_l \mathbf{a_j} a_j' J_0 \boxed{\mathbf{q_i} J_1 J_2 \boxed{b_j c_k d_j I_c}} \Rightarrow^{\texttt{1r1}} q_i q_l q_l' a_l a_j' J_0 \boxed{\mathbf{a_j} J_1 J_2 \boxed{\mathbf{b_j} c_k d_j I_c}} \Rightarrow^{\texttt{2r1}}$$
$$q_i q_l q_l' a_l \mathbf{a_j'} J_0 \boxed{\mathbf{b_j} J_1 \mathbf{J_2} \boxed{\mathbf{a_j} c_k d_j I_c}} \Rightarrow^{\texttt{1r2,2r8}} q_i q_l q_l' a_l \mathbf{b_j J_0} \boxed{\mathbf{a_j'} \mathbf{J_1} \mathbf{a_j} \boxed{J_2 c_k \mathbf{d_j} I_c}}$$
$$\Rightarrow^{\texttt{1r3,1r4,2r10}} q_i q_l \mathbf{q_l'} a_l a_j \mathbf{J_1} \boxed{\mathbf{d_j} \mathbf{J_0} \mathbf{b_j} \boxed{J_2 \mathbf{c_k} \mathbf{a_j'} I_c}} \qquad \text{(B)}$$

$$\Rightarrow^{\texttt{1r5,1r11,2r3,2r5}} q_i\mathbf{q_l}\mathbf{d_j}a_la_jJ_0\boxed{\mathbf{q'_l}J_1c_k\mathbf{a'_j}\boxed{J_2b_jI_c}} \Rightarrow^{\texttt{1r6,1r12}}$$
$$q_iq'_l\mathbf{a_l}a_ja'_jJ_0\boxed{\mathbf{q_l}J_1c_k\mathbf{d_j}\boxed{\mathbf{J_2}b_jI_c}} \Rightarrow^{\texttt{1r1,2r14}} q_iq_lq'_la_ja'_jJ_0\boxed{\mathbf{a_l}J_1c_kJ_2\boxed{b_jd_jI_c}}$$

In the way described above, q_i is replaced by q_l and c_k is removed from region 2 to region 1. Notice that symbols a_j, b_j, a'_j, J_2, J_1, J_0 have returned to their original positions. Symbol d_j returns to region 2 in the first step of the simulation of the next instruction (the last step of the illustration).

(ii) *There is no c_k in region 2:*

We start with configuration (B) without c_k in region 2.

$$q_iq_l\mathbf{q'_l}a_la_j\mathbf{a'_j}\mathbf{J_1}\boxed{\mathbf{d_j}\mathbf{J_0}\mathbf{b_j}\boxed{J_2\mathbf{a'_j}I_c}} \Rightarrow^{\texttt{1r5,1r11,2r5}} q_i\mathbf{q_l}\mathbf{d_j}a_la_j\mathbf{b_j}J_0\boxed{\mathbf{q'_l}\mathbf{J_1}\mathbf{a'_j}\mathbf{a'_j}\boxed{J_2\mathbf{I_c}}}$$

Consider two objects a'_j in region 1. One object a'_j will be returned to the environment by rule `1r6` and other one will be moved to region 2 by rule `2r17`, that will cause an infinite computation.

So in the case if we have not symbol c_k in region 2 it leads to an infinite computation.

" Test for zero" -instruction:

q_i is replaced by q_l if there is no c_k in region 2, otherwise a'_j in region 1 exchanges with c_k in region 2 and the computation will never stop.

(i) *There is no c_k in region 2:*

$$q_lq'_la_l\mathbf{a_j}a'_jJ_0\boxed{\mathbf{q_i}J_1J_2\boxed{b_jd_jI_c}} \Rightarrow^{\texttt{1r1}} q_iq_lq'_la_la'_jJ_0\boxed{\mathbf{a_j}J_1J_2\boxed{\mathbf{b_j}d_jI_c}} \Rightarrow^{\texttt{2r1}}$$
$$q_iq_lq'_la_l\mathbf{a'_j}J_0\boxed{\mathbf{b_j}J_1\mathbf{J_2}\boxed{\mathbf{a_j}d_jI_c}} \Rightarrow^{\texttt{1r2,2r8}} q_iq_lq'_la_l\mathbf{b_j}\mathbf{J_0}\boxed{a'_j\mathbf{J_1}\mathbf{a_j}\boxed{J_2d_jI_c}} \Rightarrow^{\texttt{1r3,1r4}}$$
$$q_iq_lq'_la_la_j\mathbf{J_1}\boxed{\mathbf{b_j}a'_j\mathbf{J_0}\boxed{J_2\mathbf{d_j}I_c}} \Rightarrow^{\texttt{1r5,2r9}} q_iq_l\mathbf{q'_l}a_la_jJ_0\boxed{J_1a'_j\mathbf{d_j}\boxed{J_2b_jI_c}} \Rightarrow^{\texttt{1r11}}$$
$$q_i\mathbf{q_l}a_la_j\mathbf{d_j}J_0\boxed{J_1\mathbf{a'_j}\mathbf{q'_l}\boxed{J_2b_jI_c}} \Rightarrow^{\texttt{1r6,1r12}} q_iq'_l\mathbf{a_l}a_ja'_jJ_0\boxed{J_1\mathbf{q_l}\mathbf{d_j}\boxed{\mathbf{J_2}b_jI_c}} \Rightarrow^{\texttt{1r1,2r14}}$$
$$q_iq_lq'_la_ja'_jJ_0\boxed{\mathbf{a_l}J_1J_2\boxed{b_jd_jI_c}}$$

In this case, q_i is replaced by q_l. Notice that symbols a_j, b_j, a'_j, J_2, J_1, J_0 have returned to their original positions. Symbol d_j returns to region 2 in the first step of the simulation of the next instruction (the last step of the illustration).

(ii) *There is some c_k in region 2:*

$$q_l q_l' a_l \mathbf{a_j} a_j' J_0 \boxed{\mathbf{q_i} J_1 J_2 \boxed{b_j c_k d_j I_c}} \Rightarrow^{\texttt{1r1}} q_i q_l q_l' a_l a_j' J_0 \boxed{\mathbf{a_j} J_1 J_2 \boxed{\mathbf{b_j} c_k d_j I_c}} \Rightarrow^{\texttt{2r1}}$$

$$q_i q_l q_l' a_l \mathbf{a_j'} J_0 \boxed{\mathbf{b_j} J_1 \mathbf{J_2} \boxed{\mathbf{a_j} c_k d_j I_c}} \Rightarrow^{\texttt{1r2,2r8}} q_i q_l q_l' a_l \mathbf{b_j J_0} \boxed{\mathbf{a_j' J_1 a_j} \boxed{J_2 \mathbf{c_k} d_j I_c}}$$

$$\Rightarrow^{\texttt{1r3,1r4,2r2}} q_i a_j q_l q_l' a_l \mathbf{J_1} \boxed{\mathbf{b_j} c_k \mathbf{J_0} \boxed{J_2 a_j' \mathbf{d_j} I_c}} \Rightarrow^{\texttt{1r5,2r9}}$$

$$q_i q_l \mathbf{q_l'} a_j a_l J_0 \boxed{\mathbf{J_1 d_j} c_k \boxed{J_2 \mathbf{a_j'} b_j I_c}} \Rightarrow^{\texttt{1r11,2r4}} q_i \mathbf{q_l} a_j a_l \mathbf{d_j} J_0 \boxed{\mathbf{a_j' q_l'} c_k \boxed{J_1 J_2 b_j I_c}}$$

$$\Rightarrow^{\texttt{1r6,1r12}} q_i q_l' a_j a_j' \mathbf{a_l} J_0 \boxed{\mathbf{d_j q_l} c_k \boxed{J_1 \mathbf{J_2} b_j I_c}} \Rightarrow^{\texttt{1r1,2r14}}$$

$$q_i q_l q_l' a_j a_j' J_0 \boxed{\mathbf{a_l} c_k J_2 \boxed{J_1 b_j d_j I_c}}$$

Notice, that if object J_1 is situated in region 2, then it leads to infinite computation. Indeed, after returning of object a_l into environment (rule `1r3`) object J_0 will be moved to region 1 and rule `2r18` will be applied eventually. Now object I_c will bring objects c_k from the environment and the computation will never stop (rules `1s1`, `1s2`).

3. END.

$$\begin{aligned}
R_{1,f} = {} & \{\texttt{1f1} : (T_1, out; F_1, in), \texttt{1f2} : (F_1, out; F_2, in), \texttt{1f3} : (F_2, out; F_3, in), \\
& \texttt{1f4} : (F_3, out; F_4, in),\ \texttt{1f5} : (F_4, out : F_5, in),\ \texttt{1f6} : (T_2, out)\} \\
& \cup \{\texttt{1f7} : (M, out; T_2, in),\ \texttt{1f8} : (J_2, out; T_2, in)\}, \\
R_{2,f} = {} & \{\texttt{2f1} : (T_1, out; q_f, in), \texttt{2f2} : (q_f, out),\ \texttt{2f3} : (T_2, out; q_f, in)\} \\
& \cup \{\texttt{2f4} : (b_j, out; F_5, in),\ \texttt{2f5} : (d_j, out; F_5, in) \mid j \in I\} \\
& \cup \{\texttt{2f6} : (F_5, out)\}.
\end{aligned}$$

Once the register machine reaches the final state, q_f is in region 1. It takes both T_1 and T_2 to region 1, in either order. The duty of T_2 is to bring both J_2 and M to the environment (J_2 can be brought to the environment immediately, or after M if the latter immediately goes to the environment; the object M can oscillate for indefinite time, but we are interested in halting computations). T_2 starts a chain of exchanges, to give J_2 time to go to the environment, and then have F_5 remove symbols b_j, d_j from region 2.

We illustrate the end of computations as follows:

$$F_1 F_2 F_3 F_4 F_5 J_0 \boxed{\mathbf{q_f} J_1 J_2 \boxed{\mathbf{T_1} T_2 \mathbf{M} b_j d_j I_c}} \Rightarrow^{\texttt{2f1,2s1}}$$

$$\mathbf{F_1} F_2 F_3 F_4 F_5 J_0 \boxed{\mathbf{T_1} J_1 J_2 \mathbf{M} \boxed{\mathbf{q_f} T_2 b_j d_j I_c}} \Rightarrow^{\texttt{1f1,2f2,2s2}}$$

$$T_1 \mathbf{F_2} F_3 F_4 F_5 J_0 \boxed{\mathbf{F_1} J_1 J_2 \mathbf{q_f} \boxed{\mathbf{M T_2} b_j d_j I_c}} \Rightarrow^{\texttt{1f2,2f3,2s1}}$$

$$T_1F_1\mathbf{F_3}F_4F_5J_0\boxed{\mathbf{T_2F_2}J_1J_2\mathbf{M}\boxed{\mathbf{q_f}b_jd_jI_c}} \Rightarrow^{\texttt{1f3,1f6,2f2,2s2}}$$

$$T_1\mathbf{T_2}F_1F_2\mathbf{F_4}F_5J_0\boxed{\mathbf{F_3}J_1\mathbf{J_2}q_f\boxed{\mathbf{M}b_jd_jI_c}} \Rightarrow^{\texttt{1f4,1f8,2s1}}$$

$$T_1F_1F_2F_3\mathbf{F_5}J_0J_2\boxed{\mathbf{T_2F_4}J_1q_f\mathbf{M}\boxed{b_jd_jI_c}} \Rightarrow^{\texttt{1f5,1f6,2s2}}$$

$$T_1T_2F_1F_2F_3F_4J_0J_2\boxed{\mathbf{F_5}J_1q_f\boxed{\mathbf{Mb_j}d_jI_c}} \Rightarrow^{\texttt{2f4,2s1}}$$

$$T_1\mathbf{T_2}F_1F_2F_3F_4J_0J_2\boxed{\mathbf{b_j}J_1q_f\mathbf{M}\boxed{\mathbf{F_5}d_jI_c}} \Rightarrow^{\texttt{1r13,1f7,2f6}}$$

$$T_1F_1F_2F_3F_4\mathbf{b_j}J_0J_2M\boxed{\mathbf{T_2J_1}q_f\mathbf{F_5}\boxed{\mathbf{d_j}I_c}} \Rightarrow^{\texttt{1f6,1r4,2f5}}$$

$$T_1T_2F_1F_2F_3F_4J_0J_1J_2M\boxed{\mathbf{b_j}q_f\mathbf{d_j}\boxed{\mathbf{F_5}I_c}} \Rightarrow^{\texttt{1r13,1r14,2f5}}$$

$$T_1T_2F_1F_2F_3F_4b_jd_jJ_0J_2M\boxed{J_1q_f\mathbf{F_5}\boxed{I_c}}$$

We continue in this manner until all objects $b_j, d_j,\ j \in I$ from the elementary membrane 2 have been moved to the environment. Notice that the result in the elementary membrane 2 (multiset c_1^t) cannot be changed during phase END, as object J_2 now is situated in the environment. Thus, object a'_j cannot enter into region 2 by rule `2r15` and therefore cannot bring object c_k into region 2 by rule `2r6`. Recall that the counter automaton can only *increment* the first counter c_1, so all other computations of P system Π_1 cannot change the number of symbols c_1 in the elementary membrane. Thus, at the end of a terminating computation, in the elementary membrane there are the result (multiset c_1^t) and only the one additional object I_c. □

Lemma 1. *In the construction of Theorem 3, if an object b_j, during phase "RUN" returns to the environment after coming to region 1 from the environment, then the computation never stops.*

Lemma 2. *In the construction of Theorem 3, if an object d_j, during phase "RUN" returns to the environment after coming to region 1 from the environment, then the computation never stops.*

We skip the (very technical) proofs of these lemmas, in particular, for conciseness reasons.

5 Conclusion

In this paper we have proved the new results that any set of natural numbers containing zero generated by symport/antiport P systems with two membranes and minimal cooperation is finite (for both symport/antiport P systems and for purely symport P systems) and one additional object in the output membrane allows symport/antiport P systems with two membranes and minimal cooperation generate any recursively enumerable sets of natural numbers without zero.

Thus we improved the result from [1] for symport/antiport P systems with two membranes and minimal cooperation from three objects down to one object and showed the optimality of this result.

The same question for symport P systems with two membranes and minimal cooperation (is only one additional object in the output membrane sufficient in order to get universality?) is still open. We conjecture that the same is true also for symport P systems with two membranes and minimal cooperation, i.e., one additional object in the output membrane allows generate any recursively enumerable sets of natural numbers without zero. The question about precise characterization of computational power of symport/antiport P systems with two membranes and minimal cooperation is still open.

References

1. A. Alhazov, R. Freund, Yu. Rogozhin: Computational Power of Symport/Antiport: History, Advances, and Open Problems. Membrane Computing, International Workshop, WMC 2005, Vienna, 2005, Revised Selected and Invited Papers (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), *Lecture Notes in Computer Science 3850* (2006) 1–30.
2. A. Alhazov, R. Freund, Yu. Rogozhin: Some Optimal Results on Communicative P Systems with Minimal Cooperation. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, Eds.), Fénix Editora, Sevilla, (2005) 23–36.
3. A. Alhazov, M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan: Communicative P Systems with Minimal Cooperation. Membrane Computing, International Workshop, WMC 2004, Milan, 2004, Revised Selected and Invited Papers (G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, Eds.) *Lecture Notes in Computer Science 3365* (2005) 161–177.
4. A. Alhazov, Yu. Rogozhin: Minimal Cooperation in Symport/Antiport P Systems with One Membrane. *Third Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, Eds.) RGNC TR *01/2005*, University of Seville, Fénix Editora, Sevilla (2005) 29–34.
5. A. Alhazov, Yu. Rogozhin, S. Verlan: Symport/Antiport Tissue P Systems with Minimal Cooperation. *Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, Eds.), Fénix Editora, Sevilla (2005) 37–52.
6. F. Bernardini, M. Gheorghe: On the Power of Minimal Symport/Antiport. *Workshop on Membrane Computing*, WMC 2003 (A. Alhazov, C. Martín-Vide, Gh. Păun, Eds.), Tarragona, 2003, TR *28/03*, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona (2003) 72–83.
7. F. Bernardini, A. Păun: Universality of Minimal Symport/Antiport: Five Membranes Suffice. Membrane Computing, International Workshop, WMC 2003, Tarragona, Revised Papers (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), *Lecture Notes in Computer Science 2933* (2004) 43–45.
8. R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae 49*, 1–3 (2002) 81–102.
9. R. Freund, M. Oswald: P Systems with Activated/Prohibited Membrane Channels. Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Argeş,

2002. Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science 2597* (2003) 261–268.
10. R. Freund, A. Păun: Membrane Systems with Symport/Antiport: Universality Results. Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Argeş, 2002. Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science 2597* (2003) 270–287.
11. P. Frisco: About P Systems with Symport/Antiport. *Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, Eds), TR *01/2004*, Research Group on Natural Computing, University of Seville (2004) 224–236.
12. P. Frisco, H.J. Hoogeboom: P Systems with Symport/Antiport Simulating Counter Automata. *Acta Informatica 41*, 2–3 (2004) 145–170.
13. P. Frisco, H.J. Hoogeboom: Simulating Counter Automata by P Systems with Symport/Antiport. Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Argeş, 2002. Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science 2597* (2003) 288–301.
14. L. Kari, C. Martín-Vide, A. Păun: On the Universality of P Systems with Minimal Symport/Antiport Rules. Aspects of Molecular Computing - Essays dedicated to Tom Head on the occasion of his 70th birthday, *Lecture Notes in Computer Science 2950* (2004) 254–265.
15. M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan: About P Systems with Minimal Symport/Antiport Rules and Four Membranes. *Fifth Workshop on Membrane Computing (WMC5)*, (G. Mauri, Gh. Păun, C. Zandron, Eds.), Universitá di Milano-Bicocca, Milan (2004) 283–294.
16. C. Martín-Vide, A. Păun, Gh. Păun: On the Power of P Systems with Symport Rules, *Journal of Universal Computer Science 8*, 2 (2002) 317–331.
17. M.L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey (1967).
18. A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing 20* (2002) 295–305.
19. Gh. Păun: Computing with Membranes. *Journal of Computer and Systems Science 61* (2000) 108–143.
20. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag (2002).
21. Gh. Păun: Further Twenty Six Open Problems in Membrane Computing (2005). *Third Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, Eds.) RGNC TR *01/2005*, University of Seville, Fénix Editora, Sevilla (2005) 249–262.
22. Gh.Păun: 2006 Research Topics in Membrane Computing. *Fourth Brainstorming Week on Membrane Computing*, vol. *1* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, Eds.), Fénix Edit., Sevilla (2006), 235–251.
23. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages* (3 volumes). Springer-Verlag, Berlin (1997).
24. Gy. Vaszil: On the Size of P Systems with Minimal Symport/Antiport. *Fifth Workshop on Membrane Computing (WMC5)* (G. Mauri, Gh. Păun, C. Zandron, Eds.), Universitá di Milano-Bicocca, Milan (2004) 422–431.
25. S. Verlan: Optimal Results on Tissue P Systems with Minimal Symport/ Antiport. Presented at *EMCC meeting*, Lorentz Center, Leiden (2004).
26. S. Verlan: Tissue P Systems with Minimal Symport/Antiport. Developments in Language Theory, DLT 2004 (C.S. Calude, E. Calude, M.J. Dinneen, Eds), *Lecture Notes in Computer Science 3340*, Springer-Verlag, Berlin (2004) 418–430.
27. P Systems Webpage, `http://psystems.disco.unimib.it`