Some Optimal Results on Symport/Antiport P Systems with Minimal Cooperation

Artiom Alhazov[1,2], Rudolf Freund[3], Yurii Rogozhin[1]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: {artiom,rogozhin}@math.md
[2] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: artiome.alhazov@estudiants.urv.es
[3] Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
E-mail: rudi@emcc.at

Summary. We prove that two classes of symport/antiport P systems with two membranes and with minimal cooperation, namely P systems with symport/antiport rules of size one and P systems with symport rules of size two, are computationally complete: (modulo the terminal alphabet) they generate all recursively enumerable sets of vectors of nonnegative integers. On the other hand, it is known that these systems with one membrane cannot be universal. Hence, the results we prove are optimal.

1 Introduction

P systems with *symport/antiport* were introduced in [14], and they use one of the most important features of membrane systems – the communication. These systems have two types of rules: symport rules, where several objects go together from one membrane to another one, and antiport rules, where several objects from two membranes are exchanged. These operations are very powerful, i.e., P systems with symport/antiport rules have universal computational power with only one membrane, e.g., see [6], [7].

Symport/antiport P systems with minimal cooperation, i.e., systems where symport rules move only one object and antiport rules move only two objects across the same membrane in different directions, are universal. The first proof of this result can be found in [3], and the corresponding system has nine membranes.

This first result was improved by reducing the number of membranes to six [10], five [4], and four [7, 11], and finally G. Vaszil [16] showed that three membranes

are sufficient to generate all recursively enumerable sets of numbers (but his proof had one disadvantage: the output membrane contained five additional symbols). In [1], another proof of this latest result was given where the output membrane did not contain superfluous symbols.

Symport P systems with minimal cooperation, i.e., P systems only having symport rules and only moving one or two objects, were shown to be universal with four membranes in [9]. In [1], this result was improved down to three membranes.

In this paper we prove that symport/antiport P systems with minimal cooperation, namely P systems with symport/antiport rules of size one or P systems with symport rules of size two are computationally complete with only two membranes: (modulo the terminal alphabet) they generate all recursively enumerable sets of vectors of nonnegative integers. On the other hand, it is known that systems with such rules in only one membrane cannot be universal, see [8, 17]. Hence, the results we are going to prove in this paper are already optimal.

Our proofs of both results are based on a simulation of counter automata (or register machines, see [12] and also [5]), which proof idea was also used in [3], [4], [6], and [10].

The questions how to obtain these optimal results without using the terminal objects are still open. However, for symport/antiport *tissue* P systems with minimal cooperation this problem has successfully been solved in [2], i.e., it was proved that two cells are enough in order to generate all recursively enumerable sets of numbers.

2 Basic Notions

A non-deterministic *counter automaton* is a 5-tuple $M = (d, Q, q_0, q_f, P)$, 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 $F = \{1, ..., f\}$;
- $q_0 \in Q$ is the initial state;
- $q_f \in Q$ is the final state;
- P is a finite set of instructions of the following form:
  1. $(q_i \rightarrow q_l, k+)$, 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. $(q_i \rightarrow q_l, k-)$, with $i, l \in F,\ i \neq f,\ k \in D$ "decrement" instruction). If the value of counter c_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 c_k is zero) the computation is blocked in state q_i.
  3. $(q_i \rightarrow q_l, k = 0)$, 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. *Stop*. 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 equal to zero. The result of the computation of a counter automaton is the value of the first counter 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 nonnegative integers.

A *P system with symport/antiport (symport)* is a construct

$$\Pi = (O, T, \mu, w_1, \ldots, w_k, E, R_1, \ldots, R_k, i_0),$$

where:

1. O is a finite alphabet of symbols called *objects*;
2. $T \subseteq Q$ is a set of terminal objects;
3. μ is a *membrane structure* consisting of k membranes that are labelled in a one-to-one manner by $1, 2, \ldots, k$;
4. $w_i \in O^*$, for each $1 \leq i \leq k$, is a finite multiset (i.e., a multiset where elements are present in finite number of copies) of objects associated with the region i (delimited by membrane i);
5. $E \subseteq O$ is the set of objects that appear in the environment in infinite numbers of copies;
6. R_i, for each $1 \leq i \leq k$, is a finite set of symport / antiport rules associated with the region i; these rules are of the forms (x, in), (y, out), and $(y, out; x, in)$, respectively, where $x, y \in O^*$ (for symport P systems, R_i consists rules of the forms (x, in) and (y, out) only);
7. i_0 is the label of an elementary membrane of μ that identifies the corresponding output region.

A symport/antiport (symport) P system 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 (symport) 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 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.

As usual, a computation in a symport/antiport (symport) P system 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 (symport) 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 it reaches a configuration where no rule can be applied. The result of a successful computation is a natural number that is obtained by counting the *terminal* objects present in region i_0. Given a P system Π, the set of natural numbers computed in this way by Π is denoted by $N(\Pi)_T$. If the multiplicity of each object is counted separately, then a vector of natural numbers is obtained, denoted by $Ps(\Pi)_T$, see [15].

By $NOP_m(sym_r, anti_t)_T$ ($NOP_m(sym_r)_T$) we denote the family of sets of natural numbers (non-negative integers) that are generated by a P system with symport/antiport (symport) having at most $m > 0$ membranes, symport rules of size at most $r \geq 0$, and antiport rules of size at most $t \geq 0$. The size of a symport rule (x, in) or (x, out) is given by $|x|$, while the size of an antiport rule $(y, out; x, in)$ is given by $max\{|x|, |y|\}$. By NRE we denote the family of recursively enumerable sets of non-negative numbers. If we replace numbers by vectors, then in the notations of this paragraph N is replaced by Ps.

3 Main Results

We first show that two membranes are enough to obtain computational completeness with symport/antiport rules of minimal size provided we only count the terminal objects.

Theorem 1. $NOP_2(sym_1, anti_1)_T = NRE$.

Proof. 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 now construct the P system Π_1 as follows:

$$
\begin{aligned}
\Pi_1 &= (O, T, [_1\, [_2\ \ ]_2\,]_1, w_1, w_2, E, R_1, R_2, 2),\\
T &= \{c_1\},\\
O &= E \cup \{b_j, b'_j \mid j \in I\} \cup \{\#_1, F, I_c\},\\
E &= Q \cup \{a_j, a'_j, a''_j \mid j \in I\} \cup C \cup \{F\},\\
C &= \{c_i \mid 1 \leq i \leq d\},\\
w_1 &= q_0 I_c \#_1 \#_1,
\end{aligned}
$$

$$w_2 = \prod_{j \in I} b_j \prod_{j \in I} b'_j,$$
$$R_i = R_{i,s} \cup R_{i,r} \cup R_{i,f}, \quad i \in \{1, 2\}.$$

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

1. the simulation of instructions of the counter automaton;
2. the termination of 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 also use the following idea realized by phase "START" below: from the environment, we bring symbols c_k into region 1 all time during the computation. This process may only be stopped if all stages finish correctly; otherwise, the computation will never stop.

We split our proof into several parts which depend on the logical separation of the behavior of the system. We will present rules and 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 (stage 1);
2. RUN (stage 1);
3. END (stage 2).

1. START.

$$R_{1,s} = \{\texttt{1s1} : (I_c, in),\ \texttt{1s2} : (I_c, out; c_k, in) \mid k \in D\},$$
$$R_{2,s} = \emptyset.$$

Symbol I_c brings "sufficiently many" symbols c_k from environment into region 1.

While I_c is bringing symbols c_k into region 1 (rules `1s1,1s2`), instructions $(j : q_i \to q_l, k\gamma)$, $\gamma \in \{+, -, = 0\}$, of the counter automaton are simulated as shown in the following.

2. RUN.

$$\begin{aligned} R_{1,r} = {} & \{\texttt{1r1} : (q_i, out; a_j, in) \mid (j : q_i \to q_l, k\gamma) \in P, \\ & \gamma \in \{+, -, = 0\}, k \in D\} \\ & \cup \{\texttt{1r2} : (b_j, out; a'_j, in),\ \texttt{1r3} : (a_j, out; b_j, in), \\ & \texttt{1r4} : (\#_1, out; b_j, in) \mid j \in I\} \\ & \cup \{\texttt{1r5} : (a'_j, out; a''_j, in) \mid j \in I_+ \cup I_-\} \\ & \cup \{\texttt{1r6} : (\#_1, out; \#_1, in)\} \end{aligned}$$

$$\cup \{\texttt{1r7} : (b'_j, out; a''_j, in),\ \texttt{1r8} : (a'_j, out; b'_j, in),$$
$$\texttt{1r9} : (\#_1, out; b'_j, in) \mid j \in I_{=0}\}$$
$$\cup \{\texttt{1r10} : (a''_j, out, q_l, in) \mid (j : q_i \to q_l, k\gamma) \in P,$$
$$\gamma \in \{+, -, = 0\}, k \in D\},$$

$$\begin{aligned}
R_{2,r} = {} & \{\texttt{2r1} : (b_j, out; a_j, in) \mid j \in I\} \\
& \cup \{\texttt{2r2} : (a_j, out; c_k, in) \mid (j : q_i \to q_l, k+) \in P, k \in D\} \\
& \cup \{\texttt{2r3} : (a'_j, in) \mid j \in I_+\} \\
& \cup \{\texttt{2r4} : (a'_j, out; b_j, in) \mid j \in I_+ \cup I_-\} \\
& \cup \{\texttt{2r5} : (a_j, out) \mid j \in I_- \cup I_{=0}\} \\
& \cup \{\texttt{2r6} : (c_k, out; a'_j, in) \mid (j : q_i \to q_l, k\gamma) \in P, \\
& \quad \gamma \in \{-, = 0\}, k \in D\} \\
& \cup \{\texttt{2r7} : (b'_j, out; b_j, in),\ \texttt{2r8} : (b'_j, in) \mid j \in I_{=0}\}.
\end{aligned}$$

The parts of computations illustrated in the following describe different stages of the evolution of the P system given in the corresponding theorem. 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 $\Rightarrow$.

"Increment" instruction:

$$\mathbf{a_j} a'_j a''_j q_l \boxed{\mathbf{q_i} c_k \#_1 \#_1 \boxed{b_j}} \Rightarrow^{\texttt{1r1}} a'_j a''_j q_i q_l \boxed{\mathbf{a_j} c_k \#_1 \#_1 \boxed{\mathbf{b_j}}} \Rightarrow^{\texttt{2r1}}$$
$$\mathbf{a'_j} a''_j q_i q_l \boxed{\mathbf{b_j c_k} \#_1 \#_1 \boxed{\mathbf{a_j}}} \Rightarrow^{\texttt{1r2,2r2}} \mathbf{b_j} a''_j q_i q_l \boxed{\mathbf{a_j a'_j} \#_1 \#_1 \boxed{c_k}}$$

Now there are two possibilities: we may either apply
a) rule 1r5 or
b) rule 2r3.

It is easy to see that case a) leads to an infinite computation:

$$\mathbf{b_j a''_j} q_i q_l \boxed{\mathbf{a_j a'_j} \#_1 \#_1 \boxed{c_k}} \Rightarrow^{\texttt{1r5,1r3}} a_j \mathbf{a'_j} q_i \mathbf{q_l} \boxed{\mathbf{b_j a''_j} \#_1 \#_1 \boxed{c_k}} \Rightarrow^{\texttt{1r2,1r10}}$$
$$a_j \mathbf{b_j} q_i \mathbf{a''_j} \boxed{\mathbf{a'_j q_l} \#_1 \#_1 \boxed{c_k}}$$

After that rule 1r4 will eventually be applied, object $\#_1$ will be moved to the environment and applying rule 1r6 leads to an infinite computation.

Now let us consider case b):

$$\mathbf{b_j}a''_j q_i q_l \boxed{\mathbf{a_j}\mathbf{a'_j}\#_1\#_1\boxed{c_k}} \Rightarrow^{\texttt{1r3,2r3}} a_j a''_j q_i q_l \boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a'_j}c_k}}$$

We cannot apply rule `1r2` as this leads to an infinite computation (see above). Hence, rule `2r4` has to be applied:

$$a_j a''_j q_i q_l \boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a'_j}c_k}} \Rightarrow^{\texttt{2r4}} a_j \mathbf{a''_j} q_i q_l \boxed{\mathbf{a'_j}\#_1\#_1\boxed{b_j c_k}} \Rightarrow^{\texttt{1r5}}$$
$$a_j a'_j q_i \mathbf{q_l} \boxed{\mathbf{a''_j}\#_1\#_1\boxed{b_j c_k}} \Rightarrow^{\texttt{1r10}} a_j a'_j a''_j q_i \boxed{\mathbf{q_l}\#_1\#_1\boxed{b_j c_k}}$$

In that way, q_i is replaced by q_l and c_k is moved from region 1 into region 2.

"Decrement" instruction:

$$\mathbf{a_j}a'_j a''_j q_l \boxed{\mathbf{q_i}\#_1\#_1\boxed{b_j c_k}} \Rightarrow^{\texttt{1r1}} a'_j a''_j q_i q_l \boxed{\mathbf{a_j}\#_1\#_1\boxed{\mathbf{b_j}c_k}} \Rightarrow^{\texttt{2r1}}$$
$$\mathbf{a'_j}a''_j q_i q_l \boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j}c_k}} \Rightarrow^{\texttt{1r2,2r5}} \mathbf{b_j}a''_j q_i q_l \boxed{\mathbf{a_j}\mathbf{a'_j}\#_1\#_1\boxed{c_k}} \Rightarrow^{\texttt{1r3,2r6}}$$
$$a_j a''_j q_i q_l \boxed{\mathbf{b_j}c_k\#_1\#_1\boxed{\mathbf{a'_j}}} \Rightarrow^{\texttt{2r4}} a_j \mathbf{a''_j} q_i q_l \boxed{\mathbf{a'_j}c_k\#_1\#_1\boxed{b_j}} \Rightarrow^{\texttt{1r5}}$$
$$a_j a'_j q_i \mathbf{q_l} \boxed{\mathbf{a''_j}c_k\#_1\#_1\boxed{b_j}} \Rightarrow^{\texttt{1r10}} a_j a'_j a''_j q_i \boxed{\mathbf{q_l}c_k\#_1\#_1\boxed{b_j}}$$

In the way described above, q_i is replaced by q_l and c_k is removed from region 2 to region 1.

"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:*

$$\mathbf{a_j}a'_j a''_j q_l \boxed{\mathbf{q_i}\#_1\#_1\boxed{b_j b'_j}} \Rightarrow^{\texttt{1r1}} a'_j a''_j q_i q_l \boxed{\mathbf{a_j}\#_1\#_1\boxed{\mathbf{b_j}b'_j}} \Rightarrow^{\texttt{2r1}}$$
$$a'_j a''_j q_i q_l \boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j}b'_j}}$$

Now there are two possibilities: we apply either

a) rule `2r7` or

b) rule `1r2`.

It is easy to see that the case a) leads to an infinite computation:

$$a'_j a''_j q_i q_l \boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j}\mathbf{b'_j}}} \Rightarrow^{\texttt{2r7,2r5}} a'_j a''_j q_i q_l \boxed{\mathbf{a_j}\mathbf{b'_j}\#_1\#_1\boxed{\mathbf{b_j}}} \Rightarrow^{\texttt{2r1,2r8}}$$
$$a'_j a''_j q_i q_l \boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j}\mathbf{b'_j}}} \Rightarrow^{\texttt{2r7,2r5}} \cdots \Rightarrow^{\texttt{2r1,2r8}} \mathbf{a'_j}a''_j q_i q_l \boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j}b'_j}}$$
$$\Rightarrow^{\texttt{1r2,2r5}} \mathbf{b_j}a''_j q_i q_l \boxed{\mathbf{a_j}a'_j\#_1\#_1\boxed{b'_j}} \Rightarrow^{\texttt{1r3}} a_j a''_j q_i q_l \boxed{\mathbf{b_j}a'_j\#_1\#_1\boxed{b'_j}}$$

Again there are two possibilities: we can apply either

c) rule `1r2` or

d) rule `2r7`.

The case c) leads to an infinite computation (rules `1r4` and `1r6`).

Now let us consider case d):

$$a_j a''_j q_i q_l \boxed{\mathbf{b_j} a'_j \#_1 \#_1 \boxed{\mathbf{b'_j}}} \Rightarrow^{\texttt{2r7}} a_j \mathbf{a''_j} q_i q_l \boxed{\mathbf{b'_j} a'_j \#_1 \#_1 \boxed{b_j}} \Rightarrow^{\texttt{1r7}}$$
$$a_j \mathbf{b'_j} q_i \mathbf{q_l} \boxed{\mathbf{a''_j a'_j} \#_1 \#_1 \boxed{b_j}} \Rightarrow^{\texttt{1r8,1r10}} a_j a'_j a''_j q_i \boxed{\mathbf{q_l b'_j} \#_1 \#_1 \boxed{b_j}}$$

There are two possibilities: we can apply either

e) rule `1r7` or

f) rule `2r8`.

The case e) leads to infinite computation (rules `1r9` and `1r6`).

In case f), the object b'_j comes back to region 2.

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

Consider again case d):

$$a_j a''_j q_i q_l \boxed{\mathbf{b_j a'_j} \#_1 \#_1 \boxed{\mathbf{b'_j c_k}}} \Rightarrow^{\texttt{2r7,2r6}} a_j \mathbf{a''_j} q_i q_l \boxed{\mathbf{b'_j} c_k \#_1 \#_1 \boxed{a'_j b_j}} \Rightarrow^{\texttt{1r7}}$$
$$a_j \mathbf{b'_j} q_i \mathbf{q_l} \boxed{\mathbf{a''_j} c_k \#_1 \#_1 \boxed{a'_j b_j}} \Rightarrow^{\texttt{1r9,1r10}} a_j a''_j \#_1 q_i \boxed{\mathbf{q_l b'_j} c_k \#_1 \boxed{a'_j b_j}}$$

Now the application of rule `1r6` leads to an infinite computation. Hence, we model correctly the "test for zero" instruction.

3. END.

$$R_{1,f} = \{\texttt{1f1} : (q_f, out; F, in)\},$$
$$R_{2,f} = \{\texttt{2f1} : (F, in),\ \texttt{2f2} : (F, out; I_c, in)\}.$$

If a successful computation of the counter automaton is correctly simulated, then q_f will appear in region 1 and F will appear in region 2 (rules `1f1` and `2f1`). After that the object I_c will be moved to region 2 (rule `2f2`); thus, the computation will be stopped. □

A "dual" class of systems $OP(sym_1, anti_1)$ is the class $OP(sym_2)$ where two objects are moved across the membrane in the same direction rather than in the opposite ones. We now prove a similar result for this class.

Theorem 2. $NOP_2(sym_2)_T = NRE$.

Proof. As in the proof of Theorem 1 we simulate a counter automaton $M = (d, Q, q_0, q_f, P)$ which starts with empty counters. Again we suppose that all instructions from P are labelled in a one-to-one manner with elements of $\{1, \ldots, n\} = I$ and that 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. Moreover, we define $I' = I \setminus \{n\}$ and $Q' = Q \setminus \{q_0\}$. We also suppose that there is only one instruction with initial state q_0 (labelled with number 1) and that the counter automaton empties all counters except for the first counter before stopping with the instruction labelled by n (which we suppose to be the only one with the *Stop* instruction).

We construct the P system Π_2 as follows:

$$
\begin{aligned}
\Pi_2 &= (O, T, [_1\ [_2\ \ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2),\\
O &= E \cup \{\#_1, \#_2, \$, f\} \cup Q \cup \{b_j, g_j \mid j \in I\} \cup \{g'_j \mid j \in I'\},\\
T &= \{c_1\},\\
E &= \{a_j, a'_j, d_j, d'_j \mid j \in I\} \cup C,\\
C &= \{c_i \mid 1 \le i \le d\},\\
w_1 &= \#_2 \$ f q_0 a_1 \prod_{j \in I} b_j,\\
w_2 &= \#_1 \prod_{q_i \in Q'} q_i \prod_{j \in I} g_j \prod_{j \in I'} g'_j,\\
R_i &= R_{i,s} \cup R_{i,r} \cup R_{i,f},\ i \in \{1, 2\}.
\end{aligned}
$$

The functioning of this system again may be split into two stages:

1. the simulation of instructions of the counter automaton;
2. the termination of the computation.

We code the counter automaton as follows: The environment will hold the current state of the automaton, represented by a symbol $q_i \in Q$, membrane 1 will hold the value of all counters, represented by the number of occurrences of symbols c_k, $k \in D$, where $D = \{1, ..., d\}$. We also use the following idea realized by phase START below: in our system we have a symbol $\#_2$ moving from the environment to membrane 1 and back in an infinite loop. This loop can only be stopped if all stages have completed correctly. Otherwise, the computation will never stop.

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 that we present is the union of all these parts.

The rules R_i again are given by three phases:

1. START (stage 1);
2. RUN (stage 1);
3. END (stage 2).

1. START.

$$R_{1,s} = \{\texttt{1s1} : (\#_2, out),\ \texttt{1s2} : (\#_2, in)\},$$
$$R_{2,s} = \emptyset.$$

Notice that system Π_2 begins its functioning by applying rule `1s1` and moving objects $q_0 a_1$ to region 2 (see phase RUN below). Thus system Π_2 starts to simulate the counter automaton M.

2. RUN.

$$\begin{aligned}
R_{1,r} = {} & \{\texttt{1r1} : (q_i a_j, in) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{+, -, = 0\}, k \in D\} \\
& \cup \{\texttt{1r2} : (b_j g_j, out) \mid j \in I_+ \cup I_{=0}\} \\
& \cup \{\texttt{1r3} : (c_k b_j, in) \mid (j : q_i \to q_l, k+) \in P, k \in D\} \\
& \cup \{\texttt{1r4} : (g_j c_k, out) \mid (j : q_i \to q_l, k-) \in P, k \in D\} \\
& \cup \{\texttt{1r5} : (a'_j g_j, in) \mid j \in I'\} \\
& \cup \{\texttt{1r6} : (\#_1, out), \texttt{1r7} : (\#_1, in)\} \\
& \cup \{\texttt{1r8} : (d_j b_j, in) \mid j \in I_{=0}\} \\
& \cup \{\texttt{1r9} : (d_j c_k, out) \mid (j : q_i \to q_l, k = 0) \in P, k \in D\} \\
& \cup \{\texttt{1r10} : (a'_j q_l, out) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{+, -\}, k \in D\} \\
& \cup \{\texttt{1r11} : (a'_j g'_j, out),\ \texttt{1r12} : (d'_j g'_j, in),\ \texttt{1r13} : (d'_j, out) \mid j \in I_{=0}\} \\
& \cup \{\texttt{1r14} : (d_j q_l, out) \mid (j : q_i \to q_l, k = 0) \in P, k \in D\}, \\
R_{2,r} = {} & \{\texttt{2r1} : (a_j b_j, in) \mid j \in I'\} \\
& \cup \{\texttt{2r2} : (q_i, in) \mid q_i \in Q\} \\
& \cup \{\texttt{2r3} : (b_j g_j, out) \mid j \in I'\} \\
& \cup \{\texttt{2r4} : (a'_j \$, in) \mid j \in I\} \\
& \cup \{\texttt{2r5} : (\#_1 \$, out)\} \\
& \cup \{\texttt{2r6} : (a'_j g_j, in) \mid j \in I'\} \\
& \cup \{\texttt{2r7} : (a'_j q_l, out) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{+, -\}, k \in D\} \\
& \cup \{\texttt{2r8} : (a'_j g'_j, out) \mid j \in I_{=0}\} \\
& \cup \{\texttt{2r9} : (d'_j g'_j, in) \mid j \in I_{=0}\} \\
& \cup \{\texttt{2r10} : (d'_j q_l, out) \mid (j : q_i \to q_l, k = 0) \in P, k \in D\}
\end{aligned}$$

Notice that the starting configuration of Π_2 corresponds to the result of the first step of the simulation of the starting instruction (`1r1` is "already made").

"Increment" instruction:

$$a'_j c_k \mathbf{q_i a_j} \boxed{b_j \$ \boxed{g_j q_l \#_1}} \Rightarrow^{\texttt{1r1}} a'_j c_k \boxed{\mathbf{q_i a_j b_j} \$ \boxed{g_j q_l \#_1}} \Rightarrow^{\texttt{2r1,2r2}}$$
$$a'_j c_k \boxed{\$ \boxed{q_i a_j \mathbf{b_j g_j} q_l \#_1}} \Rightarrow^{\texttt{2r3}} a'_j c_k \boxed{\mathbf{b_j g_j} \$ \boxed{q_i a_j q_l \#_1}} \Rightarrow^{\texttt{1r2}}$$

$$\mathbf{c_k b_j g_j a'_j}\,[\,\$\,[\,q_i a_j q_l \#_1\,]\,] \Rightarrow^{\texttt{1r3,1r5}} [\,c_k b_j g_j a'_j \$\,[\,q_i a_j q_l \#_1\,]\,]$$

Now there are two variants of computations (depending on the application of rule `1r2` or rule `2r6`):

a) Applying rule `1r2`:

$$c_k a'_j\,[\,c_k \mathbf{b_j g_j a'_j} \$\,[\,q_i a_j q_l \#_1\,]\,] \Rightarrow^{\texttt{1r2,2r4}}$$
$$\mathbf{c_k b_j g_j a'_j}\,[\,c_k\,[\,q_i a_j \mathbf{a'_j q_l} \$ \#_\mathbf{1}\,]\,] \Rightarrow^{\texttt{1r5,1r3,2r5,2r7}}$$
$$[\,b_j g_j a'_j a'_j q_l c_k c_k \$ \#_1\,[\,q_i a_j\,]\,] \quad \cdots$$

After that application of rules `1r6` and `1r7` leads to infinite computation.

b) Applying rule `2r6`:

$$[\,c_k b_j \mathbf{g_j a'_j} \$\,[\,q_i a_j q_l \#_1\,]\,] \Rightarrow^{\texttt{2r6}} [\,c_k b_j \$\,[\,q_i a_j g_j \mathbf{a'_j q_l} \#_1\,]\,] \Rightarrow^{\texttt{2r7}}$$
$$[\,\mathbf{a'_j q_l} c_k b_j \$\,[\,q_i a_j g_j \#_1\,]\,] \Rightarrow^{\texttt{1r10}} a'_j q_l\,[\,c_k b_j \$\,[\,q_i a_j g_j \#_1\,]\,]$$

q_i is replaced by q_l and c_k is moved into region 1.

"Decrement" instruction:

$$a'_j \mathbf{q_i a_j}\,[\,c_k b_j \$\,[\,g_j q_l \#_1\,]\,] \Rightarrow^{\texttt{1r1}} a'_j\,[\,c_k \mathbf{q_i a_j b_j} \$\,[\,g_j q_l \#_1\,]\,] \Rightarrow^{\texttt{2r1,2r2}}$$
$$a'_j\,[\,c_k \$\,[\,q_i a_j \mathbf{b_j g_j} q_l \#_1\,]\,] \Rightarrow^{\texttt{2r3}} a'_j\,[\,\mathbf{c_k g_j} b_j \$\,[\,q_i a_j q_l \#_1\,]\,] \Rightarrow^{\texttt{1r4}}$$
$$c_k \mathbf{g_j a'_j}\,[\,b_j \$\,[\,q_i a_j q_l \#_1\,]\,] \Rightarrow^{\texttt{1r5}} c_k\,[\,b_j g_j a'_j \$\,[\,q_i a_j q_l \#_1\,]\,]$$

Now there are two variants of computations (depending on the application of rule `1r4` or rule `2r6`).

c) Applying rule `1r4`:

$$a'_j c_k\,[\,b_j \mathbf{c_k g_j a'_j} \$\,[\,q_i a_j q_l \#_1\,]\,] \Rightarrow^{\texttt{1r4,2r4}} \mathbf{a'_j g_j} c_k c_k\,[\,b_j\,[\,q_i a_j \mathbf{a'_j q_l} \$ \#_1\,]\,] \Rightarrow^{\texttt{1r5,2r5,2r7}}$$
$$c_k c_k\,[\,a'_j a'_j g_j q_l b_j \$ \#_1\,[\,q_i a_j\,]\,] \quad \cdots$$

After that the application of rules `1r6` and `1r7` leads to an infinite computation.

d) Applying rule `2r6`:

$$c_k\,[\,b_j \mathbf{g_j a'_j} \$\,[\,q_i a_j q_l \#_1\,]\,] \Rightarrow^{\texttt{2r6}} c_k\,[\,b_j \$\,[\,q_i a_j g_j \mathbf{a'_j q_l} \#_1\,]\,] \Rightarrow^{\texttt{2r7}}$$
$$c_k\,[\,\mathbf{a'_j q_l} b_j \$\,[\,q_i a_j g_j \#_1\,]\,] \Rightarrow^{\texttt{1r10}} c_k a'_j q_l\,[\,b_j \$\,[\,q_i a_j g_j \#_1\,]\,]$$

In that way, q_i is replaced by q_l and c_k is removed from region 1.

"Test for zero" instruction:

q_i is replaced by q_l if there is no c_k in region 1 (case e)), otherwise the computation will never stop (case f)).

Case e):

$$a'_j d_j d'_j \mathbf{q_i a_j} [\, b_j \$ [\, g_j g'_j q_l \#_1]] \Rightarrow^{1r1} a'_j d_j d'_j [\, \mathbf{q_i a_j b_j} \$ [\, g_j g'_j q_l \#_1]] \Rightarrow^{2r1,2r2}$$
$$a'_j d_j d'_j [\, \$ [\, q_i a_j \mathbf{b_j g_j} g'_j q_l \#_1]] \Rightarrow^{2r3} a'_j d_j d'_j [\, \mathbf{b_j g_j} \$ [\, q_i a_j g'_j q_l \#_1]] \Rightarrow^{1r2}$$
$$d'_j \mathbf{b_j d_j g_j a'_j} [\, \$ [\, q_i a_j g'_j q_l \#_1]] \Rightarrow^{1r8,1r5} d'_j [\, d_j b_j g_j a'_j \$ [\, q_i a_j g'_j q_l \#_1]]$$

Again there are two variants of computations, depending on the application of rule `1r2` or rule `2r6`, where applying rule `1r2` leads to an infinite computation (see case a)). Hence, we only consider the case of applying rule `2r6`:

$$d'_j [\, d_j b_j \mathbf{g_j a'_j} \$ [\, q_i a_j g'_j q_l \#_1]] \Rightarrow^{2r6} d'_j [\, d_j b_j \$ [\, q_i a_j g_j \mathbf{a'_j g'_j} q_l \#_1]] \Rightarrow^{2r8}$$
$$d'_j [\, \mathbf{a'_j g'_j} d_j b_j \$ [\, q_i a_j g_j q_l \#_1]] \Rightarrow^{1r11} a'_j \mathbf{g'_j d'_j} [\, d_j b_j \$ [\, q_i a_j g_j q_l \#_1]] \Rightarrow^{1r12}$$
$$a'_j [\, d_j b_j \$ \mathbf{g'_j d'_j} [\, q_i a_j g_j q_l \#_1]] \Rightarrow^{2r9} a'_j [\, d_j b_j \$ [\, q_i a_j g_j g'_j \mathbf{d'_j q_l} \#_1]] \Rightarrow^{2r10}$$
$$a'_j [\, \mathbf{d'_j q_l d_j} b_j \$ [\, q_i a_j g_j g'_j \#_1]] \Rightarrow^{1r13,1r14} a'_j d_j d'_j q_l [\, b_j \$ [\, q_i a_j g_j g'_j \#_1]]$$

Thus, q_i is replaced by q_l.

Case f):

$$a'_j d_j d'_j \mathbf{q_i a_j} [\, c_k b_j \$ [\, g_j g'_j q_l \#_1]] \Rightarrow^{1r1} a'_j d_j d'_j [\, c_k \mathbf{q_i a_j b_j} \$ [\, g_j g'_j q_l \#_1]] \Rightarrow^{2r1,2r2}$$
$$a'_j d_j d'_j [\, c_k \$ [\, q_i a_j \mathbf{b_j g_j} g'_j q_l \#_1]] \Rightarrow^{2r3} a'_j d_j d'_j [\, c_k \mathbf{b_j g_j} \$ [\, q_i a_j g'_j q_l \#_1]] \Rightarrow^{1r2}$$
$$d'_j \mathbf{b_j d_j g_j a'_j} [\, c_k \$ [\, q_i a_j g'_j q_l \#_1]] \Rightarrow^{1r8,1r5} d'_j [\, \mathbf{c_k d_j g_j a'_j} b_j \$ [\, q_i a_j g'_j q_l \#_1]] \Rightarrow^{1r9,2r6}$$
$$c_k d_j d'_j [\, b_j \$ [\, q_i a_j g_j \mathbf{a'_j g'_j} q_l \#_1]] \Rightarrow^{2r8} c_k d_j d'_j [\, \mathbf{a'_j g'_j} b_j \$ [\, q_i a_j g_j q_l \#_1]] \Rightarrow^{1r11}$$
$$a'_j c_k d_j \mathbf{g'_j d'_j} [\, b_j \$ [\, q_i a_j g_j q_l \#_1]] \Rightarrow^{1r12} a'_j c_k d_j [\, b_j \$ \mathbf{g'_j d'_j} [\, q_i a_j g_j q_l \#_1]] \Rightarrow^{2r9}$$
$$a'_j c_k d_j [\, b_j \$ [\, q_i a_j g_j g'_j \mathbf{d'_j q_l} \#_1]] \Rightarrow^{2r10} a'_j c_k d_j [\, \mathbf{d'_j} q_l b_j \$ [\, q_i a_j g_j g'_j \#_1]] \Rightarrow^{1r13}$$
$$a'_j c_k d_j d'_j [\, q_l b_j \$ [\, q_i a_j g_j g'_j \#_1]]$$

Further we continue our work only by applying the rules `1s1` and `1s2`, thus, the computation will never stop.

3. END.

$R_{1,f} = \{\texttt{1f1} : (q_f a_n, in)\}$
$R_{2,f} = \{\texttt{2f1} : (\#_2 g_n, in),\ \texttt{2f2} : (c_1 a_n, in),\ \texttt{2f3} : (f a_n, in),\ \texttt{2f4} : (f g_n, out),$
$\texttt{2f5} : (a_n, out)\}.$

All objects c_1 move to region 2, and object $\#_2$ moves to region 2, thus we stop without continuing the loop. □

4 Final Remarks

Both constructions from Theorem 1 and Theorem 2 can easily be modified to show that

$PsOP_2(sym_1, anti_1)_T = PsRE$ and
$PsOP_2(sym_2)_T = PsRE$,

i.e., the results proved in Theorem 1 and Theorem 2 can be extended from sets of natural numbers to sets of vectors of natural numbers.

An interesting open question remains: can we avoid the terminal alphabet? The answer partially is yes, i.e., the proof of Theorem 1 can be modified in such a way that only three additional symbols remain, but so far we were not able to completely avoid additional symbols that remain after halting a computation. Hence, although the results presented in this paper are optimal with respect to the number of membranes and the size of the rules used in the communicative P systems we considered, there still remain some challenging open problems for future research.

Acknowledgements

The first author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona. The first and the third authors acknowledge the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034 for providing a challenging and fruitful framework for cooperation.

References

1. A. Alhazov, M. Margenstern, V. Rogozhin, Y. Rogozhin, S. Verlan: Communicative P systems with minimal cooperation. In *Membrane Computing, 5th International Workshop, WMC5* (G. Mauri, Gh. Păun, M.J. Pérez Jiménez, G. Rozenberg, A. Salomaa, eds.), LNCS 3365 (2005), 161–177.

2. A. Alhazov, Y. Rogozhin, S. Verlan: Symport/antiport tissue P systems with minimal cooperation. In the present volume, 37–51.
3. F. Bernardini, M. Gheorghe: On the power of minimal symport/antiport. In *Pre–proceedings of the Workshop on Membrane Computing* (A. Alhazov, C. Martín-Vide, G. Păun, eds.), Tarragona, July 17-22, 2003, 72–83.
4. F. Bernardini, A. Păun: Universality of minimal symport/antiport: five membranes suffice. In *Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain, July 2003, Revised Papers* (C. Martín-Vide, Gh. Paun, G. Rozenberg, A. Salomaa, eds.), LNCS 2933 (2004), 43–45.
5. R. Freund, M. Oswald: GP systems with forbidding context. *Fundamenta Informaticae*, 49, 1-3 (2002), 81–102.
6. R. Freund, A. Păun: Membrane systems with symport/antiport: universality results. In *Membrane Computing. International Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597 (2003), 270–287.
7. P. Frisco: About P systems with symport/antiport. In *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, University of Seville, 2004, 224–236.
8. P. Frisco, H.J. Hoogeboom: P systems with symport/antiport simulating counter automata. *Acta Informatica*, 41, 2-3 (2004), 145–170.
9. P. Frisco, H.J. Hoogeboom: Simulating counter automata by P systems with symport/antiport. In *Membrane Computing. International Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597 (2003), 288–301.
10. L. Kari, C. Martín-Vide, A.Păun: On the universality of P systems with minimal symport/antiport rules. In *Aspects of Molecular Computing* (N. Jonoska, Gh. Păun, G. Rozenberg, eds.), LNCS 2950 (2004), 254–265.
11. M. Margenstern, V. Rogozhin, Y. Rogozhin, S. Verlan: About P systems with minimal symport/antiport rules and four membranes. In *Preliminary Proceedings of WMC 2004, Workshop on Membrane Computing, Milano, Italy, June, 14-16, 2004* (G. Mauri, Gh. Păun, C. Zandron, eds.), Universitá di Milano-Bicocca 2004, 283–294.
12. M.L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
13. Gh. Păun: Computing with membranes. *Journal of Computer and Systems Science*, 61 (2000), 108–143.
14. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20 (2002), 295–305.
15. Gh. Păun: *Membrane computing. An Introduction*. Springer-Verlag, Berlin, 2002.
16. G. Vaszil: On the size of P systems with minimal symport/antiport. In *Preliminary Proceedings of WMC 2004, Workshop on Membrane Computing, Milano, Italy, June, 14-16, 2004* (G. Mauri, Gh. Păun, C. Zandron, eds.), Universitá di Milano-Bicocca, 2004, 422–431.
17. S. Verlan: Optimal results on tissue P systems with minimal symport/antiport. Presented at *Molecular Computing meeting*, Lorentz Center, Leiden, The Netherlands, 22–26 November, 2004.