

# About P Systems with Minimal Symport/Antiport rules and Four Membranes

Maurice MARGENSTERN<sup>1</sup>, Vladimir ROGOZHIN<sup>2</sup>,  
Yurii ROGOZHIN<sup>3</sup>, Sergey VERLAN<sup>1</sup>

<sup>1</sup> LITA, Université de Metz, France

E-mail: {margens,verlan}@sciences.univ-metz.fr

<sup>2</sup> State University of Moldova

E-mail: rv@math.md

<sup>3</sup> Institute of Mathematics and Computer Science

Academy of Sciences of Moldova

E-mail: rogozhin@math.md

## Abstract

P systems with symport/antiport rules of a minimal size (only one object passes in any direction in a communication step) and with five membranes have been recently proved to be computationally universal [2]. We improved this result and reduced the number of membranes down to four.

## 1 Introduction

P systems were introduced by Gheorghe Păun in [10] as distributed parallel computing devices of a biochemical inspiration. The original definition is rather abstract and a lot of different variants of P systems were proposed; see [13] for a comprehensive bibliography. One of these variants, P systems with *symport/antiport*, was introduced in [11], and it uses one of the most important properties of membrane systems: the communication. These systems have two types of rules: symport rules, when several objects go together from one membrane to another, and antiport rules, when several objects from two membranes are exchanged. In spite of a simple definition, we can compute all Turing computable sets of numbers [11]. This result was improved with respect to the number of used membranes and/or the size of symport/antiport rules ([4], [6], [8], [12], [2]).

Rather unexpectedly, minimal symport/antiport membrane systems, that is, which uses only one object in symport or antiport rules, are also universal. The proof of this result can be found in [1] and the corresponding system has 9 membranes.

This result was improved first by reducing the number of membranes to six [7] and after that to five [2]. In [5] P. Frisco showed that four membranes are sufficient to generate all recursively enumerable sets of numbers. In this paper we give another proof of the last result which was obtained independently. We also remark that our

proof is simpler than the proof in [5] but it uses a non-elementary membrane as an output membrane.

Our proof is based on a simulation of counter automaton (or register machine [9]), see also [3], a technique which was also used in [1], [4], [7] and [2].

## 2 Basic Notions

A non-deterministic *counter automaton* is a 5-tuple  $M = (Q, q_1, q_n, R, P)$ , where

- $Q = \{q_i\}$ ,  $1 \leq i \leq n$ , is the set of possible states,
- $q_1 \in Q$  is the initial state,
- $q_n \in Q$  is the final state,
- $R = \{r_j\}$ ,  $1 \leq j \leq m$ , is the set of counters,
- $P$  is a finite set of instructions of the following form:
  1.  $(p, +r_j, q, r)$ , with  $p, q, r \in Q$ ,  $p \neq q_n$ ,  $r_j \in R$ . This instruction increments by 1 the counter  $r_j$  and changes the state of the system from  $p$  to  $q$  or  $r$ , non-deterministically. If we consider  $q = r$ , then we obtain a *deterministic* counter automaton.
  2.  $(p, -r_j, q, s)$ , with  $p, q, s \in Q$ ,  $p \neq q_n$ ,  $r_j \in R$ . If the value of counter  $r_j$  is bigger than zero, then this instruction decrements it by 1 and changes the state of the system from  $p$  to  $q$ . Otherwise, *i.e.*, if the value of  $r_j$  is zero, the state of the system is changed to  $s$ .
  3. *Stop*. This instruction stops the computation of the counter automaton and it can be assigned only to the final state  $q_n$ .

We note that for each  $p \in Q$ ,  $p \neq q_n$ , there is only one instruction of form  $(p, +r_j, q)$  or  $(p, -r_j, q, s)$ .

A transition of the counter automaton consists in updating/checking the value of a counter according to an instruction of one of types above and by changing the current state to another one. The computation starts in state  $q_1$  and with all counters equal to zero. A result of the computation of a counter automaton is the set of all values of the first counter when the computation is halted. We assume that the automaton empties all counters except the first counter before stopping.

It is known that non-deterministic counter automata generate all recursively enumerable sets of non-negative natural numbers starting from empty counters.

A  $P$  system with symport/antiport is a construct

$$\Pi = (V, \mu, w_0, w_1, \dots, w_k, E, R_1, \dots, R_k, i_0),$$

where:

1.  $V$  is a finite alphabet of symbols called objects,

2.  $\mu$  is a membrane structure consisting of  $m$  membranes that are labelled in a one-to-one manner with  $\{1, \dots, k\}$ ,
3.  $w_i \in V^*$ , for each  $1 \leq i \leq k$  is a *finite* multiset of objects associated with the region  $i$  (delimited by membrane  $i$ ),  $w_0$  is the finite multiset of objects from  $V$  associated with the environment,
4.  $E \subseteq V$  is the set of objects that appear in the environment in infinite numbers of copies,
5.  $R_i$ , for each  $1 \leq i \leq k$ , is a finite set of symport/antiport rules associated with the region  $i$  and which have the following form  $(x, in)$ ,  $(x, out)$ ,  $(x, out; y, in)$ , where  $x, y \in V^*$ ,
6.  $i_0$  is the label of a membrane of  $\mu$  that identifies the corresponding output region.

A  $P$  system with symport/antiport is defined as a computational device consisting of a set of  $k$  hierarchically nested membranes that identify  $k$  distinct regions (the membrane structure  $\mu$ ), where to each region  $i$  is 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 to objects specified by  $x$  to be moved from the outer membrane of  $i$  into membrane  $i$ . A rule  $(x, out) \in R_i$  permits to the multiset  $x$  to be moved from membrane  $i$  into the outer membrane. A rule  $(x, out; y, in)$  permits to multisets  $x$  and  $y$ , which are situated in membrane  $i$  and the outer membrane of  $i$  respectively, to be exchanged. It is clear that a rule can be applied if and only if the multisets involved by this rules are present in the corresponding regions. Rules associated with the skin membrane are only allowed to exchange with or send objects to the environment.

As usual, a computation in a  $P$  system with symport/antiport is obtained by applying the rules in a non-deterministic maximal 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 the multiset  $w_0$  containing objects in finite number of copies as well as objects from  $E$  that appear in an infinite number of copies. We note that we can introduce in the environment any finite number of copies of objects, therefore we can also consider the environment as in [2], where the environment before computation contains only an infinite number of copies of objects.

A computation is successful if starting from the initial configuration it reaches a configuration where no more rule can be applied. The result of a successful computation is a natural number that is obtained by counting the objects that are contained in membrane  $i_0$ . Given a  $P$  system  $\Pi$ , the set of natural numbers computed in this way by  $\Pi$  is denoted by  $N(\Pi)$ .

We denote by  $NOP_m(sym_r, anti_t)$  the family of sets of natural numbers that are generated by a  $P$  system with symport/antiport 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 is given by  $max\{|x|, |y|\}$ . We denote by  $NRE$  the family of recursively enumerable sets of natural numbers.

### 3 Main Result

**Theorem 3.1**  $NOP_4(sym_1, anti_1) = NRE$ .

*Proof.* We prove the result in the following way. We shall simulate a non-deterministic counter automaton  $M = (Q, q_1, q_n, R, P)$ ,  $|R| = m$ ,  $|Q| = n$ , which starts with empty counters. We also suppose that all addition instructions are numbered from 1 to  $n_1$  ( $(q_i, r_j+, q_k, q_r)$  and  $1 \leq i \leq n_1$ ) and that all subtraction instructions are numbered from  $n_1 + 1$  to  $n - 1$  ( $(q_i, r_j-, q_k, q_r)$  and  $n_1 + 1 \leq i \leq n - 1$ ).

We construct a P system  $\Pi$  with the following membrane structure:

$$[1[2[3[4]4]3]2]_1$$

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

1. Preparation of the system for the computation.
2. The simulation of instructions of the counter automaton.

We code the counter automaton as follows. At each moment (after stage one) the membrane 1 holds the current state of the automaton, represented by a symbol  $q_k$ , and the value of all counters, represented by the number of occurrences of symbols  $c_j$ . We simulate the instructions of the counter automaton and we use for this simulation the symbols  $c_j$ ,  $f_i$ ,  $d'_i$  and  $d''_i$ , which are present in membranes 2, 3 and 4. Therefore, the number of these symbols decrease constantly. During the first stage we bring an arbitrary number of these symbols in the corresponding membranes and we suppose that we have enough symbols in the corresponding membranes to perform the computation. We also use the following idea: in our system we have a symbol  $t$  which moves from membrane 2 to membrane 1 and back in an infinite loop. This loop may be stopped only if all stages completed correctly. Otherwise, the computation will never stop.

We split our proof in several parts which depend on the logical separation of the behaviour of the system. We will present rules and initial symbols for each part, but we remark that the system that we present is the union of all these parts.

#### Part I (Push and pop)

In this part we shall present rules and objects which permit to push symbols from the upper membranes into inner membranes, as well as to pop symbols from inner membranes.

Objects and rules used (membrane 0 is the environment):

Membrane	Object(s)	Rules	
0.	$O_1$		
1.	$O_2$		
2.		1.2.1 : $(I_2, in)$	1.2.2 : $(O_1, out)$
3.	$I_2, I_3$	1.3.1 : $(I_3, in)$	1.3.2 : $(O_2, out)$
4.	$I_4$	1.4.1 : $(I_4, in)$	

Now we shall describe how one can use these rules and objects in order to push and pop other objects. Suppose that we have the following rule in membrane 2:  $r : (X, in; I_2, out)$ . Suppose also that we have symbol  $X$  in membrane 1. Then this rule permits to bring  $X$  from the first membrane into the second. Indeed, rule  $r$  exchanges  $X$  and  $I_2$ , while the rule 1.2.1 brings  $I_2$  back. In a similar way, rules of form  $(X, in; I_k, out)$ ,  $2 \leq k \leq 4$ , permit to push  $X$  from the upper membrane  $k - 1$  into membrane  $k$ . We remark that initially  $I_2$  is situated in the third membrane, so the push mechanism for the second membrane is initially deactivated.

Similarly, rules of form  $(X, out; O_k, in)$  in membrane  $k + 1$ ,  $k \in \{1, 2\}$  permit to pop symbol  $X$  into membrane  $k$  providing that  $O_k$  is in membrane  $k$ . We remark that initially  $O_k$  is situated in membrane  $k - 1$ , so this pop mechanism is initially deactivated.

### Part II (Infinite computation)

In this part we present rules and objects which permit to organise an infinite computation that will stop only under some conditions.

Objects and rules:

Membrane	Object(s)	Rules	
0.			
1.			
2.	$t$	2.2.1 : $(t, out)$	2.2.2 : $(t, in)$
3.			
4.			

One can see that symbol  $t$  is moving in a cycle between membranes 1 and 2. This cycle may end only under some conditions, more exactly, when symbol  $q_n$  which corresponds to the final state of the automaton is present in membrane 1. We shall use in the future the following observation: at some moment only an infinite computation with  $t$  is possible. We shall say that the system is *stuck* in this case. Of course, we do not obtain any result when the system is stuck because of the infinite computation with  $t$  that is still present.

### Part III (Pumping symbols)

This part represents the first stage and below we present rules and objects which permit to pump necessary symbols from the environment and to distribute them in corresponding membranes.

Objects and rules (where  $X \in \{d'_i, d''_i, f_i, c_j\}$  and  $Y \in \{d'_i, d''_i, f_i\}$ ,  $1 \leq i \leq n - 1$ ,  $1 \leq j \leq m$ ):

Mem.	Obj.	Rules		
0.	$f_i^\infty, d_i^{\prime\infty},$ $d_i^{\prime\prime\infty}, c_j^\infty$			
1.	$b_1$	3.1.1 : $(b_1, out; X, in),$ 3.1.2 : $(b_1, in)$		
2.	$b_2$	3.2.1 : $(b_2, out; X, in),$ 3.2.2 : $(b_2, in)$	3.2.3 : $(b_1, in)$	
3.	$b_3$	3.3.1 : $(b_3, out; Y, in),$ 3.3.2 : $(b_3, in)$	3.3.3 : $(b_1, in; I_3, out),$ 3.3.4 : $(b_1, out; b_2, in)$	
4.	$S, S'$	3.4.1 : $(d_i^{\prime\prime}, in)$	3.4.3 : $(b_1, in; I_4, out),$ 3.4.4 : $(b_1, out; b_3, in)$	3.4.5 : $(b_2, in; S', out)$ 3.4.6 : $(S', in; S, out)$

These rules permit to bring from the environment to the third membrane objects  $d_i^{\prime}$  and  $f_i$ , as well as the objects  $d_i^{\prime\prime}$  to the fourth membrane and objects  $c_j$  to the second membrane. This is done in the following way. Rules of type 3.x.1 and 3.x.2,  $1 \leq x \leq 3$  permit to push these objects from an upper membrane into an inner membrane in a way similar to that of Part I. In this way membrane 2 receives an arbitrary number of  $c_j$ 's, membrane 3 receives an arbitrary number of  $f_i$ 's and  $d_i^{\prime}$ 's and membrane 4 receives an arbitrary number of  $d_i^{\prime\prime}$ 's. At some moment, this process of pumping may stop: this happens if rule 3.2.3 is used. After that, symbol  $b_1$  moves down until the fourth membrane and brings in the same time symbols  $b_2$  and  $b_3$  to membranes 3 and 4 respectively. In this way the pumping procedure stops and symbols  $b_1, b_2, b_3$  cannot be used any more. Symbol  $b_2$  plays also an important role: it triggers the beginning of the computation. More exactly, by appearing in the third membrane it permits to rule 3.4.5 to be applied which brings symbol  $S'$  to the third membrane which brings during the next step the symbol  $S$  to the third membrane by using rule 3.4.6. In the next part we shall show how this symbol is moved outside and how it triggers the beginning of the simulation of the instructions of counter automaton  $M$ .

We remark that when a symbol  $f_i$  is in the first membrane we may apply either the rule 3.1.1 which pushes it to the second membrane, or one of rules 5.1.2 or 5.1.3 which are presented later. The application of the last two rules may lead to an incorrect evolution and in part VIII we show that we obtain an infinite computation in this case.

Similarly, the symbols  $d_i^{\prime}$  and  $d_i^{\prime\prime}$  in the first membrane may be used in rules 6.1.2 and 6.1.3 and in part VIII we shall show that this does not alter the computation.

#### Part IV (Starting the simulation)

This part makes the transition between the first stage and the second stage. We present below rules and objects which permit to start the simulation of the instructions of  $M$ . We recall that at this moment object  $S$  is in the third membrane.

Objects and rules:

Membrane	Object(s)	Rules	
0.			
1.		4.1.1 : ( $S, out; O_1, in$ )	
2.	$q_1, b_s$	4.2.1 : ( $S, out; O_2, in$ )	4.2.2 : ( $q_1, out; O_1, in$ )
3.		4.3.1 : ( $S, out; b_s, in$ )	4.3.2 : ( $I_2, out; O_2, in$ )
4.			

So, symbol  $S$  will move until the environment and it will bring  $O_k$ ,  $k \in \{1, 2\}$ , into membrane  $k$  and  $I_2$  (with the help of the symbol  $O_2$ ) into the second membrane. Therefore,  $O_k$  may further participate in popping symbols from membrane  $k + 1$  into membrane  $k$  and  $I_2$  may further participate in pushing symbols from membrane 1 into membrane 2. A first example of application of the popping is symbol  $q_1$  which is moved to membrane 1. At this moment the system starts to simulate the instructions of counter automaton  $M$ . We can easily see that in membrane 1 we have the symbol  $q_1$  and no symbol  $c_j$ , what corresponds to the initial state of the automaton having all counters set to zero.

In next two parts we shall show how to simulate the instructions of the automaton.

### Part V (Addition)

Let  $(q_i, +c_j, q_k, q_r)$  be an instruction of the automaton. We associate to this instruction the following objects and rules:

Mem.	Obj.	Rules	
0.	$a_i^\infty$		
1.		5.1.1 : ( $q_i, out; a_i, in$ )	5.1.2 : ( $f_i, out; q_k, in$ ) 5.1.3 : ( $f_i, out; q_r, in$ )
2.		5.2.1 : ( $a_i, in; c_j, out$ )	5.2.2 : ( $f_i, out; O_1, in$ )
3.		5.3.1 : ( $a_i, in; f_i, out$ )	
4.			

The simulation of the instruction above is done as follows. Symbol  $q_i$  is sent to the environment and introduces symbol  $a_i$  into the first membrane. After that,  $a_i$  brings one object  $c_j$  (corresponding to the value of the counter) into the first membrane and object  $f_i$  into the second membrane. After that object  $f_i$  goes to the environment and brings the new state  $q_k$  or  $q_r$  into the first membrane. We remark that this procedure may be done only if we have at least one  $c_j$ 's in the second membrane and at least one  $f_i$ 's in the third membrane. Otherwise the system is stuck and does not bring any result. We assume that the number of  $c_j$  and  $f_i$  which was brought into corresponding membranes during the initialisation stage (part III) is large enough, hence we may always simulate the corresponding instruction of the automaton.

### Part VI (Subtraction)

Let  $(q_i, -c_j, q_k, q_r)$  be an instruction of the automaton. We associate to this instruction the following objects and rules:

Mem.	Obj.	Rules		
0.	$d_i^\infty$			
1.		6.1.1 : $(q_i, out; d_i, in)$	6.1.2 : $(d'_i, out; q_k, in)$	6.1.3 : $(d''_i, out; q_r, in)$
2.		6.2.1 : $(d_i, in; I_2, out)$	6.2.2 : $(d'_i, out; c_j, in)$	6.2.3 : $(d''_i, out; O_1, in)$
3.		6.3.1 : $(d_i, in; d'_i, out)$	6.3.2 : $(d'_i, in; d''_i, out)$	
4.		6.4.1 : $(d_i, in; d''_i, out)$		

The simulation of the instruction above is done as follows. Symbol  $q_i$  is sent to the environment and introduces symbol  $d_i$  into the first membrane. Then this symbols moves down and in brings  $d'_i$  to the second membrane and  $d''_i$  to the third membrane. If the number of objects  $c_j$  is bigger than zero, then symbol  $d'_i$  goes to the first membrane and it sends at the same time one  $c_j$  into the third membrane. After that  $d'_i$ , goes to the environment and brings state  $q_k$ . We remark that  $d''_i$  returns to the fourth membrane because of rule 3.4.1.

If there are no objects  $c_j$  in the first membrane, then symbol  $d'_i$  waits for one step in the second membrane and after that it may be exchanged with symbol  $d''_i$  which was brought into the third membrane during the previous step. After that, symbol  $d''_i$  moves up until the environment from which it brings state  $q_r$ . We remark that if rule 6.3.2 is not applied, *i.e.*, the exchange of  $d'_i$  and  $d''_i$  does not take place, then the system is stuck after several steps because of the application of rule 3.4.1.

### Part VII (Stop)

If the halting state  $q_n$  of the automaton is present in the first membrane, then the computation is stopped after several steps. We present below objects and rules which are used to do this:

Membrane	Object(s)	Rules	
0.			
1.		7.1.1 : $(STOP, in; t, out)$	7.1.2 : $(STOP, in; O_1, out)$ 7.1.3 : $(STOP, out)$
2.		7.2.1 : $(STOP, out; q_n, in)$	
3.	$STOP$	7.3.1 : $(O_2, in; STOP, out)$	
4.			

If the object  $q_n$  corresponding to the halting state of the automaton is present in membrane 1, then it goes to membrane 2 and brings from there symbol  $STOP$ . This symbol permits to take symbol  $t$  to the environment, therefore it stops the infinite loop presented in part II. The same symbol permits to send  $O_1$  to the environment. After that the system stops and membrane 1 holds the result given by the number of occurrences of  $c_1$ . We remark that initially  $STOP$  is in the third membrane and it is brought into the second one by the symbol  $O_2$  during part IV.

Finally, we remark that the environment contains initially one occurrence of symbols  $q_i$ ,  $2 \leq i \leq n$ :

Membrane	Object(s)
0.	$q_i$
1.	
2.	
3.	
4.	

We recall that  $q_1$  is situated in membrane 2 (see part IV).

### Part VIII (Collision handling)

In this part we discuss a possible collision between rules of two stages: during the pumping, stage one, part III, when a symbol  $f_i$ ,  $d'_i$  or  $d''_i$  is present in membrane 1 we may apply either the rule 3.2.1 which leads to a correct evolution, or one of rules 5.1.2, 5.1.3, 6.1.2 or 6.1.3. We show below rules which are used to turn this application into a non-halting computation. We assume  $1 \leq i \leq n_1$ ,  $n_1 + 1 \leq l \leq n - 1$ .

Membrane	Object(s)	Rules	
0.			
1.		8.1.1 : $(a_i, out)$	8.1.2 : $(d_l, out)$ 8.1.3 : $(q_n, out)$
2.			
3.		8.3.1 : $(a_i, in)$	
4.		8.4.1 : $(a_i, in; S, out)$	8.4.2 : $(a_i, out; b_s, in)$ 8.4.3 : $(b_s, out; a_i, in)$

Suppose that we are during pumping (part III) and we have a symbol  $f_i$ ,  $d'_i$  or  $d''_i$  in the first membrane. Now suppose that instead of the correct evolution given by rule 3.2.1 one of rules 5.1.2, 5.1.3, 6.1.2 or 6.1.3 is used. In this case a symbol  $q_k$  is brought into the first membrane. Suppose  $k \in \{1, \dots, n_1\}$ , *i.e.*,  $q_k$  is an addition instruction:  $(q_k, c_j+, q_r, q_s)$ . In this case symbol  $a_k$  is exchanged with  $q_k$  during the next step. After that  $a_k$  goes to the second membrane, bringing at the same time  $c_j$  into the first one, and finally arrives in the third membrane. Since we are during the pumping part, symbol  $S$  is present in the fourth membrane. Then, the rule 8.4.1 is applied and  $a_k$  goes to the fourth membrane. During the next step, symbol  $S$  brings symbol  $b_s$  into the third membrane (rule 4.3.1). This symbol exchanges with  $a_k$  infinitely by rules 8.4.2 and 8.4.3. In this case the computation never stops. We remark that during a correct simulation symbol  $S$  leaves the fourth membrane before any symbol  $a_i$  arrives into the third, hence  $a_i$  will never go into the fourth membrane in this case. Finally we shall consider the case when the erroneous computation described above starts simultaneously with the end of the pumping, *i.e.*, when we apply the rule 3.2.3. We shall show that in this case symbol  $S$  is not exchanged with  $b_2$  before  $a_i$  arrives into the third membrane. Indeed, we start with a configuration having  $b_1$  and  $q_k$  into the first membrane. It is easy to see that  $b_1$  brings  $S$  into the third membrane in 5 steps (with the help of  $b_2$  and  $S'$ ), while  $q_k$  brings  $S$  into the third membrane in 4 steps (with the help of  $a_k$ ). Therefore, the exchange with  $a_k$  will happen before and an infinite computation will start.

We also remark that if there are not enough symbols  $c_j$  or  $f_k$  in corresponding membranes, rules 8.1.1 and 8.3.1 permit to handle this case by sending  $a_k$  to the environment in the first case, or by taking it into the third where it starts an infinite computation in the second case.

Now let us suppose that symbol  $q_k$  which is brought into the first membrane corresponds to a subtraction operation, *i.e.*  $n_1 + 1 \leq k \leq n - 1$ . In this case, the symbol  $d_k$  is brought into the first membrane. During the next step, this symbol goes to the environment by the rule 8.1.2 because symbol  $I_2$  is not present in the second membrane, hence rule 6.1.2 cannot be applied. The apparition of  $I_2$  is controlled by the symbol  $S$  which brings it (with the help of  $O_2$ ) to the second membrane. Therefore,  $I_2$  appears in the second membrane only after the pumping.

Finally, let us suppose that symbol  $q_n$  is brought into the first membrane. In this case it is sent to the environment by rule 8.1.3 because symbol  $STOP$  is not present in the second membrane, hence rule 7.2.1 cannot be applied. The apparition of  $STOP$  is controlled by symbol  $S$  which brings it (with the help of  $O_2$ ) to the second membrane. Therefore,  $STOP$  appears in the second membrane during part IV.

We remark that if we use one of rules 8.1.1, 8.1.2, 8.1.3 or 8.3.1 during a correct computation, then the system is stuck in this case. ■

We note that we used the fact that the environment may initially contain symbols with finite multiplicity. We remark that we do not need this assumption. It is easy to see that only states  $q_i$ ,  $2 \leq i \leq n$ , are initially present in one copy in the environment. We may replace them by the following objects and rules:

Membrane	Object(s)	Rules	
0.	$Z^\infty$		
1.	$q_i$	$q.1.1 : (Z, in; q_i, out)$	$q.1.2 : (Z, out)$
2.			
3.			
4.			

Indeed, during the first step symbols  $q_i$  are sent to the environment, and we continue as before. The new rules  $q.1.1$  may be used during the simulation of the counter automaton, but if they are used, then symbol  $q_i$  that codes the current state of the automaton is sent to the environment and the system is stuck.

## 4 Conclusions

In this paper we proved that P systems with symport/antiport rules of minimal size (only one object passes in any direction in a communication step) with four membranes are universal if a non-elementary membrane is permitted as an output membrane. However, we suppose that it is possible to modify the proof in such way that the result will be present in the elementary membrane 4. The question which is the size of families of numbers computed by P systems with minimal symport/antiport rules with 1, 2 and 3 membranes is still open. It is interesting also to

consider some control features on the use of rules (promoters, inhibitors and so on) in order to reduce the number of membranes sufficient to get universality.

## Program check

The system used in the proof of our result was checked for errors by the second author using a modification of a program that simulates P systems, originally developed by A. Alhazov.

**Acknowledgements** The authors acknowledge the project IST-2001-32008 "Mol-CoNet" and the second and the third authors acknowledge also 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] F. Bernardini, M. Gheorghe: On the Power of Minimal Symport/Antiport. In A. Alhazov, C. Martin-Vide, Gh. Păun (eds.): Workshop on Membrane Computing, WMC-2003, Tarragona, July 17–22, 2003, Technical Report N. 28/03, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona (2003) 72–83.
- [2] F. Bernardini, A. Păun: Universality of Minimal Symport/Antiport: Five Membranes Suffice. C. Martin-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (eds.): WMC 2003, LNCS **2933** (2004) 43–45.
- [3] R. Freund, M. Oswald: GP systems with forbidding context. *Fundamenta Informaticae*, **49**, 1-3 (2002), 81–102.
- [4] R. Freund, A. Păun: Membrane Systems with Symport/Antiport: Universality Results. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (eds.): Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, August 19–23, 2002. Revised Papers. LNCS **2597** (2003) 270–287
- [5] P. Frisco: About P Systems with Symport/Antiport. In Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez and F. Sancho-Caparrini (eds): Second Brainstorming Week on Membrane Computing. Technical report of University of Seville, *TR 01/2004* (2004) 224–236.
- [6] P. Frisco, J.H. Hoogeboom: Simulating Counter Automata by P Systems with Symport/Antypot. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (eds.): Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, August 19–23, 2002. Revised Papers. LNCS **2597** (2003) 288–301.
- [7] L. Kari, C. Martin-Vide, A. Păun: On the universality of P systems with Minimal Symport/Antiport Rules. LNCS **2950** (2004) 254–265.

- [8] C. Martin-Vide, A. Păun, Gh. Păun: On the Power of P systems with Symport and Antiport rules. *Journal of Universal Computer Science* **8** (2002) 295–305.
- [9] M.L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey (1967).
- [10] Gh. Păun: Computing with Membranes. *Journal of Computer and Systems Science* **61** (2000) 108–143.
- [11] A. Păun, Gh. Păun: The Power of Communication: P systems with Symport/Antiport. *New Generation Computing* **20** (2002) 295–305.
- [12] Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag (2002).
- [13] The P systems web page. <http://psystems.disco.unimib.it/>.