On the Universality of P Systems with Minimal Symport/Antiport Rules*

Lila Kari[1], Carlos Martín-Vide[2], and Andrei Păun[3]

[1] Department of Computer Science, University of Western Ontario
London, Ontario, Canada N6A 5B7
lila@csd.uwo.ca
[2] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
cmv@astor.urv.es
[3] Department of Computer Science
College of Engineering and Science
Louisiana Tech University, Ruston
P.O. Box 10348, Louisiana, LA-71272 USA
apaun@coes.latech.edu

Abstract. P systems with symport/antiport rules of a minimal size (only one object passes in any direction in a communication step) were recently proven to be computationally universal. The proof from [2] uses systems with nine membranes. In this paper we improve this results, by showing that six membranes suffice. The optimality of this result remains open (we believe that the number of membranes can be reduced by one).

1 Introduction

The present paper deals with a class of P systems which has recently received a considerable interest: the purely communicative ones, based on the biological phenomena of symport/antiport.

P systems are distributed parallel computing models which abstract from the structure and the functioning of the living cells. In short, we have a *membrane structure*, consisting of several membranes embedded in a main membrane (called the *skin*) and delimiting *regions* (Figure 1 illustrates these notions) where multisets of certain *objects* are placed. In the basic variant, the objects evolve according to given *evolution rules*, which are applied non-deterministically (the rules to be used and the objects to evolve are randomly chosen) in a maximally parallel manner (in each step, all objects which can evolve must evolve). The objects can also be communicated from one region to another one. In this way, we get *transitions* from a *configuration* of the system to the next one. A sequence of transitions constitutes a *computation*; with each *halting computation* we associate a *result*, the number of objects in an initially specified *output membrane*.

* Research supported by Natural Sciences and Engineering Research Council of Canada grants and the Canada Research Chair Program to L.K. and A.P.

N. Jonoska et al. (Eds.): Molecular Computing (Head Festschrift), LNCS 2950, pp. 254–265, 2004.

Since these computing devices were introduced ([8]) several different classes were considered. Many of them were proved to be computationally complete, able to compute all Turing computable sets of natural numbers. When membrane division, membrane creation (or string-object replication) is allowed, NP-complete problems are shown to be solved in polynomial time. Comprehensive details can be found in the monograph [9], while information about the state of the art of the domain can be found at the web address `http://psystems.disco.unimib.it`.

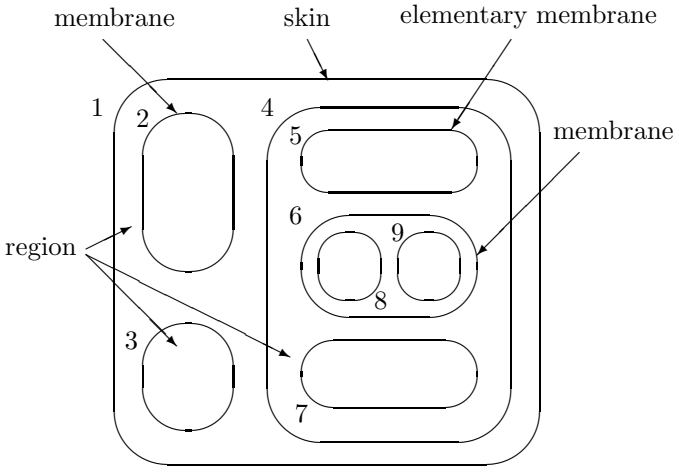

Figure 1: A membrane structure

A purely communicative variant of P systems was proposed in [7], modeling a real life phenomenon, that of membrane transport in pairs of chemicals – see [1]. When two chemicals pass through a membrane only together, in the same direction, we say that we have a process of *symport*. When the two chemicals pass only with the help of each other, but in opposite directions, the process is called *antiport*. For uniformity, when a single chemical passes through a membrane, one says that we have an *uniport*.

Technically, the rules modeling these biological phenomena and used in P systems are of the forms $(x, in), (x, out)$ (for symport), and $(x, out; y, in)$ (for antiport), where x, y are strings of symbols representing multisets of chemicals. Thus, the only used rules govern the passage of objects through membranes, the objects only change their places in the compartments of the membrane structure, they never transform/evolve.

Somewhat surprisingly, computing by communication only, in this "osmotic" manner, turned out to be computationally universal: by using only symport and antiport rules we can compute all Turing computable sets of numbers, [7]. The results from [7] were improved in several places – see, e.g., [3], [4], [6], [9] – in what concerns the number of membranes used and/or the size of symport/antiport rules.

Recently, a rather unexpected result was reported in [2]: in order to get the universality, minimal symport and antiport rules, that is of the forms $(a, in), (a, out), (a, out; b, in)$, where a, b are objects, are sufficient. The price was to use nine membranes, much more than in the results from [3] and [4], for example. The problem whether or not the number of membranes can be decreased was formulated as an open problem in [2]. We contribute here to this question, by improving the result from [2]: six membranes suffice. The proof uses the same techniques as the proofs from [2], [4]: simulating a counter automaton by a P system with minimal symport/antiport rules.

It is highly probable that our result is not optimal, but we conjecture that it cannot be significantly improved; we believe that at most one membrane can be saved.

2 Counter Automata

In this section we briefly recall the concept of counter automata, useful in the proof of our main theorem. We follow here the style of [2] and [4]. Informally speaking, a *counter automaton is a finite state machine* that has a finite number of *counters* able to store values represented by natural numbers; the machine runs a program consisting of instructions which can increase or decrease by one the contents of registers, changing at the same time the state of the automaton; starting with each counter empty, the machine performs a computation; if it reaches a terminal state, then the number stored in a specified counter is said to be generated during this computation. It is known (see, e.g., [5]) that counter automata (of various types) are computationally universal, they can generate exactly all Turing computable sets of natural numbers.

More formally, a *counter automaton* is a construct $M = (Q, F, p_0, C, c_{out}, S)$, where:

- Q is the set of the possible states,
- $F \subseteq Q$ is the set of the final states,
- $p_0 \in Q$ is the start state,
- C is the set of the counters,
- $c_{out} \in C$ is the output counter,
- S is a finite set of instructions of the following forms:
  - $(p \to q, +c)$, with $p, q \in Q,\ c \in C$: add 1 to the value of the counter c and move from state p into state q;
  - $(p \to q, -c)$, with $p, q \in Q,\ c \in C$: if the current value of the counter c is not zero, then subtract 1 from the value of the counter c and move from state p into state q; otherwise the computation is blocked in state p;
  - $(p \to q, c = 0)$, with $p, q \in Q,\ c \in C$: if the current value of the counter c is zero, then move from state p into state q; otherwise the computation is blocked in state p.

A transition step in such a counter automaton consists in updating/checking the value of a counter according to an instruction of one of the types presented above and moving from a state to another one. Starting with the number zero stored in each counter, we say that the counter automaton computes the value n if and only if, starting from the initial state, the system reaches a final state after a finite sequence of transitions, with n being the value of the output counter c_{out} at that moment.

Without loss of generality, we may assume that in the end of the computation the automaton makes zero all the counters but the output counter; also, we may assume that there are no transitions possible that start from a final state (this is to avoid the automaton getting stuck in a final state).

As we have mentioned above, such counter automata are computationally equivalent to Turing machines, and we will make below an essential use of this result.

3 P Systems with Symport/Antiport Rules

The language theory notions we use here are standard, and can be found in any of the many monographs available, for instance, in [11].

A membrane structure is pictorially represented by a Venn diagram (like the one in Figure 1), and it will be represented here by a string of matching parentheses. For instance, the membrane structure from Figure 1 can be represented by $[_1[_2\,]_2[_3\,]_3[_4[_5\,]_5[_6[_8\,]_8[_9\,]_9]_6[_7\,]_7]_4]_1$.

A multiset over a set X is a mapping $M : X \longrightarrow \mathbf{N}$. Here we always use multisets over finite sets X (that is, X will be an alphabet). A multiset with a finite support can be represented by a string over X; the number of occurrences of a symbol $a \in X$ in a string $x \in X^*$ represents the multiplicity of a in the multiset represented by x. Clearly, all permutations of a string represent the same multiset, and the empty multiset is represented by the empty string, λ.

We start from the biological observation that there are many cases where two chemicals pass at the same time through a membrane, with the help of each other, either in the same direction, or in opposite directions; in the first case we say that we have a *symport*, in the second case we have an *antiport* (we refer to [1] for details).

Mathematically, we can capture the idea of symport by considering rules of the form (ab, in) and (ab, out) associated with a membrane, and stating that the objects a, b can enter, respectively, exit the membrane together. For antiport we consider rules of the form $(a, out; b, in)$, stating that a exits and at the same time b enters the membrane. Generalizing such kinds of rules, we can consider rules of the unrestricted forms (x, in), (x, out) (generalized symport) and $(x, out; y, in)$ (generalized antiport), where x, y are non-empty strings representing multisets of objects, without any restriction on the length of these strings.

Based on rules of this types, in [7] one introduces *P systems with symport/antiport* as constructs

$$\Pi = (V, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_o),$$

where:

- V is an alphabet (its elements are called *objects*);
- μ is a membrane structure consisting of m membranes, with the membranes (and hence the regions) injectively labeled with $1, 2, \ldots, m$; m is called the *degree* of Π;
- $w_i, 1 \leq i \leq m$, are strings over V representing multisets of objects associated with the regions $1, 2, \ldots, m$ of μ, present in the system at the beginning of a computation;
- $E \subseteq V$ is the set of objects which are supposed to continuously appear in the environment in arbitrarily many copies;
- $R_1, \ldots, R_m$ are finite sets of symport and antiport rules over the alphabet V associated with the membranes $1, 2, \ldots, m$ of μ;
- i_o is the label of an elementary membrane of μ (the *output membrane*).

For a symport rule (x, in) or (x, out), we say that $|x|$ is the *weight* of the rule. The *weight* of an antiport rule $(x, out; y, in)$ is $\max\{|x|, |y|\}$.

The rules from a set R_i are used with respect to membrane i as explained above. In the case of (x, in), the multiset of objects x enters the region defined by the membrane, from the surrounding region, which is the environment when the rule is associated with the skin membrane. In the case of (x, out), the objects specified by x are sent out of membrane i, into the surrounding region; in the case of the skin membrane, this is the environment. The use of a rule $(x, out; y, in)$ means expelling the objects specified by x from membrane i at the same time with bringing the objects specified by y into membrane i. The objects from E (in the environment) are supposed to appear in arbitrarily many copies; since we only move objects from a membrane to another membrane and do not create new objects in the system, we need a supply of objects in order to compute with arbitrarily large multisets. The rules are used in the non-deterministic maximally parallel manner specific to P systems with symbol objects: in each step, a maximal number of rules is used (all objects which can change the region should do it).

In this way, we obtain transitions between the configurations of the system. A configuration is described by the m-tuple of multisets of objects present in the m regions of the system, as well as the multiset of objects from $V - E$ which were sent out of the system during the computation; it is important to keep track of such objects because they appear only in a finite number of copies in the initial configuration and can enter the system again. On the other hand, it is not necessary to take care of the objects from E which leave the system because they appear in arbitrarily many copies in the environment as defined before (the environment is supposed to be inexhaustible, irrespective how many copies of an object from E are introduced into the system, still arbitrarily many remain in the environment). The initial configuration is $(w_1, \ldots, w_m, \lambda)$. A sequence of transitions is called a computation.

With any halting computation, we may associate an output represented by the number of objects from V present in membrane i_o in the halting configuration. The set of all such numbers computed by Π is denoted by $N(\Pi)$. The

family of all sets $N(\Pi)$ computed by systems Π of degree at most $m \geq 1$, using symport rules of weight at most p and antiport rules of weight at most q, is denoted by $NOP_m(sym_p, anti_q)$ (we use here the notations from [9]).

Details about P systems with symport/antiport rules can be found in [9]; a complete formalization of the syntax and the semantics of these systems is provided in [10].

We recall from [3], [4] the best known results dealing with the power of P systems with symport/antiport.

Theorem 1. *$NRE = NOP_m(sym_r, anti_t)$, for $(m, r, t) \in \{(1, 1, 2), (3, 2, 0), (2, 3, 0)\}$.*

The optimality of these results is not known. In particular, it is an *open problem* whether or not also the families $NOP_m(sym_r, anti_t)$ with $(m, r, t) \in \{(2, 2, 0), (2, 2, 1)\}$ are equal to NRE.

Note that we do not have here a universality result for systems of type $(m, 1, 1)$. Recently, such a surprising result was proved in [2]:

Theorem 2. *$NRE = NOP_9(sym_1, anti_1)$.*

Thus, at the price of using nine membranes, uniport rules together with antiport rules as common in biology (one chemical exits in exchange with other chemical) suffice for obtaining the Turing computational level. The question whether or not the number of membranes can be decreased was formulated as an open problem in [2].

4 Universality with Six Membranes

We (partially) solve the problem from [2], by improving the result from Theorem 2: the number of membranes can be decreased to six – but we do not know whether this is an optimal bound or not.

Theorem 3. *$NRE = NOP_6(sym_1, anti_1)$.*

Proof. Let us consider a counter automaton $M = (Q, F, p_0, C, c_{out}, S)$ as specified in Section 2. We construct the symport/antiport P system

$$\Pi = (V, \mu, w_1, w_2, w_3, w_4, w_5, w_6, E, R_1, R_2, R_3, R_4, R_5, R_6, i_o),$$

where:

$$\begin{aligned} V = {} & Q \cup \{c_q \mid c \in C,\ q \in Q, \text{ and } (p \to q, +c) \in S\} \\ & \cup \{c'_q,\ d_{c,q} \mid c \in C,\ q \in Q, \text{ and } (p \to q, -c) \in S\} \\ & \cup \{c''_q,\ d'_{c,q} \mid c \in C,\ q \in Q, \text{ and } (p \to q, c = 0) \in S\} \\ & \cup \{a_1, a_2, a_3, a_4, b_1, b_2, i_1, i_2, i_3, i_4, i_5, h, h', h'', n_1, n_2, n_3, n_4, \#_1, \#_3\}, \\ \mu = {} & [_1[_2\ [_3[_4\ ]_4]_3\ [_5[_6\ ]_6]_5\ ]_2]_1, \end{aligned}$$

$$
\begin{aligned}
w_1 &= b_1 b_2 \#_3, \\
w_2 &= a_1 i_1 i_2 i_4 i_5 n_1 n_2 n_3 n_4 h'' \#_1, \\
w_3 &= a_2 a_3 i_3, \\
w_4 &= a_4 \#_3, \\
w_5 &= h', \\
w_6 &= \lambda, \\
E &= V - \{\#_1, b_1\}, \\
i_o &= 6, \\
R_i &= R'_i \cup R''_i \cup R'''_i, \text{ where } 1 \leq i \leq 6.
\end{aligned}
$$

Each computation in M will be simulated by Π in three main phases; the first phase will use rules from R'_i, $1 \leq i \leq 6$, R''_i contains the rules for the second phase, and R'''_i are the rules used for the third phase. These phases perform the following operations: (1) preparing the system for the simulation, (2) the actual simulation of the counter automaton, and (3) terminating the computation and moving the relevant objects into the output membrane.

We give now the rules from the sets R'_i, R''_i, R'''_i for each membrane together with explanations about their use in the simulation of the counter automaton.

Phase 1 performs the following operations: we bring in membrane 2 an arbitrary number of objects $q \in Q$ that represent the states of the automaton, then we also bring in membrane 4 an arbitrary number of objects $d_{c,q}$ and $d'_{c,q}$ that will be used in the simulation phase for simulating the rules $(p \to q, -c)$ and $(p \to q, c = 0)$, respectively. The rules used in this phase are as follows:

$$
\begin{aligned}
R'_1 &= \{(b_1, out; X, in) \mid X \in Q \cup \{d_{c,p}, d'_{c,p} \mid c \in C, p \in Q\}\} \cup \{(b_1, in)\}, \\
R'_2 &= \{(b_2, out; X, in) \mid X \in Q \cup \{d_{c,p}, d'_{c,p} \mid c \in C, p \in Q\}\} \cup \{(b_2, in)\} \\
&\quad \cup \{(a_1, out; b_1, in), (b_2, out; \#_3, in), (a_2, out; a_1, in), (a_2, out; \#_3, in)\}, \\
R'_3 &= \{(a_3, out; d, in) \mid d \in \{d_{c,p}, d'_{c,p} \mid c \in C, p \in Q\}\} \cup \{(a_3, in)\} \\
&\quad \cup \{(a_2, out; b_2, in), (a_4, out, b_1, in), (\#_3, in), (\#_3, out)\}, \\
R'_4 &= \{(a_4, out; d, in) \mid d \in \{d_{c,p}, d'_{c,p} \mid c \in C, p \in Q\}\} \cup \{(a_4, in)\} \\
&\quad \cup \{(a_4, out; b_2, in), (b_2, out; a_3, in)\}, \\
R'_5 &= \{(h', out; h'', in), (h'', out, h', in)\}, \\
R'_6 &= \emptyset.
\end{aligned}
$$

The special symbol b_1 brings from the environment the objects q, $d_{c,q}$, $d'_{c,q}$ by means of the rules $(b_1, out; X, in)$, and at the same time the symbol b_2 enters membrane 2 using the rule (b_2, in). At the next step b_1 comes back in the system, while b_2 moves the object that was introduced in membrane 1 in the previous step, q, $d_{c,q}$, or $d'_{c,q}$ into membrane 2 by means of the rules $(b_2, out; X, in)$. We can iterate these steps since we reach a configuration similar with the original configuration.

If the objects moved from the environment into membrane 2 are versions of d, then those objects are immediately moved into membrane 4 by the rules

$(a_3, out; d, in) \in R'_3$ and $(a_4, out; d, in) \in R'_4$. One can notice that in the simulation phase these special symbols that we bring in the system in this initial phase are used to simulate some specific rules from the counter automaton. A difficulty appears here because there are rules that will allow such a symbol to exit membrane 1 bringing in another such symbol (this leads to a partial simulation of rules from the counter automaton). To solve this problem we make sure that the symbols that we bring in the system do end up into one of membranes 2 or 4: if an object q, $d_{c,q}$, or $d'_{c,q}$ exits membrane 1 (using a rule from R''_1) immediately after it is brought in, then b_2 which is in membrane 2 now has to use the rule $(a_2, out; \#_3, in) \in R'_2$ and then the computation will never halt since $\#_3$ is present in membrane 2 and will move forever between membranes 2 and 3 by means of $(\#_3, in)$, $(\#_3, out)$ from R'_3.

After bringing in membrane 2 an arbitrary number of symbols q and in membrane 4 an arbitrary number of symbols $d_{c,q}, d'_{c,q}$ we pass to the second phase of the computation in Π, the actual simulation of rules from the counter automaton. Before that we have to stop the "influx" of special symbols from the environment: instead of going into environment, b_1 is interchanged with a_1 from membrane 2 by means of the rule $(a_1, out; b_1, in)$; at the same step b_2 enters the same membrane 2 by (b_2, in). Next b_2 is interchanged with a_2 by using the rule $(a_2, out; b_2, in) \in R'_3$, then in membrane 4 the same b_2 is interchanged with a_4 by means of $(a_4, out; b_2, in) \in R'_4$; simultaneously, for membrane 2 we apply $(a_2, out; a_1, in)$. At the next step we use the rules (a_4, out, b_1, in) and $(b_2, out; a_3, in)$ from R'_4.

There are two delicate points in this process. First, if b_2 instead of bringing in membrane 2 the objects from environment starts the finishing process by $(a_2, out; b_2, in) \in R'_3$, then at the next step the only rule possible is $(a_2, out; \#_3, in) \in R'_2$, since a_1 is still in membrane 2, and then the computation will never render a result. The second problem can be noticed by looking at the rules (a_4, in) and $(a_4, out; b_1, in)$ associated with membranes 4 and 3, respectively: if instead of applying the second rule in the finishing phase of this step, we apply the rule of membrane 4, then the computation stops in membranes 1 through 4, but for membrane 5 we will apply the rules from R'_5 continuously.

The second phase starts by introducing the start state of M in membrane 1, then we simulate all the rules from the counter automaton; to do this we use the following rules:

$$
\begin{aligned}
R''_1 = {} & \{(a_4, out; p_0, in), (\#_1, out), (\#_1, in)\} \cup \{(p, out; c_q, in), (p, out; c'_q, in), \\
& (d_{c,q}, out; q, in), (p, out; c''_q, in), (d'_{c,q}, out; q, in) \mid p, q \in Q, c \in C\}, \\
R''_2 = {} & \{(q, out; c_q, in), (\#_1, out; c_q, in), (n_1, out; c'_q, in), (d_{c,q}, out; n_4, in), \\
& (i_1, out; c''_q, in), (d_{c,q}, out; \#_3, in), (d'_{c,q}, out; i_5, in), (d'_{c,q}, out; \#_3, in) \mid \\
& q \in Q, c \in C\} \\
& \cup \{(a_4, out), (n_2, out; n_1, in), (n_3, out; n_2, in), (n_4, out; n_3, in), \\
& (\#_1, out; n_4, in), (i_2, out; i_1, in), (i_3, out; i_2, in), (i_4, out; i_3, in), \\
& (i_5, out; i_4, in)\},
\end{aligned}
$$

$$R_3'' = \{(c_q', in), (d_{c,q}, out; c_\alpha, in), (i_3, out; c_q'', in), (d_{c,q}', out; c_\alpha, in), (d_{c,q}', out; i_3, in) \mid q, \alpha \in Q, c \in C\},$$
$$R_4'' = \{(d_{c,q}, out; c_q', in), (\#_3, out; c_q', in), (d_{c,q}', out; c_q'', in), (\#_3, out; c_q'', in) \mid q \in Q, c \in C\},$$
$$R_5'' = \emptyset,$$
$$R_6'' = \emptyset.$$

We explain now the usage of these rules: we bring in the system the start state p_0 by using the rules $(a_4, out) \in R_2''$ and then $(a_4, out; p_0, in) \in R_1''$; a_4 should be in membrane 2 at the end of the step 1 if everything went well in the first phase.

We are now ready to simulate the transitions of the counter automaton.

The simulation of an instruction $(p \to q, +c)$ is done as follows. First p is exchanged with c_q by $(p, out; c_q, in) \in R_1''$, and then at the next step q is pushed in membrane 1 while c_q enters membrane 2 by means of the rule $(q, out; c_q, in) \in R_2''$. If there are no more copies of q in membrane 2, then we have to use the rule $(\#_1, out; c_q, in) \in R_2''$, which kills the computation. It is clear that the simulation is correct and can be iterated since we have again a state in membrane 1.

The simulation of an instruction $(p \to q, -c)$ is performed in the following manner. The state p is exchanged in this case with c_q' by the rule $(p, out; c_q', in) \in R_1''$. The object c_q' is responsible of decreasing the counter c and then moving the automaton into state q. To do this c_q' will go through the membrane structure up to membrane 4 by using the rules $(n_1, out; c_q', in) \in R_2''$, $(c_q', in) \in R_3''$, and $(d_{a,q}, out; c_q', in) \in R_4''$. When entering membrane 2, it starts a "timer" in membrane 1 and when entering membrane 4 it brings out the symbol $d_{c,q}$ which will perform the actual decrementing of the counter c.

The next step of the computation involves membrane 3, by means of the rule $(d_{c,q}, out; c_\alpha, in) \in R_3''$, which is effectively decreasing the content of counter c. If no more copies of c_α are present in membrane 2, then $d_{c,q}$ will sit in membrane 3 until the object n, the timer, reaches the subscript 4 and then $\#_3$ is brought in killing the computation, by means of the following rules from R_2'': $(n_2, out; n_1, in), (n_3, out; n_2, in), (n_4, out; n_3, in), (\#_1, out; n_4, in)$. If there is at least one copy of c_α in membrane 2, then we can apply $(d_{c,q}, out; n_4, in) \in R_2''$ and then we finish the simulation by bringing q in membrane 1 by means of $(d_{c,q}, out; q, in) \in R_1''$. If $d_{c,q}$ was not present in membrane 4, then $\#_3$ will be released from membrane 4 by $(\#_3, out; c_q', in) \in R_4''$. It is clear that also these instructions are correctly simulated by our system, and also the process can be iterated.

It remains to discuss the case of rules $(p \to q, c = 0)$ from the counter automaton. The state p is replaced by c_q'' by $(p, out; c_q'', in) \in R_1''$, then this symbol will start to increment the subscripts of i when entering membrane 2: $(i_1, out; c_q'', in) \in R_2''$, at the next step the subscript of i is incremented in membrane 1 and also i_3 is pushed in membrane 2 by means of $(i_3, out; c_q'', in) \in R_3''$. At the next step the special marker $d_{c,q}'$ is brought out of membrane 4 by

means $(d'_{c,q}, out; c''_q, in) \in R''_4$ and the subscript of i is still incremented by $(i_3, out; i_2, in) \in R''_2$. Now $d'_{c,q}$ performs the checking for the counter c (whether it is zero or not): if there is at least one c_α present, then $d'_{c,q}$ will enter membrane 2, and at the next step will bring $\#_3$ from membrane 1 since the subscript of i did not reach position 5; on the other hand, if there are no copies of c in membrane 2, then $d'_{c,q}$ will sit unused in membrane 3 for one step until i_3 is brought from membrane 1 by $(i_4, out; i_3, in) \in R''_2$, then we apply the following rules: $(i_5, out; i_4, in) \in R''_2$ and $(d'_{c,q}, out; i_3, in) \in R''_3$. Next we can apply $(d'_{c,q}, out; i_5, in) \in R''_2$ and then in membrane 1 we finish the simulation by using $(d'_{c,q}, out; q, in) \in R''_1$. One can notice that all the symbols are in the same place as they were in the beginning of this simulation (i_3 is back in membrane 3, i_1, i_2, i_4, i_5 are in membrane 2, etc.), the only symbols moved are one copy of $d'_{c,q}$ which is now in the environment and c''_q which is in membrane 4. It is clear that we can iterate the process described above for all the types of rules in the counter automaton, so we correctly simulate the automaton.

The third phase, the finishing one, will stop the simulation and move the relevant objects into the output membrane. Specifically, when we reach a state $p \in F$ we can use the following rules:

$$\begin{aligned}
R'''_1 &= \{(p, out; h, in) \mid p \in F\},\\
R'''_2 &= \{(h, in)\},\\
R'''_3 &= \emptyset,\\
R'''_4 &= \emptyset,\\
R'''_5 &= \{(h', out; h, in), (h'', out; h, in), (h, in)\}\\
&\quad \cup \{(h, out; c_{out\alpha}, in) \mid \alpha \in Q\},\\
R'''_6 &= \{(c_{out\alpha}, in) \mid \alpha \in Q\}.
\end{aligned}$$

We first use $(p, out; h, in) \in R'''_1$, then h enters membrane 2 by (h, in) and at the next step h stops the oscillation of h' and h'' by putting them together in membrane 2 by means of $(h', out; h, in) \in R'''_5$ or $(h'', out; h, in) \in R'''_5$. After this h begins moving the content of output counter c_{out} into membrane 5 by using $(h, out; c_\alpha, in) \in R'''_5$. When the last c_α enters membrane 6 by using $(c_\alpha, in) \in R'''_6$ the system will be in a halting state only if a correct simulation was done in phases one and two, so the counter automaton was correctly simulated. This completes the proof.

5 Final Remarks

One can notice that membrane 6 was used only to collect the output. The same system without membrane 6 will simulate in the same way the counter automaton, but, when reaching the halt state will also contain the symbol h in the output membrane 5. This suggests that it could be possible to use a similar construct to improve the result from Theorem 3 to a result of the form:

Conjecture: $NOP_5(sym_1, anti_1) = RE$.

Obviously, P systems with minimal symport/antiport rules and using only one membrane can compute at most finite sets of numbers, at most as large as the number of objects present in the system in the initial configuration: the antiport rules do not increase the number of objects present in the system, the same with the symport rules of the form (a, out), while a symport rule of the form (a, in) should have $a \in V - E$ (otherwise the computation never stops, because the environment is inexhaustible).

The family $NOP_2(sym_1, anti_1)$ contains infinite sets of numbers. Consider, for instance, the system

$$\begin{aligned} \Pi &= (\{a, b\}, [_1[_2\ ]_2]_1, a, \lambda, \{b\}, R_1, R_2, 2), \\ R_1 &= \{(a, out; b, in),\ (a, in)\}, \\ R_2 &= \{(a, in),\ (b, in)\}. \end{aligned}$$

After bringing an arbitrary number of copies of b from the environment, the object a gets "hidden" in membrane 2, the output one.

An estimation of the size of families $NOP_m(sym_1, anti_1)$ for $m = 2, 3, 4, 5$ remains to be found.

The P systems with symport and antiport rules are interesting from several points of view: they have a precise biological inspiration, are mathematically elegant, the computation is done only by communication, by moving objects through membranes (hence the conservation law is observed), they are computationally complete. Thus, they deserve further investigations, including from the points of view mentioned above.

References

1. B. Alberts, *Essential Cell Biology. An Introduction to the Molecular Biology of the Cell*, Garland Publ. Inc., New York, London, 1998.
2. F. Bernardini, M. Gheorghe, On the Power of Minimal Symport/Antiport, *Workshop on Membrane Computing*, Tarragona, 2003.
3. R. Freund, A. Păun, Membrane Systems with Symport/Antiport: Universality Results, in *Membrane Computing. Intern. Workshop WMC-CdeA2002, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), *Lecture Notes in Computer Science*, 2597, Springer-Verlag, Berlin, 2003, 270–287.
4. P. Frisco, J.H. Hogeboom, Simulating Counter Automata by P Systems with Symport/Antiport, in *Membrane Computing. Intern. Workshop WMC-CdeA2002, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), *Lecture Notes in Computer Science*, 2597, Springer-Verlag, Berlin, 2003, 288–301.
5. J. Hopcroft, J. Ulmann, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
6. C. Martín-Vide, A. Păun, Gh. Păun, On the Power of P Systems with Symport and Antiport Rules, *J. of Universal Computer Sci.*, 8, 2 (2002) 317–331.
7. A. Păun, Gh. Păun, The Power of Communication: P Systems with Symport/Antiport, *New Generation Computing*, 20, 3 (2002) 295–306.
8. Gh. Păun, Computing with Membranes, *J. of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).

9. Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
10. Gh. Păun, M. Perez-Jimenez, F. Sancho-Caparrini, On the Reachability Problem for P Systems with Symport/Antiport, *Proc. Automata and Formal Languages Conf.*, Debrecen, Hungary, 2002.
11. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.