Communicative P Systems with Minimal Cooperation

Artiom Alhazov[1,2], Maurice Margenstern[3], Vladimir Rogozhin[4], Yurii Rogozhin[1], and Sergey Verlan[3]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
{artiom,rogozhin}@math.md
[2] Research Group on Mathematical Linguistics
Rovira i Virgili University, Tarragona, Spain
artiome.alhazov@estudiants.urv.es
[3] LITA, Université de Metz, France
{margens,verlan}@sciences.univ-metz.fr
[4] State University of Moldova
rv@math.md

Abstract. We proved that two classes of Communicative P systems with 3 membranes and with minimal cooperation, namely P systems with symport/antiport rules of size 1 and and P systems with symport rules of size 2 are computationally complete: they generate all recursively enumerable sets of vectors of nonnegative integers. The result of computation is obtained in the elementary membrane.

1 Introduction

P systems were introduced by Gheorghe Păun in [11] as distributed parallel computing devices of biochemical inspiration. The original definition is quite general and many different variants of P systems were proposed, see [15] for a comprehensive bibliography. One of these variants, P systems with *symport/antiport*, was introduced in [12]. This variant uses one of the most important features of membrane systems: the communication. This operations is so powerful, that it suffices by itself for a big computational power. 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 [12]. This result was improved with respect to the number of used membranes and/or the size of symport/antiport rules ([4], [6], [9], [13], [2], [8], [14]).

Rather unexpectedly, *minimal symport/antiport* P systems (membrane systems), *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 proof of this result may 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], five [2], four [5, 8] and at last G.Vaszil [14] showed that three membranes are sufficient to generate all recursively enumerable sets of numbers (but his proof had one disadvantage: the output membrane contains 5 additional symbols). In this paper we give another proof of the last result which was obtained independently. We also remark that in our proof the output membrane does not contain superfluous symbols.

Minimally cooperative symport P systems (membrane systems), *i.e.*, P systems only having symport rules and only moving one or two objects, are universal with four membranes [6]. In this paper we improve that result down to three membranes.

Our proofs of both results are based on a simulation of counter automata (or register machines [10]), see also [3], which was also used in [1], [4], [7] and [2].

The question about universality of P systems with minimal symport/antiport (symport) rules with 1 and 2 membranes is still open.

2 Basic notions

A non-deterministic *counter automaton* is the 5-tuple $M = (Q, q_0, q_f, C, P)$, where

- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $q_f \in Q$ is the final state,
- C is a finite set of counters,
- P is a finite set of instructions of the following form:
  1. $(q_i \to q_l, c_k+)$, with $q_i, q_l \in Q,\ q_i \neq q_f,\ c_k \in C$ "increment" instruction). This instruction increments counter c_k by 1 and changes the state of the system from q_i to q_l.
  2. $(q_i \to q_l, c_k-)$, with $q_i, q_l \in Q,\ q_i \neq q_f,\ c_k \in C$ "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. $(q_i \to q_l, c_k = 0)$, with $q_i, q_l \in Q,\ q_i \neq q_f,\ c_k \in C$ ("zero test" instruction). If the value of counter c_k is zero, then this instruction changes the state of the system from q_i to q_l. Otherwise (the value of c_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 be assigned only 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 types above and by changing the current state to another one. The computation starts in state q_0

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 $c_1 \in C$ when the computation halts in state $q_f \in Q$.

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 (symport) is a construct

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

where:

1. O is a finite alphabet of symbols called objects,
2. μ is a membrane structure consisting of m membranes that are labelled in a one-to-one manner by $1, 2, \ldots, k$.
3. $w_i \in O^*$, for each $1 \leq i \leq k$ is a *finite* multiset (*i.e.* multiset where elements are present in finite number of copies) of objects associated with the region i (delimited by membrane i),
4. $E \subseteq O$ is the set of objects that appear in the environment in 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), (y, out), $(y, out; x, in)$, where $x, y \in O^*$ (for symport P systems R_i consists rules of the form (x, in), (y, out) only),
6. 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 region 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 to 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 to the multiset x to be moved from region i into the outer region. A rule $(y, out; x, in)$ permits to 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 rules 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 objects that are presented in region i_0. Given a P system Π, the set of natural numbers computed in this way by Π is denoted by $N(\Pi)$. If the multiplicity of each object is counted separately, then a vector of natural numbers is obtained, denoted by $Ps\Pi$, see [13].

We denote by $NOP_m(sym_r, anti_t)$ ($NOP_m(sym_r)$) the family of sets of natural numbers 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 $max\{|x|, |y|\}$. We denote by NRE the family of recursively enumerable sets of natural numbers. If we replace numbers by vectors, then in the 3 notations of this paragraph N is replaced by Ps.

3 Main Results

Theorem 1. $NOP_3(sym_1, anti_1) = NRE$.

Proof. We prove this result in the following way. We shall simulate a nondeterministic counter automaton $M = (Q, q_0, q_f, C, P)$ which starts with empty counters. We also suppose that all instructions from P are labelled in a one-to-one manner with $\{1, \ldots, n\} = I$. Denote by I_+ ($I_+ \subseteq I$) a set of labels of "increment" instructions, by I_- ($I_- \subseteq I$) a set of labels of "decrement" instructions and by $I_{=0}$ ($I_{=0} \subseteq I$) is a set of labels of "zero test" instructions.

We construct a P system Π with the following membrane structure:

$$[_1\ [_2\ [_3\ \ ]_3\ ]_2\ ]_1$$

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

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

We code the counter automaton as follows. At each moment (after stage one) region 1 holds the current state of the automaton, represented by a symbol $q_i \in Q$, region 2 keeps the value of all counters, represented by the number of occurrences of symbols $c_k \in C$. We simulate the instructions of the counter automaton and we use for this simulation the symbols $c_k \in C$, $a_j, b_j, d_j, e_j,\ j \in I$. During the first stage we bring from the environment an arbitrary number of symbols b_j into region 3, symbols d_j into region 2 and symbols c_k into region 1. We suppose that we have enough symbols in the corresponding membranes to perform the computation. We also use the following idea: we bring from the environment symbols c_k into region 1 all time during the computation. This process

may be stopped only if all stages finish correctly. Otherwise, the computation will never stop.

We split our proof in 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 that we present is the union of all these parts.

We construct the P system Π as follows:

$$
\begin{aligned}
\Pi &= (O, [_1\, [_2\, [_3\ ]_3\,]_2\,]_1, w_1, w_2, w_3, E, R_1, R_2, R_3, 3),\\
O &= E \cup \{f_j \mid j \in I\} \cup \{m_1 \mid 1 \le i \le 5\}\\
&\quad \cup \{l_7, l_8, g_1, g_2, g_3, I_a, I_1, I_2, I_3, I_c, O_b, O_2, i, t, \#_0, \#_1, \#_2\},\\
E &= \{a_j, b_j, d_j, e_j \mid j \in I\} \cup \{c_k \mid c_k \in C\}\\
&\quad \cup \{q_i \mid q_i \in Q\} \cup \{l_i \mid 1 \le i \le 6\},\\
w_1 &= I_1 I_2 I_3 O_2 g_2 i l_7 l_8 \#_1 \#_2,\\
w_2 &= I_c t m_1 m_2 \#_0,\\
w_3 &= I_a O_b g_1 g_3 m_3 m_4 m_5 \prod_{j \in I} f_j,\\
R_i &= R_{i,s} \cup R_{i,r} \cup R_{i,f} \cup R_{i,a},\ 1 \le i \le 3.
\end{aligned}
$$

The rules are given by phases: START (stage 1), RUN (stage 2), FIN (stage 3) and AUX.

AUX.

$$
\begin{aligned}
R_{1,a} &= \{\mathtt{1a1} : (I_c, in), \mathtt{1a2} : (I_1, in)\} \cup \{\mathtt{1a3} : (I_c, out; c_k, in) \mid c_k \in C\}\\
&\quad \cup \{\mathtt{1a4} : (I_1, out; b_j, in) \mid j \in I\} \cup \{\mathtt{1a5} : (I_1, out; d_j, in) \mid j \in I_{=0}\}\\
&\quad \cup \{\mathtt{1a6} : (\#_0, in), \mathtt{1a7} : (\#_0, out)\},\\
R_{2,a} &= \{\mathtt{2a1} : (O_b, out), \mathtt{2a2} : (I_a, in), \mathtt{2a3} : (I_2, in)\}\\
&\quad \cup \{\mathtt{2a4} : (b_j, out; O_b, in) \mid j \in I_-\} \cup \{\mathtt{2a5} : (I_a, out; a_j, in) \mid j \in I_+\}\\
&\quad \cup \{\mathtt{2a6} : (I_2, out; b_j, in) \mid j \in I\} \cup \{\mathtt{2a7} : (I_2, out; d_j, in) \mid j \in I_{=0}\},\\
R_{3,a} &= \{\mathtt{3a1} : (O_2, out), \mathtt{3a2} : (I_3, in), \mathtt{3a3} : (I_3, out; c_1, in)\}\\
&\quad \cup \{\mathtt{3a4} : (x, out; O_2, in) \mid x \in \{I_1, I_2, g_2, l_1, l_2, l_3, l_7\}\}\\
&\quad \cup \{\mathtt{3a5} : (a_j, out; O_2, in) \mid j \in I\}\\
&\quad \cup \{\mathtt{3a6} : (\#_i, in), \mathtt{3a7} : (\#_i, out) \mid 1 \le i \le 2\}.
\end{aligned}
$$

Symbols I_a, I_1, I_2, I_3, I_c bring symbols inside some membrane and return. Symbols O_1, O_b take symbols outside some membrane and return. Symbols $\#_0, \#_1, \#_2$ check for "invalid" computation.

START.

$$
\begin{aligned}
R_{1,s} &= \{\texttt{1s1} : (g_3, out; q_0, in)\},\\
R_{2,s} &= \{\texttt{2s1} : (I_2, out; \#_1, in), \texttt{2s2} : (t, out; I_1, in), \texttt{2s3} : (I_2, out; t, in)\}\\
&\quad \cup \{\texttt{2s4} : (g_1, out; g_2, in), \texttt{2s5} : (I_c, out; g_1, in), \texttt{2s6} : (g_3, out; i, in)\},\\
R_{3,s} &= \{\texttt{3s1} : (b_j, in) \mid j \in I\} \cup \{\texttt{3s2} : (g_1, out; I_1, in), \texttt{3s3} : (g_3, out; g_2, in)\}\\
&\quad \cup \{\texttt{3s4} : (I_1, out; I_2, in), \texttt{3s5} : (O_b, out; I_1, in), \texttt{3s6} : (I_a, out; i, in)\}.
\end{aligned}
$$

Symbols I_1, I_2 bring from environment "sufficiently many" symbols d_j in region 2 and a "correct number of" symbols b_j in region 3 for the computation (rules `1a4,2a3,1a2,2a6,1a5,3s1,2a7`). We illustrate this process by Figure 1.

The figures in this paper describe different stages of 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 object in a corresponding membrane (symbols outside of the rectangle are 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 $\Rightarrow$ symbol.

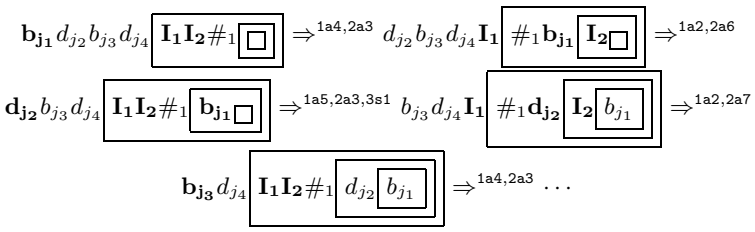

Fig. 1. Bringing objects b_j, d_j.

Notice that I_2 cannot be idle, as it immediately leads to infinite computation (rules `2s1,3a6,3a7`), so d_j and b_j in region 1 must be moved to region 2 by I_2 (rules `2a6` and `2a7`).

At some point, I_1 stops bringing symbols d_j,b_j. I_1 and I_2 are removed from their "pumping" positions, I_c is placed in region 1, where it can "pump" symbols c_k into the skin membrane, and q_0 is brought into region 1 to start the simulation of the register machine. In the meantime I_a reaches region 2 and O_b reaches region 1. Notice that both $(g_1, out; I_1, in)$ and $(O_b, out; I_1, in)$ from $R_{3,s}$ are applied, in either order (Figure 2).

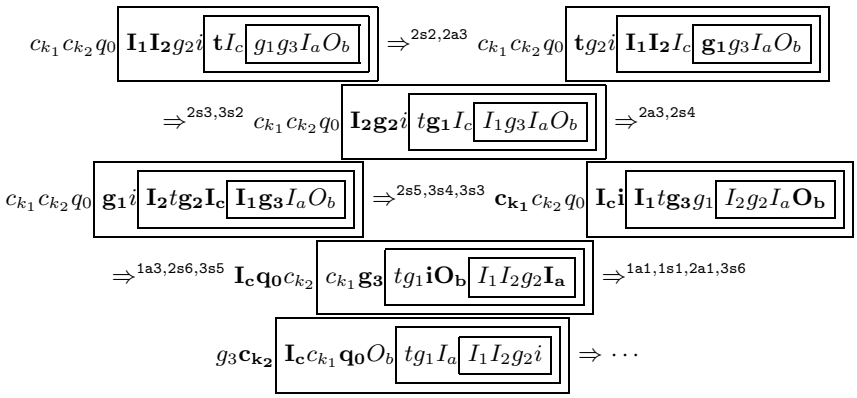

Fig. 2. Ending of the initialization (stage 1).

RUN.

$$
\begin{aligned}
R_{1,r} = {} & \{\texttt{1r1} : (q_i, out; a_j, in), \texttt{1r2} : (b_j, out; q_l, in) \\
& \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, = 0\}\} \\
& \cup \{\texttt{1r3} : (d_j, out; e_j, in) \mid (j : q_i \to q_l, c_k = 0) \in P\}, \\
R_{2,r} = {} & \{\texttt{2r1} : (b_j, out; c_k, in) \mid (j : q_i \to q_l, c_k+) \in P\}\} \\
& \cup \{\texttt{2r2} : (c_k, out; a_j, in) \mid (j : q_i \to q_l, c_k-) \in P\} \\
& \cup \{\texttt{2r3} : (d_j, out; a_j, in), \texttt{2r4} : (c_k, out; e_j, in), \\
& \quad \texttt{2r5} : (b_j, out; e_j, in) \mid (j : q_i \to q_l, c_k = 0) \in P\}, \\
R_{3,r} = {} & \{\texttt{3r1} : (b_j, out; a_j, in) \mid (j : q_i \to q_l, c_k+) \in P\} \\
& \cup \{\texttt{3r2} : (b_j, out; a_j, in) \mid (j : q_i \to q_l, c_k-) \in P\} \\
& \cup \{\texttt{3r3} : (f_j, out; a_j, in), \texttt{3r4} : (b_j, out; f_j, in) \\
& \quad \mid (j : q_i \to q_l, c_k = 0) \in P\}.
\end{aligned}
$$

While I_c is bringing symbols c_k into the skin membrane (rules `1a1,1a3`), instructions $(j : q_i \to q_l, c_k\gamma)$, $\gamma \in \{+, -, = 0\}$ of the register machine are simulated.

"Increment" instruction:

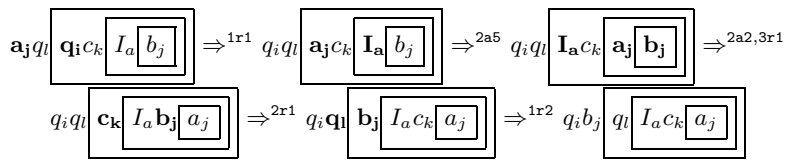

Fig. 3. q_i replaced by q_l, c_k moved into region 2.

"Decrement" instruction:

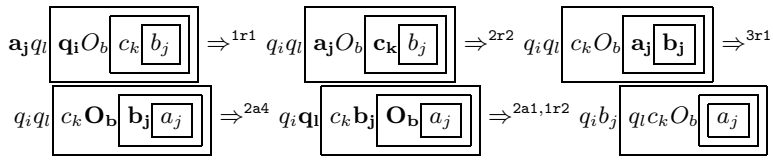

Fig. 4. q_i replaced by q_l, c_k removed from region 2.

Checking for zero. q_i replaced by q_l if there is no c_k in region 2 (Figure 5), otherwise e_j exchanges with c_k and b_j remains in region 2 (Figure 6).

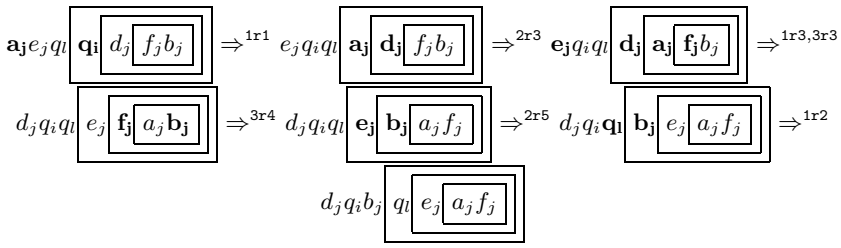

Fig. 5. "Zero test" instruction. There is no c_k in region 2.

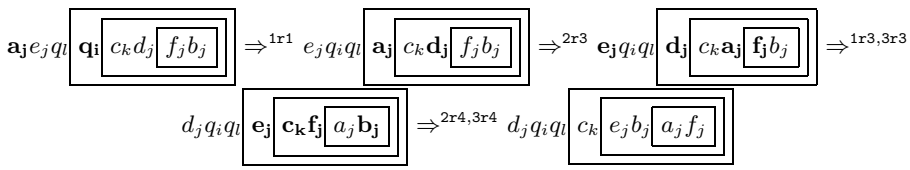

Fig. 6. "Zero test" instruction. There is c_k in region 2.

FIN.

$$
\begin{aligned}
R_{1,f} &= \{\texttt{1f1} : (m_1, out; l_1, in), \texttt{1f2} : (\#_1, out; m_1, in), \texttt{1f3} : (m_2, out; l_2, in)\} \\
&\quad \cup \{\texttt{1f4} : (m_3, out; l_3, in), \texttt{1f5} : (m_4, out; l_4, in), \texttt{1f6} : (l_4, out; l_5, in)\} \\
&\quad \cup \{\texttt{1f7} : (m_5, out; l_6, in)\}, \\
R_{2,f} &= \{\texttt{2f1} : (m_1, out; q_f, in), \texttt{2f2} : (q_f, out; l_7, in), \texttt{2f3} : (m_2, out; l_1, in)\} \\
&\quad \cup \{\texttt{2f4} : (m_3, out; O_2, in), \texttt{2f5} : (m_4, out; I_3, in), \texttt{2f6} : (I_3, out; l_2, in)\} \\
&\quad \cup \{\texttt{2f7} : (m_5, out; l_8, in), \texttt{2f8} : (l_8, out; I_c, in), \texttt{2f9} : (c_1, out; l_6, in)\} \\
&\quad \cup \{\texttt{2fa} : (l_6, out; \#_2, in), \texttt{2fb} : (l_3, in), \texttt{2fc} : (\#_0, out, l_5, in)\} \\
&\quad \cup \{\texttt{2fd} : (l_3, out, l_5, in)\}, \\
R_{3,f} &= \{\texttt{3f1} : (m_3, out; l_7, in), \texttt{3f2} : (m_4, out; l_1, in), \texttt{3f3} : (m_5, out; l_2, in)\} \\
&\quad \cup \{\texttt{3f4} : (b_j, out; l_3, in) \mid j \in I\}.
\end{aligned}
$$

If a successful computation of the register machine is correctly simulated, then q_f will appear in region 1. $\#_1$ is removed from region 1, and a chain reaction is started, during which symbols l_i move inside the membrane structure, and symbols m_i move outside the membrane structure (Figure 7).

$$
\begin{aligned}
& l_1 l_1 l_2 \left[\mathbf{q_f} l_7 O_2 I_3 \#_1 \left[\mathbf{m_1} m_2 \left[m_3 m_4 \right] \right] \right] \Rightarrow^{\texttt{2f1}} \mathbf{l_1} l_1 l_2 \left[\mathbf{m_1 l_7} O_2 I_3 \#_1 \left[\mathbf{q_f} m_2 \left[m_3 m_4 \right] \right] \right] \Rightarrow^{\texttt{1f1,2f2}} \\
& \mathbf{m_1} l_1 l_2 \left[\mathbf{l_1} q_f O_2 I_3 \#_1 \left[\mathbf{l_7 m_2} \left[\mathbf{m_3} m_4 \right] \right] \right] \Rightarrow^{\texttt{1f2,2f3,3f1}} \#_1 \mathbf{l_1 l_2} \left[\mathbf{m_2} q_f \mathbf{O_2 m_1} I_3 \left[\mathbf{m_3 l_1} \left[l_7 \mathbf{m_4} \right] \right] \right] \\
& \Rightarrow^{\texttt{1f1,1f3,2f4,3f2}} \#_1 m_1 m_2 \left[l_2 q_f m_3 l_1 I_3 \left[O_2 m_4 \left[l_7 l_1 \right] \right] \right].
\end{aligned}
$$

Fig. 7. Beginning of the termination (stage 3).

Now O_2 will pump outside the elementary membrane any symbol which stays there, except c_1 (rules `3a1, 3a4, 3a5`). m_4 will exchange with I_3 (rule `2f5`), and the latter will pump symbols c_1 into the elementary membrane (rules `3a2, 3a3`), and eventually exchange with l_2 (rule `2f6`).

m_3 comes to the environment in exchange for l_3 (rule `1f4`), which goes to membrane 2 (rule `2fb`), and stays there if there is no object b_j in the elementary membrane (otherwise l_3 will exchange with b_j by rule `3f4`). m_4 comes to the environment in exchange for l_4 (rule `1f5`), which brings l_5 in the skin. l_5 then exchanges with l_3 by rule `2fd`. Notice that presence of b_j in region 3 will force l_5 to move $\#_0$ in region 1 (rule `2fc`), leading to an infinite computation (rules `1a6, 1a7`), as l_3 will be situated in region 3.

Finally (after I_3 returns to region 1 and l_2 comes in region 2 by rule `2f6`), l_2 moves m_5 into region 2 (rule `3f3`), and the latter exchanges with l_8 (rule `2f7`) and then with l_6 (rule `1f7`). At some point l_8 moves I_c into region 2 (rule `2f8`),

to finish pumping objects c_k. As for l_6 in membrane 1, it guarantees that no more objects c_1 remain in membrane 2 (otherwise it moves $\#_2$ in membrane 2 (rules `2f9, 2fa`), leading to an infinite computation (rule `3a6, 3a7`)).

If the computation halts, then the elementary membrane will only contain objects c_1, in the multiplicity of the value of the first register of the register machine. Conversely, any computation of the register machine allows a correct simulation (from the construction). Thus, the class of P systems with symport and antiport of weight 1 generate exactly all recursively enumerable sets of non-negative integers. □

A "dual" class of systems $OP(sym_1, anty_1)$ 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 the other class.

Theorem 2. $NOP_3(sym_2) = NRE$.

Proof. As in the proof of Theorem 1 we simulate a non-deterministic counter automaton $M = (Q, q_0, q_f, C, P)$ which starts with empty counters. Again we suppose that all instructions from P are labelled in a one-to-one manner with $\{1, \ldots, n\} = I$, and I_+ ($I_+ \subseteq I$) is a set of labels of "increment" instructions, I_- ($I_- \subseteq I$) is a set of labels of "decrement" instructions, and $I_{=0}$ ($I_{=0} \subseteq I$) is a set of labels of "zero test" instructions.

We construct the P system Π_2 as follows:

$$
\begin{aligned}
\Pi_2 &= (O, E, [_1\, [_2\, [_3\ ]_3\,]_2\,]_1, w_1, w_2, w_3, R_1, R_2, R_3, 3),\\
O &= E \cup \{d_j, e_j \mid j \in I\} \cup \{t_i \mid 0 \le i \le 10\}\\
&\quad \cup \{g_1, g_3, I_a, I_1, I_2, I_c, O_b, \#_1, \#_2\}\\
&\quad \cup \{q_i \mid q_i \in Q\},\\
E &= \{a_j, b_j \mid j \in I\} \cup \{c_k \mid c_k \in C\} \cup \{l_i \mid 3 \le i \le 8\} \cup \{g_2\},\\
w_1 &= t_0 t_1 t_2 t_3 t_4 I_1 I_2 I_a l_1 l_2 \#_1 \prod_{j \in I} e_j \prod_{q_i \in Q} q_i,\\
w_2 &= t_5 t_6 t_7 t_8 t_9 t_{10} I_c g_3 s_1 m_2 \#_2 \prod_{j \in I} d_j,\\
w_3 &= g_1 O_b s_2 m_1,\\
R_i &= R_{i,s} \cup R_{i,r} \cup R_{i,m} \cup R_{i,c} \cup R_{i,f} \cup R_{i,a},\ 1 \le i \le 3.
\end{aligned}
$$

The functioning of this system may be split in three stages as it done in Theorem 1.

We code the counter automaton as follows. At each moment (after stage one) the environment holds the current state of the automaton, represented by a symbol $q_i \in Q$, the membrane 2 holds the value of all counters, represented by the number of occurrences of symbols $c_k \in C$. We simulate the instructions of the counter automaton and we use for this simulation the symbols $c_k \in C$, a_j, b_j, d_j, e_j, $j \in I$. During the first stage we bring from environment in the membrane 3 an arbitrary number of symbols b_j. We suppose that we have enough

symbols b_j in membrane 3 to perform the computation. We also use the following idea: we bring from environment to membrane 1 the symbols c_k all time during the computation. This process 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 behavior 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.

The rules R_i are given by phases: START (stage 1); RUN (stage 2); MOVE, CLEANUP and FIN (stage 3), and AUX.

AUX.

$$\begin{aligned}
R_{1,a} &= \{\texttt{1a1} : (I_c, out), \texttt{1a2} : (I_1, out)\} \cup \{\texttt{1a3} : (I_c c_k, in) \mid c_k \in C\} \\
&\quad \cup \{\texttt{1a4} : (I_1 b_j, in) \mid j \in I\} \cup \{\texttt{1a5} : (\#_2, in), \texttt{1a6} : (\#_2, out)\}, \\
R_{2,a} &= \{\texttt{2a1} : (O_b, in), \texttt{2a2} : (I_a, out), \texttt{2a3} : (I_2, out)\} \\
&\quad \cup \{\texttt{2a4} : (O_b b_j, out) \mid j \in I_+\} \cup \{\texttt{2a5} : (I_a a_j, in) \mid j \in I_-\} \\
&\quad \cup \{\texttt{2a6} : (I_2 b_j, in) \mid j \in I\}, \\
R_{3,a} &= \{\texttt{3a1} : (\#_1, in), \texttt{3a2} : (\#_1, out)\} \\
&\quad \cup \{\texttt{3a3} : (s_i, in), \texttt{3a4} : (s_i, out) \mid 1 \leq i \leq 2\}.
\end{aligned}$$

Symbol I_1 brings symbols b_j inside membrane 1 and returns to the environment. Symbol I_c brings symbols c_k inside membrane 1 and returns to the environment. Symbol I_2 brings symbols b_j inside membrane 2 and returns to membrane 1. Symbol I_a brings symbols a_j inside membrane 2 and returns to membrane 1. Symbol O_b takes symbols b_j outside membrane 2 and returns. Symbols $\#_1, \#_2$ check for "invalid" computations. Symbols s_1, s_2 remember whether the derivation step is even or odd.

START.

$$\begin{aligned}
R_{1,s} &= \{\texttt{1s1} : (g_1 t_2, out), \texttt{1s2} : (t_2 g_2, in), \texttt{1s3} : (g_3 q_0, out)\}, \\
R_{2,s} &= \{\texttt{2s1} : (t_0 I_2, in), \texttt{2s2} : (I_2 \#_1, in), \texttt{2s3} : (I_1 t_1, in)\} \\
&\quad \cup \{\texttt{2s4} : (g_1 I_c, out), \texttt{2s5} : (g_2 t_3, in), \texttt{2s6} : (t_3 g_3, out)\}, \\
R_{3,s} &= \{\texttt{3s1} : (b_j, in) \mid j \in I\} \cup \{\texttt{3s2} : (I_2 t_1, in), \texttt{3s3} : (I_1 t_7, in)\} \\
&\quad \cup \{\texttt{3s4} : (t_7 g_1, out), \texttt{3s5} : (g_2 t_{10}, in), \texttt{3s6} : (t_{10} O_b, out)\}.
\end{aligned}$$

Symbols I_1, I_2 bring from environment a "correct number of" symbols b_j in region 3 for the computation (rules `1a2, 1a4, 2a6, 2a3, 3s1`) (see Figure 8). Notice that I_2 cannot be idle, as it immediately leads to infinite computation (rules `2s2, 3a1, 3a2`), so b_j in region 1 must be moved by I_2 by rule `2a6`.

At some point, I_1 stops bringing symbols b_j. I_1 and I_2 are removed from their "pumping" positions, I_c is placed in region 1, where it can "pump" symbols c_k into the skin membrane, and q_0 is brought into the environment to start the simulation of the register machine. In the meantime O_b reaches region 2 (Figure 9).

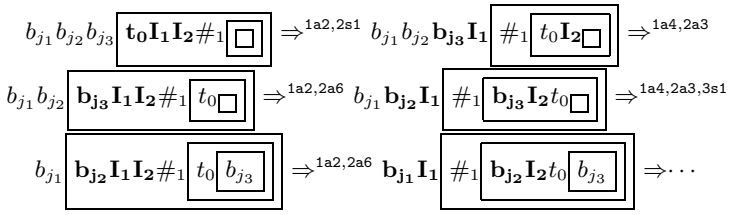

Fig. 8. Bringing objects b_j.

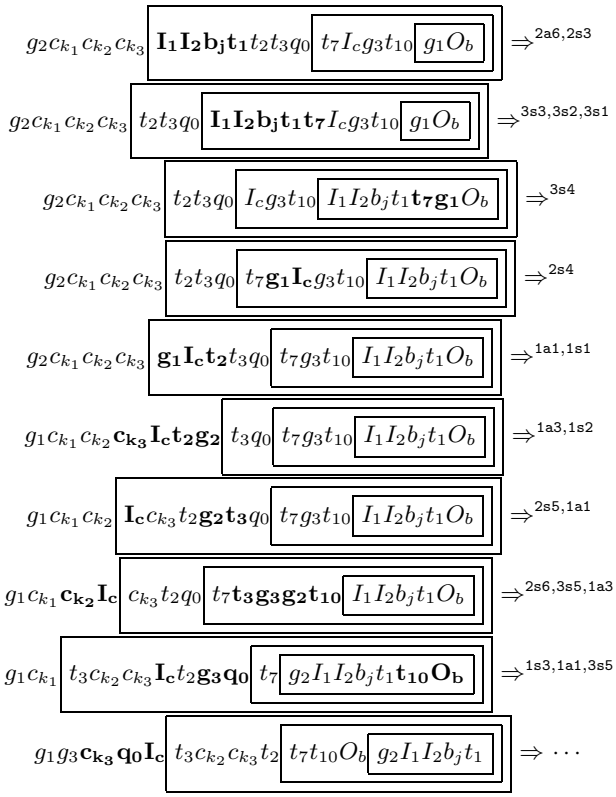

Fig. 9. End of the initialization (stage 1).

RUN.

$$
\begin{aligned}
R_{1,r} &= \{\texttt{1r1} : (q_i a_j, in), \texttt{1r2} : (b_j q_l, out) \\
&\quad \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, = 0\}\} \\
R_{2,r} &= \{\texttt{2r1} : (a_j c_k, in) \mid (j : q_i \to q_l, c_k+) \in P\} \\
&\quad \cup \{\texttt{2r2} : (b_j c_k, out) \mid (j : q_i \to q_l, c_k-) \in P\} \\
&\quad \cup \{\texttt{2r3} : (a_j e_j, in), \texttt{2r4} : (e_j c_k, out), \\
&\qquad \texttt{2r5} : (e_j b_j, out) \mid (j : q_i \to q_l, c_k = 0) \in P\}, \\
R_{3,r} &= \{\texttt{3r1} : (a_j d_j, in), \texttt{3r2} : (d_j b_j, out) \\
&\quad \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, = 0\}\}.
\end{aligned}
$$

While I_c is bringing symbols c_k into the skin membrane (rules `1a1, 1a3`), instructions $(j : q_i \to q_l, c_k\gamma)$, $\gamma \in \{+, -, = 0\}$ of the register machine are simulated.

"Increment" instruction:

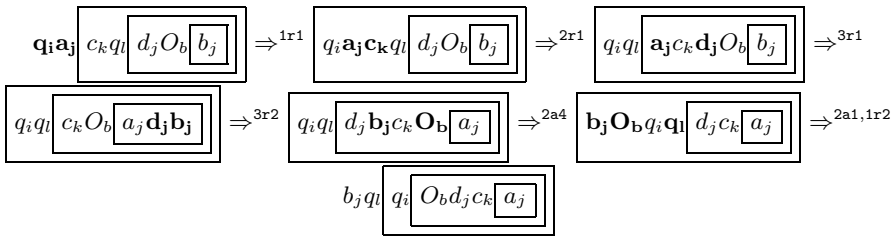

Fig. 10. q_i replaced by q_l, c_k moved into region 2.

"Decrement" instruction:

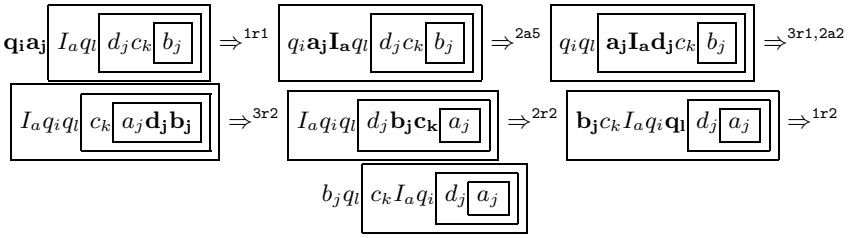

Fig. 11. q_i replaced by q_l, c_k removed from region 2.

Checking for zero. q_i replaced by q_l if there is no c_k in region 2 (Figure 12), otherwise e_j comes in region 1 with c_k and b_j remains in region 2 (Figure 13).

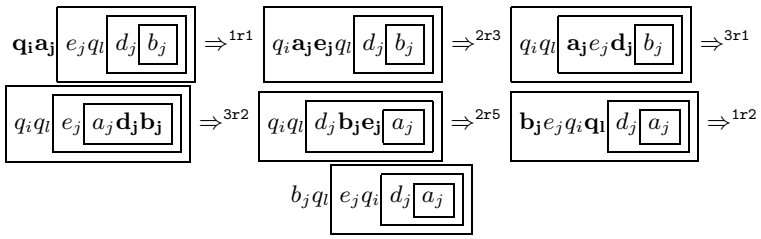

Fig. 12. "Zero test" instruction. There is no c_k in region 2.

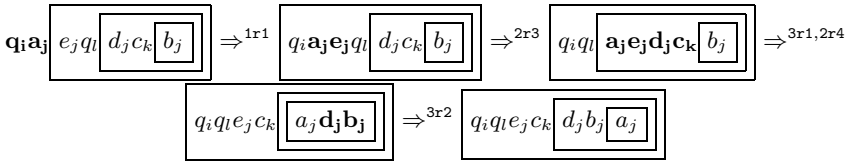

Fig. 13. "Zero test" instruction. There is c_k in region 2.

MOVE.

$$R_{1,m} = \{\texttt{1m1} : (q_f l_3, in), \texttt{1m2} : (m_1 t_4, out), \texttt{1m3} : (t_4 l_4, in)\},$$
$$R_{2,m} = \{\texttt{2m1} : (l_3 l_1, in), \texttt{2m2} : (m_1 t_6, out), \texttt{2m3} : (t_6 l_2, in), \texttt{2m4} : (l_2 \#_2, out)\},$$
$$R_{3,m} = \{\texttt{3m1} : (l_1 c_1, in), \texttt{3m2} : (l_1, out), \texttt{3m3} : (l_3 t_5, in)\}$$
$$\cup \{\texttt{3m4} : (t_5 m_1, out), \texttt{3m5} : (l_2 t_8, in)\} \cup \{\texttt{3m6} : (l_2 b_j, out) \mid j \in I\}.$$

If a successful computation of the register machine is correctly simulated, then q_f will appear in region 1. A chain reaction is started, during which symbols l_i move inside the membrane structure, and symbols m_i move outside the membrane structure. Notice that q_f brings l_3 into region 1 (rule `1m1`), then l_3 brings l_1 into region 2 (`rule 2m1`), then l_1 moves objects c_1 from region 2 into region 3 by rules `3m1` and `3m2`. Also, the system verifies that no objects b_j are present in the inner region (otherwise l_2 would bring $\#_2$ in region 1 (rules `3m6, 2m4`) and it immediately leads to infinite computation (rules `1a5,1a6`)) and moves l_4 into the skin membrane, as shown below (Figure 14).

CLEANUP.

$$R_{1,c} = \{\texttt{1c1} : (l_4 s_1, out), \texttt{1c2} : (s_1 l_5, in), \texttt{1c3} : (m_2 \#_1, out)\}$$
$$\cup \{\texttt{1c4} : (l_5 s_2, out), \texttt{1c5} : (s_2 l_7, in), \texttt{1c6} : (l_6 s_2, in)\},$$
$$R_{2,c} = \{\texttt{2c1} : (l_4, in), \texttt{2c2} : (l_4 s_1, out), \texttt{2c3} : (l_5 t_9, in)\}$$
$$\cup \{\texttt{2c4} : (t_9 m_2, out), \texttt{2c5} : (l_5 s_2, out)\},$$
$$\cup \{\texttt{2c6} : (l_4 x, out) \mid x \in \{t_5, t_7, t_{10}\} \cup \{d_j \mid j \in I\}\},$$
$$R_{3,c} = \{\texttt{3c1} : (l_5, in), \texttt{3c2} : (l_5 s_2, out)\}$$
$$\cup \{\texttt{3c3} : (l_5 x, out) \mid x \in \{I_1, I_2, g_2, t_8, l_3\} \cup \{a_j \mid j \in I\}\}.$$

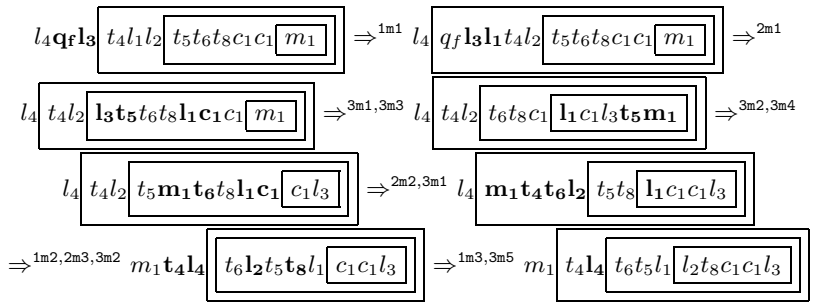

Fig. 14. Beginning of the termination (stage 3).

Objects d_j, $j \in I$ and t_5, t_7, t_{10} are removed from region 2, and then objects a_j, $j \in I$ and I_1, I_2, g_2, t_8, l_3 are removed from the inner region. Notice that l_4 only "meets" s_1 (and l_5 only "meets" s_2) after the corresponding cleanup is completed. Really, it is easily to see that object l_4 will be in region 2 after odd steps of computation. Symbol s_1 after odd steps of computation will be located in region 3 (rules `3a3,3a4`). Thus we cannot apply rule `2c2` and can apply rule `2c6` only, until all symbols t_5, t_7, t_{10} and d_j, $j \in I$ will be removed to region 1. After that symbol l_4 waits one step and together with symbol s_1 moves to region 1 and finally to the environment (rules `2c2` and `1c1`).

So l_4 will be in the environment after even steps of computation and object l_5 will appear in region 3 after odd steps of computation (rules `1c2`, `2c3` and `3c1`). Notice that symbol s_2 can appear in region 3 after even steps of computation (rules `3a4,3a3`). Thus we cannot apply rule `3c2` and can apply rule `3c3` only, until all symbols I_1, I_2, g_2, t_8, l_3 and a_j, $j \in I$ will be removed to region 2. After that object l_5 moves to the environment together with symbol s_2 (rules `3c2,2c5,1c4`) and object l_6 is brought in region 1 (rule `1c6`). At that moment in membrane 3 among symbols c_1 there are only two "undesirable" symbols: t_1 and l_2.

FIN.

$$
\begin{aligned}
R_{1,f} &= \{\texttt{1f1} : (l_6 t_1, out), \texttt{1f2} : (t_1 l_7, in), \texttt{1f3} : (l_7 l_2, out), \texttt{1f4} : (l_2 l_8, in)\},\\
R_{2,f} &= \{\texttt{2f1} : (l_6, in), \texttt{2f2} : (l_6 t_1, out), \texttt{2f3} : (l_7, in)\}\\
&\quad \cup \{\texttt{2f4} : (l_7 l_2, out), \texttt{2f5} : (l_8 I_c, in)\},\\
R_{3,f} &= \{\texttt{3f1} : (l_6, in), \texttt{3f2} : (l_6 t_1, out), \texttt{3f3} : (l_7, in), \texttt{3f4} : (l_7 l_2, out)\}.
\end{aligned}
$$

Objects t_1 and l_2 are removed from the inner region, as shown below (Figure 15), and then l_8 moves I_c from region 1 into region 2 (rule `2f5`) so that the computation can halt.

If the computation halts, then the elementary membrane will only contain objects c_1, in the multiplicity of the value of the first register of the register machine. Conversely, any computation of the register machine allows a correct

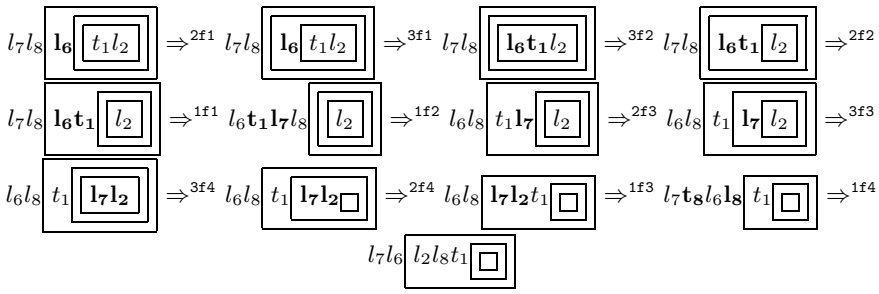

Fig. 15. End of the termination.

simulation (from the construction). Thus, the class of P systems with symport of weight 2 generate exactly all recursively enumerable sets of nonnegative integers.

4 Final Remarks

Both constructions can be easily modified to show $PsOP_3(sym_1, anti_1) = PsRE$ and $PsOP_3(sym_2) = PsRE$ by moving all output symbols c_k to the elementary membrane, as it is done for symbol c_1. In the proof of Theorem 1 we simply change rule `3a3`: $(I_3, out; c_1, in)$ by rules `3a3`: $(I_3, out; c_k, in)$ for all $c_k \in C$ and in the proof of Theorem 2 change rule `3m1`: (l_1c_1, in) by rules `3m1`: (l_1c_k, in) for all $c_k \in C$.

The questions what is the size of families of numbers computed by minimal symport/antipot (symport) P systems rules with 1 and 2 membranes is still open.

Program check

P systems in both theorems were checked for errors by the third author using a modification of a program that simulates P systems, originally developed by the first author.

Acknowledgements The first author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona. The all authors acknowledge the project IST-2001-32008 "MolCoNet" and the third and the fourth 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/Antypot. In A.Alhazov, C.Martin-Vide, Gh.Păun (eds.): Workshop on Membrane Computing,

WMC-2003, Tarragona, July 17–22, 2003, Technical Rewport 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 et al. (Eds.): WMC 2003, LNCS **2933** (2004) 43–45.
3. R.Freund, M.Oswald: GP systems with forbidding context. *Fundamenta Informaticae*, **49**, vol.1-3, (2002) 81–102.
4. R.Freund, A.Păun: Membrane Systems with Symport/Antiport: Universality Results. 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. M.Margenstern, V.Rogozhin, Y.Rogozhin, S.Verlan: About P systems with minimal symport/antiport rules and four membranes. In Pre-Proceedings of Fifth Workshop on Membrane Computing (WMC5), (G.Mauri, G.Paun, C.Zandron, Eds.), Universita di Milano-Bicocca, Italy, June 14 - 16, (2004) 283 - 294.
9. 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.
10. M.L.Minsky: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, New Jersey (1967).
11. Gh.Păun: Computing with Membranes. Journal of Computer and Systems Science **61**, (2000) 108–143.
12. A.Păun, Gh.Păun: The Power of Communication: P systems with Symport/Antiport. New Generation Computing **20**, (2002) 295–305.
13. Gh.Păun: Membrane computing. An Introduction. Springer-Verlag (2002).
14. G.Vaszil: On the size of P systems with minimal symport/antiport. Pre-Proceedings of Fifth Workshop on Membrane Computing (WMC5), (G.Mauri, G.Paun, C.Zandron, Eds.), Universita di Milano-Bicocca, Italy, June 14 - 16, (2004) 422 - 431.
15. C. Zandron. The P systems web page. `http://psystems.disco.unimib.it/`.