Computer Science Journal of Moldova, vol.**14**, no.3(42), 2006

Generating Languages by P Systems with Minimal Symport/Antiport *

Artiom Alhazov Yurii Rogozhin

Abstract

It is known that P systems with two membranes and minimal symport/antiport rules are "almost" computationally complete as generators of number or vector sets.

Interpreting the result of the computation as the sequence of terminal symbols sent to the environment, we show that P systems with two membranes and symport rules of weight two or symport/antiport rules of weight one generate all recursively enumerable languages.

1 Introduction

Membrane System (also called P system), which is a model of living cell, was introduced by Gh. Păun recently ([21, 22]). Membrane systems are distributed parallel computing devices, processing multisets of objects, synchronously, in compartments delimited by a membrane structure. The objects, which correspond to chemicals evolving in the compartments of a cell, can also pass through membranes. The membranes form a hierarchical structure (they can be dissolved, divided, created, and their permeability can be modified). A sequence of transitions between configurations of a P system forms a computation. The result of a halting computation is the number of objects present at the end of the computation in a specified membrane, called the output membrane. The objects can also have a structure of their own that can

*The authors acknowledge the project 06.411.03.04P from the Supreme Council for Science and Technological Development of the Academy of Sciences of Moldova.

be described by strings over a given alphabet of basic molecules - in this case the result of a computation is a set of strings. An important version of membrane systems deals with membranes arranged not in a hierarchical structure (which mathematically corresponds to a tree), but in a tissue-like structure (which mathematically corresponds to a graph). Many open problems related to the computational power of P systems remain. The challenge is especially interesting, and apparently difficult, for restricted versions of P systems, in our case for P systems with symport/antiport rules.

P systems with symport/antiport rules are parallel distributed systems, processing multisets according the rules that move the objects between the regions. They were first introduced in [20]; symport rules move objects across a membrane (which is a separator of regions) together in one direction, whereas antiport rules move objects across a membrane in opposite directions.

A comprehensive overview of the most important results obtained in the area of P systems and tissue P systems with *symport/antiport* rules, with respect to the development of computational completeness results improving descriptional complexity parameters as the number of membranes and cells, respectively, the weight of the rules and the number of objects can be found in [1] and the last results in [5]. We consider P systems with symport/antiport rules and minimal cooperation, i.e., P systems with symport/antiport rules of weight one and P systems with symport rules of weight two, such systems are called P systems with minimal symport/antiport.

In this paper we consider generating languages by such P systems with two membranes, in the way it has been done, e.g., for transitional P systems and for P systems with active membranes, see [22]. This is a natural generalization of studies of the generative power of P systems with symport/antiport. Since the environment in this model is considered not empty at the beginning of the computation, we distinguish the result by considering the sequence of objects from a terminal sub-alphabet, sent into the environment.

2 Preliminaries

Let O be a finite set, O^* is a free monoid generated by O. Consider $x \in O^*$; we will denote all permutations of x by $Perm(x)$.

By RE we denote the family of recursively enumerable languages.

2.1 Counter Automata with an Output Tape

A non-deterministic counter automaton (see [9], [1]) with an output tape is a tuple

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

- d is the number of counters
  (we will use the notation $D = \{1, \cdots, d\}$);
- T is an output alphabet;
- Q is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $q_f \in Q$ is the final state;
- P is a finite set of instructions.

The instructions operating on counters are of the forms

$$(q_i \to q_l, k\gamma), \text{ with } q_i, q_j \in Q,\ q_i \neq q_f,\ k \in D,\ \gamma \in \{+, -, =0\}$$

(changing the state from q_i to q_l and applying operation γ to counter k). The operations are *increment* (add one to the value of the counter), *decrement* (subtract one from the value of the counter) and *zero-test* (test whether the value of the counter is zero or not), respectively. If an empty counter is decremented or a non-empty counter is tested for zero, then the computation is blocked.

The instructions operating on the tape are of the form

$$(q_i \to q_l, write(c)), \text{ where } q_i, q_j \in Q,\ q_i \neq q_f \text{ and } c \in T$$

(changing the state and writing the symbol c on the tape). One can also speak about the *halt* instruction of the counter automaton, assigned to the final state q_f.

A transition of the counter automaton consists in updating or checking the value of a counter, or writing a symbol on the tape, according to an instruction of one of the types described above and by changing the current state to another one. The computation starts in state q_0 with all counters equal to zero. The result of the computation is a word consisting of the symbols written on the tape when the automaton halts in state q_f.

The result we will use is that counter automata are computationally complete, and only two counters are needed.

2.2 P Systems with Symport/Antiport

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [22]; comprehensive information can be found in the P systems web page, [29].

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

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

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

and (y, out) *(symport rules)* and $(y, out; x, in)$ *(antiport rules)*, respectively, where $x, y \in O^+$.

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

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

2.3 Generating Languages

A computation is halting if starting from the initial configuration, the system reaches a configuration where no rule can be applied anymore.

Consider the output alphabet T. Throughout the computation we register when objects from T are sent to the environment. The result of a halting computation is the sequence of objects from T sent to the environment in the order in which it happened (if multiple objects from

T are ejected in the environment simultaneously, all permutations are considered). [1]

Example 1 *Consider the following P system*

$$\begin{aligned} \Pi &= (O = \{p, a, b\}, T = \{a, b\}, E = \{p, a, b\}, \mu = [_1 \ ]_1, w_1, R_1), \\ w_1 &= paab, \\ R_1 &= \{r_1 : (pa, out; pb, in), r_2 : (ab, out; a, in)\}. \end{aligned}$$

The environment contains an unbounded supply of objects p, a, b, so the state of the system is determined by objects in the skin membrane. The system starts with one copy of p, two copies of a and one copy of b in the skin membrane, simultaneously applying rules r_1 and r_2. Having one copy of each of the objects p, a, b, two computations are possible:

$$\boxed{pab}_1 \Rightarrow_1^{r_1} \boxed{pbb} \qquad or \qquad \boxed{pab}_1 \Rightarrow^{r_2} \boxed{pa}_1 \Rightarrow^{r_1} \boxed{pb}_1.$$

Consider the terminal objects sent out in either case. These computations correspond to generating a language

$$Perm(aab) \cdot a \cup Perm(aab) \cdot Perm(ab) \cdot a,$$

consisting of 9 words.

We denote the family of languages generated by a P system with symport/antiport rules with at most $m > 0$ membranes, symport rules of weight at most $s \geq 0$, and antiport rules of weight at most $t \geq 0$ by

$$LOP_m(sym_s, anti_t)$$

If $t = 0$, we may omit $anti_t$.

[1] Intuitively, sending the terminal objects out of the skin membrane is like writing them on the output tape.

3 Main Results

Theorem 1 $LOP_2(sym_1, anti_1) = RE$.

Proof. Without loss of generality we simulate a counter automaton with an output tape $M = (d, T, Q, q_0, q_f, P)$ which starts with empty counters. We also suppose that all instructions from P are labeled in a one-to-one manner with elements of $\{1, \ldots, n\} = I$, n is a label of the *halt* instruction and $I' = I \setminus \{n\}$. We denote by I_+, I_-, and $I_{=0}$ the set of labels for the "increment" -, "decrement" -, and "test for zero" -instructions, respectively ("increment"-instructions include also instructions operating on the tape). We use the next notation: $C = \{c_k\}, k \in D$.

We construct the P system Π_1 as follows:

$$
\begin{aligned}
\Pi_1 &= (O, T, E, [_1\, [_2\ \ ]_2\,]_1, w_1, w_2, R_1, R_2),\\
O &= E \cup \{X, Y_1, Y_2, J_1, J_2, J_3\} \cup \{b_j, d_j \mid j \in I\},\\
E &= T \cup Q \cup C \cup \{a_j, e_j \mid j \in I\} \cup \{a'_j \mid j \in I' \setminus I_-\} \cup \{J_0, F, Y_3\},\\
w_1 &= q_0 J_1 J_2 J_3,\\
w_2 &= X J_3 Y_1 Y_2 \prod_{j \in I} b_j \prod_{j \in I} d_j,\\
R_i &= R_{i,s} \cup R_{i,r} \cup R_{i,f}, \quad i = 1, 2.
\end{aligned}
$$

We code the counter automaton as follows:

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

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

1. START: preparation of the system for the computation.
2. RUN: simulation of instructions of the counter automaton.

3. END: terminating the computation.

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

1. START.

We use here the following idea: in our system we have a symbol X which moves from region 2 to region 1 and back in an infinite loop. This loop may be stopped only if all stages completed correctly.

$$\begin{aligned} R_{1,s} &= \emptyset, \\ R_{2,s} &= \{\mathtt{2s1} : (X, out), \mathtt{2s2} : (X, in)\}. \end{aligned}$$

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

2. RUN.

$$\begin{aligned} R_{1,r} &= \{\mathtt{1r1} : (q_i, out; a_j, in) \mid (j : q_i \to q_l, k\gamma) \text{ or} \\ &\quad (j : q_i \to q_l, write(c)) \in P,\ \gamma \in \{+, -, = 0\},\ k \in D,\ c \in T\} \\ &\cup \{\mathtt{1r2} : (q_f, out; a_n, in)\} \\ &\cup \{\mathtt{1r3} : (b_j, out; a'_j, in) \mid j \in I' \setminus I_-\} \\ &\cup \{\mathtt{1r4} : (b_j, out)) \mid j \in I_-\} \\ &\cup \{\mathtt{1r5} : (a_j, out; J_0, in),\ \mathtt{1r6} : (J_1, out; b_j, in) \mid j \in I\} \\ &\cup \{\mathtt{1r7} : (J_0, out; J_1, in)\} \end{aligned}$$

$$
\begin{aligned}
&\cup \quad \{\texttt{1r8} : (a'_j, out; c_k, in) \mid (j : q_i \to q_l, k+) \in P\} \\
&\cup \quad \{\texttt{1r9} : (a'_j, out; c, in) \mid (j : q_i \to q_l, write(c)) \in P\} \\
&\cup \quad \{\texttt{1r10} : (c, out) \mid c \in T\} \\
&\cup \quad \{\texttt{1r11} : (a'_j, out) \mid j \in I_{=0}\} \\
&\cup \quad \{\texttt{1r12} : (d_j, in) \mid j \in I_{=0} \cup I_+\} \\
&\cup \quad \{\texttt{1r13} : (c_k, out; d_j, in),\ \underline{\texttt{1r14}} : (J_3, out; d_j, in) \\
&\qquad \mid (j : q_i \to q_l, k-) \in P\} \\
&\cup \quad \{\texttt{1r15} : (d_j, out; e_j, in) \mid j \in I\} \\
&\cup \quad \{\texttt{1r16} : (e_j, out, q_l, in) \mid (j : q_i \to q_l, k\gamma) \text{ or} \\
&\qquad (j : q_i \to q_l, write(c)) \in P,\ \gamma \in \{+, -, = 0\},\ k \in D,\ c \in T\} \\
&\cup \quad \{\texttt{1r17} : (e_n, out; F, in), \texttt{1r18} : (b_n, out)\} \\
&\cup \quad \{\underline{\texttt{1r19}} : (J_3, out; J_1, in)\} \\
&\cup \quad \{\underline{\texttt{1r20}} : (\#, out),\ \underline{\texttt{1r21}} : (\#, in)\}. \\
R_{2,r} \ = \ &\{\texttt{2r1} : (b_j, out; a_j, in), \texttt{2r2} : (a_j, out; J_2, in), \\
&\underline{\texttt{2r3}} : (a_j, out; J_1, in) \mid j \in I\} \\
&\cup \quad \{\texttt{2r4} : (d_j, out; b_j, in) \mid j \in I\} \\
&\cup \quad \{\texttt{2r5} : (J_2, out; d_j, in) \mid j \in I_- \cup I_+\} \\
&\cup \quad \{\texttt{2r6} : (J_2, out; a'_j, in),\ \texttt{2r7} : (a'_j, out; d_j, in) \mid j \in I_{=0}\} \\
&\cup \quad \{\underline{\texttt{2r7}} : (a'_j, out; c_k, in) \mid j \in I_{=0}\} \\
&\cup \quad \{\underline{\texttt{2r8}} : (\#, out; J_0, in)\ \}.
\end{aligned}
$$

First of all, we mention that if during the phase RUN object J_3 comes to the environment (rules 1r14, 1r19), it remains there forever (**Scenario 0**). Then during the phase END the second symbol J_3 from region 2 will be moved to region 1 that leads to an infinite computation (by rule 1f4, see phase END).

Let us explain the synchronization of a_j coming to the environment and b_j leaving the environment: the first one brings J_0 into region 1 while the latter brings J_1 into the environment; then rule 1r7 returns J_0 and J_1 to their original locations.

If a_j comes to the environment without b_j leaving it, J_1 remains

in region 1 (or 2) and J_0 comes to region 1 (**Scenario 1**), so 2r8 is applied, causing an endless computation since 1r20 or 1r21 is always applicable.

If b_j leaves the environment without a_j coming there, J_0 remains in the environment and J_1 comes there (**Scenario 2**), so 1r19 is applied and symbol J_3 appears in the environment. Thus, the computation never halts, see scenario 0.

We also mention that applying rule 2r3 causes scenario 1. Therefore, in order for a computation to halt, no underlined rules should be applied.

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

" Increment" -instruction:

$$q_l\mathbf{a_j}a_tc_ke_jJ_0\boxed{\mathbf{q_i}J_1J_2J_3\boxed{b_jd_j\#}} \Rightarrow^{1r1} q_iq_la_ta'_jc_ke_jJ_0\boxed{\mathbf{a_j}J_1J_2J_3\boxed{\mathbf{b_j}d_j\#}}$$
$$\Rightarrow^{2r1} q_iq_l\mathbf{a'_j}a_tc_ke_jJ_0\boxed{\mathbf{b_j}J_1\mathbf{J_2}J_3\boxed{\mathbf{a_j}d_j\#}} \Rightarrow^{1r3,2r2}$$
$$q_iq_la_t\mathbf{b_jc_k}e_j\mathbf{J_0}\boxed{\mathbf{a_j}\mathbf{a'_j}\mathbf{J_1}J_3\boxed{J_2d_j\#}} \quad \text{(A)}$$
$$\Rightarrow^{1r5,1r6,1r8} q_iq_la_ja'_ja_te_j\mathbf{J_1}\boxed{\mathbf{b_j}c_k\mathbf{J_0}J_3\boxed{J_2\mathbf{d_j}\#}} \Rightarrow^{1r7,2r4}$$
$$q_iq_la_ja'_ja_t\mathbf{e_j}J_0\boxed{c_k\mathbf{d_j}J_1J_3\boxed{b_jJ_2\#}} \Rightarrow^{1r15}$$
$$q_i\mathbf{q_l}a_ja'_ja_t\mathbf{d_j}J_0\boxed{c_k\mathbf{e_j}J_1J_3\boxed{b_jJ_2\#}} \Rightarrow^{1r12,1r16}$$
$$q_ia_ja'_j\mathbf{a_t}e_jJ_0\boxed{\mathbf{q_l}c_k\mathbf{d_j}J_1J_3\boxed{b_j\mathbf{J_2}\#}} \Rightarrow^{1r1,2r5}$$
$$q_iq_la_ja'_je_jJ_0\boxed{\mathbf{a_t}c_kJ_1J_2J_3\boxed{b_jd_j\#}}$$

In that way, q_i is replaced by q_l and c_k is moved from the environment into region 1. In configuration (A) rule 1r9 may be applied instead of rule 1r8 and after that rule 1r10, so terminal symbol c will be sent to the environment. Thus we model instruction $j : (q_i \to q_l, write(c))$ of counter automaton M. Symbol d_j returns to region 2 in the first step of the simulation of the next instruction (the last step of the illustration).

Notice that symbols a_j, b_j, a'_j, d_j, e_j, J_2, J_1, J_0 have returned to their original positions.

" Decrement" -instruction:

$$q_l\mathbf{a_j}a_te_jJ_0\boxed{\mathbf{q_i}c_kJ_1J_2J_3\boxed{b_jd_j\#}} \Rightarrow^{\texttt{1r1}} q_iq_la_te_jJ_0\boxed{\mathbf{a_j}c_kJ_1J_2J_3\boxed{\mathbf{b_j}d_j\#}}$$
$$\Rightarrow^{\texttt{2r1}} q_iq_la_te_jJ_0\boxed{\mathbf{b_j}c_kJ_1\mathbf{J_2}J_3\boxed{\mathbf{a_j}d_j\#}} \Rightarrow^{\texttt{1r4,2r2}}$$
$$q_iq_la_t\mathbf{b_j}e_j\mathbf{J_0}\boxed{\mathbf{a_j}c_k\mathbf{J_1}J_3\boxed{d_jJ_2\#}} \Rightarrow^{\texttt{1r5,1r6}} q_iq_la_ja_te_j\mathbf{J_1}\boxed{\mathbf{b_j}c_k\mathbf{J_0}J_3\boxed{\mathbf{d_j}J_2\#}}$$
$$\Rightarrow^{\texttt{1r7,2r4}} q_iq_la_ja_t\mathbf{e_j}J_0\boxed{\mathbf{d_j}c_kJ_1J_3\boxed{b_jJ_2\#}} \Rightarrow^{\texttt{1r15}}$$
$$q_i\mathbf{q_l}a_ja_t\mathbf{d_j}J_0\boxed{\mathbf{c_k}\mathbf{e_j}J_1J_3\boxed{b_jJ_2\#}} \quad \text{(B)}$$
$$\Rightarrow^{\texttt{1r13,1r16}} q_ia_j\mathbf{a_t}c_ke_jJ_0\boxed{\mathbf{q_l}\mathbf{d_j}J_1J_3\boxed{b_j\mathbf{J_2}\#}} \Rightarrow^{\texttt{1r1,2r5}}$$
$$q_iq_la_jc_ke_jJ_0\boxed{\mathbf{a_t}J_1J_2J_3\boxed{b_jd_j\#}}$$

In the way described above, q_i is replaced by q_l and c_k is removed from region 1 to the environment. If symbol c_k is absent in region 1 in configuration (B) it enforces the applying of rule 1r14 that leads to an infinite computation. Symbol d_j returns to region 2 in the first step of the simulation of the next instruction (the last step of the illustration). Notice that symbols a_j, b_j, d_j, e_j, J_2, J_1, J_0 have returned to their original positions.

" Test for zero" -instruction:

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

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

$$q_la_t\mathbf{a_j}a'_je_jJ_0\boxed{\mathbf{q_i}J_1J_2J_3\boxed{b_jd_j\#}} \Rightarrow^{\texttt{1r1}} q_iq_la_ta'_je_jJ_0\boxed{\mathbf{a_j}J_1J_2J_3\boxed{\mathbf{b_j}d_j\#}}$$
$$\Rightarrow^{\texttt{2r1}} q_iq_la_t\mathbf{a'_j}e_jJ_0\boxed{\mathbf{b_j}J_1\mathbf{J_2}J_3\boxed{\mathbf{a_j}d_j\#}} \Rightarrow^{\texttt{1r3,2r2}}$$

$$q_i q_l a_t \mathbf{b_j} e_j \mathbf{J_0} \boxed{\mathbf{a_j} \mathbf{a'_j} \mathbf{J_1} J_3 \boxed{\mathbf{J_2} d_j \#}} \Rightarrow^{\texttt{1r5,1r6,2r6}}$$

$$q_i q_l a_j a_t e_j \mathbf{J_1} \boxed{\mathbf{b_j} \mathbf{J_0} J_2 J_3 \boxed{a'_j \mathbf{d_j} \#}} \qquad \text{(C)}$$

$$\Rightarrow^{\texttt{1r7,2r4}} q_i q_l a_j a_t \mathbf{e_j} J_0 \boxed{\mathbf{d_j} J_1 J_2 J_3 \boxed{a'_j b_j \#}} \Rightarrow^{\texttt{1r15}}$$

$$q_i \mathbf{q_l} a_j a_t \mathbf{d_j} J_0 \boxed{\mathbf{e_j} J_1 J_2 J_3 \boxed{a'_j b_j \#}} \Rightarrow^{\texttt{1r12,1r16}}$$

$$q_i a_j \mathbf{a_t} e_j J_0 \boxed{\mathbf{q_l} \mathbf{d_j} J_1 J_2 J_3 \boxed{\mathbf{a'_j} b_j \#}} \Rightarrow^{\texttt{1r1,2r7}} q_i q_l a_j e_j J_0 \boxed{\mathbf{a_t} \mathbf{a'_j} J_1 J_2 J_3 \boxed{b_j d_j \#}}$$

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

(ii) *There is some c_k in region 1:*
Consider configuration (C) with object c_k in region 1:

$$q_i q_l a_j a_t e_j \mathbf{J_1} \boxed{\mathbf{b_j} \mathbf{c_k} \mathbf{J_0} J_2 J_3 \boxed{\mathbf{a'_j} \mathbf{d_j} \#}} \Rightarrow^{\texttt{1r7,2r4,2r7}}$$

$$q_i q_l a_j a_t \mathbf{e_j} J_0 \boxed{\mathbf{a'_j} \mathbf{d_j} J_1 J_2 J_3 \boxed{b_j c_k \#}} \Rightarrow^{\texttt{1r11,1r15}}$$

$$q_i \mathbf{q_l} a_j a'_j a_t \mathbf{d_j} J_0 \boxed{\mathbf{e_j} J_1 J_2 J_3 \boxed{b_j c_k \#}} \Rightarrow^{\texttt{1r12,1r16}}$$

$$q_i a_j a'_j \mathbf{a_t} \mathbf{e_j} J_0 \boxed{\mathbf{q_l} \mathbf{d_j} J_1 J_2 J_3 \boxed{b_j c_k \#}}$$

Now rule `1r15` again will be applied and two symbols a_t, a_s appear in several steps in region 1 that leads to an infinite computation (see scenario 1).

Let us consider the symbols from region 2 visiting the environment and going back: $2 \to 1 \to 0 \to 1 \to 2$ (b_j, d_j for all instructions) and the symbols from the environment visiting region 2 and going back: $0 \to 1 \to 2 \to 1 \to 0$. The latter ones are: a_j for all instructions, and also $\{a'_j \mid j \in I_{=0}\}$.

Then we have to argue that if they **Return** to their "home region" ($2 \to 1 \to 2$ or $0 \to 1 \to 0$) or **Repeat** their visit to the "opposite region" before returning "home" ($2 \to 1 \to 0 \to 1 \to 0$ or $0 \to 1 \to 2 \to 1 \to 2$), an infinite computation is unavoidable, or such case is not possible.

$a_j, j \in I$. Return: see scenario 1; repeat: impossible without b_j.

$a'_j, j \in I_{=0}$. Return: as d_j cannot come back to region 2 (is not possible to apply rule `2r6`), it again goes to the environment (see d_j, repeat); repeat: impossible without J_2.

$b_j, j \in I$. Return: if a_j comes to the environment, scenario 1 takes place. If a_j returns to region 2, rule `2r3` is applied; repeat: symbol J_1 will be moved to the environment, where it changes with J_3 (rule `1r19`), so it leads to an infinite computation (scenario 0).

$d_j, j \in I$. Return: e_j stays in the environment, the simulation stops and the computation never ends due to `2s1`, `2s2`; repeat: two symbols a_t, a_s appear in several steps in region 1 that leads to an infinite computation (see scenario 1).

3. END.

$$\begin{aligned} R_{1,f} &= \{\texttt{1f1} : (Y_1, out; Y_3, in),\ \texttt{1f2} : (Y_2, out)\} \\ &\cup \{\texttt{1f3} : (X, out; Y_2, in),\ \texttt{1f4} : (J_3, out, J_3, in)\}. \\ R_{2,f} &= \{\texttt{2f1} : (Y_1, out; F, in),\ \texttt{2f2} : (Y_2, out; F, in),\ \texttt{2f3} : (F, out)\} \\ &\cup \{\texttt{2f4} : (J_3, out; Y_3, in)\}. \end{aligned}$$

Once the counter automaton reaches the final state, q_f is in region 1 and it exchanges with object a_n (rule `1r2`), object F will be moved to region 1 in several steps. It takes Y_1, Y_2 and J_3 to region 1, in either order. The duty of Y_2 is to bring X to the environment (the object X can oscillate for indefinite time, but we are interested in halting computations). The duty of Y_1 is to bring J_3 from region 2 to region 1. If during the previous steps of simulation of counter automaton M object J_3 from region 1 was moved to the environment (by rules `1r14` or `1r19`), rule `1f4` will be applied, leading to an infinite computation.

If on the previous steps of simulation of counter automaton M object J_1 was moved to region 2 (by rules 2r3), rule 2r8 will be applied in several steps, and the computation never halts.

Thus, at the end of a terminating computation terminal word $w \in T$ will be sent to the environment. □

Theorem 2 $LOP_2(sym_2) = RE$.

Proof. Our proof is based on the construction introduced in Theorem 2 from [2]. As in the proof of Theorem 1 we simulate a counter automaton $M = (d, T, Q, q_0, q_f, P)$ which starts with empty counters. Again we suppose that all instructions from P are labelled in a one-to-one manner with elements of $\{1, \ldots, n\} = I$ and that I is the disjoint union of $\{n\}$ as well as I_+, I_-, and $I_{=0}$, where by I_+, I_-, and $I_{=0}$ we denote the set of labels for the "increment", "decrement", and "test for zero" instructions, respectively (recall that "increment"-instructions include also instructions operating on the tape). Moreover, we define $I' = I \setminus \{n\}$ and $Q' = Q \setminus \{q_0\}$. We also suppose that there is only one instruction with initial state q_0 (labeled with number 1) and only one instruction with the final state q_f (labeled with number n).

We construct the P system Π_2 as follows:

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

We code the counter automaton as follows: The environment will hold the current state of the automaton, represented by a symbol $q_i \in$

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

Again as in Theorem 1 we split our proof into several parts that depend on the logical separation of the behavior of the system. We will present the rules and the initial symbols for each part, but we remark that the system that we present is the union of all these parts.

The rules R_i are given by three phases:

1. START: preparation of the system for the computation.
2. RUN: simulation of instructions of the counter automaton.
3. END: terminating the computation.

1. START.

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

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

2. RUN.

$$\begin{aligned} R_{1,r} &= \{\texttt{1r1} : (q_i a_j, in) \mid (j : q_i \to q_l, k\gamma) \text{ or} \\ &\quad (j : q_i \to q_l, write(c)) \in P,\ \gamma \in \{+, -, = 0\},\ k \in D,\ c \in T\} \\ &\cup \{\texttt{1r2} : (b_j g_j, out) \mid j \in I_+ \cup I_{=0}\} \\ &\cup \{\texttt{1r3} : (c_k b_j, in) \mid (j : q_i \to q_l, k+) \in P, k \in D\} \\ &\cup \{\texttt{1r4} : (g_j c_k, out) \mid (j : q_i \to q_l, k-) \in P, k \in D\} \end{aligned}$$

$$
\begin{aligned}
&\cup \ \{\texttt{1r5} : (a'_j g_j, in) \mid j \in I'\} \\
&\cup \ \{\texttt{1r6} : (\#_1, out), \texttt{1r7} : (\#_1, in)\} \\
&\cup \ \{\texttt{1r8} : (d_j b_j, in) \mid j \in I_{=0}\} \\
&\cup \ \{\texttt{1r9} : (d_j c_k, out) \mid (j : q_i \to q_l, k = 0) \in P, k \in D\} \\
&\cup \ \{\texttt{1r10} : (a'_j q_l, out) \mid (j : q_i \to q_l, k\gamma) \text{ or} \\
&\quad (j : q_i \to q_l, write(c)) \in P,\ \gamma \in \{+, -\},\ k \in D,\ c \in T\} \\
&\cup \ \{\texttt{1r11} : (a'_j g'_j, out), \texttt{1r12} : (d'_j g'_j, in), \\
&\quad \texttt{1r13} : (d'_j, out) \mid j \in I_{=0}\} \\
&\cup \ \{\texttt{1r14} : (d_j q_l, out) \mid (j : q_i \to q_l, k = 0) \in P, k \in D\}, \\
&\cup \ \{\texttt{1r15} : (c b_j, in) \mid (j : q_i \to q_l, write(c)) \in P\} \\
&\cup \ \{\texttt{1r16} : (c, out) \mid c \in T\}, \\
R_{2,r} \ = \ &\{\texttt{2r1} : (a_j b_j, in) \mid j \in I'\} \\
&\cup \ \{\texttt{2r2} : (q_i, in) \mid q_i \in Q\} \\
&\cup \ \{\texttt{2r3} : (b_j g_j, out) \mid j \in I'\} \\
&\cup \ \{\texttt{2r4} : (a'_j \$, in) \mid j \in I\} \\
&\cup \ \{\texttt{2r5} : (\#_1 \$, out)\} \\
&\cup \ \{\texttt{2r6} : (a'_j g_j, in) \mid j \in I'\} \\
&\cup \ \{\texttt{2r7} : (a'_j q_l, out) \mid (j : q_i \to q_l, k\gamma) \text{ or} \\
&\quad (j : q_i \to q_l, write(c)) \in P,\ \gamma \in \{+, -\},\ k \in D,\ c \in T\} \\
&\cup \ \{\texttt{2r8} : (a'_j g'_j, out) \mid j \in I_{=0}\} \\
&\cup \ \{\texttt{2r9} : (d'_j g'_j, in) \mid j \in I_{=0}\} \\
&\cup \ \{\texttt{2r10} : (d'_j q_l, out) \mid (j : q_i \to q_l, k = 0) \in P, k \in D\}.
\end{aligned}
$$

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

"Increment" instruction:

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

$$a'_j c_k \boxed{\$\boxed{q_i a_j \mathbf{b_j g_j} q_l \#_1}} \Rightarrow^{\texttt{2r3}} a'_j c_k \boxed{\mathbf{b_j g_j}\$\boxed{q_i a_j q_l \#_1}} \Rightarrow^{\texttt{1r2}}$$
$$\mathbf{c_k b_j g_j a'_j}\boxed{\$\boxed{q_i a_j q_l \#_1}} \quad \text{(A)}$$
$$\Rightarrow^{\texttt{1r3,1r5}} \boxed{c_k b_j g_j a'_j \$\boxed{q_i a_j q_l \#_1}}$$

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

a) Applying rule `1r2`:

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

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

b) Applying rule `2r6`:

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

q_i is replaced by q_l and c_k is moved into region 1. In configuration (A) rule `1r15` may be applied instead of rule `1r3` and after that rule `1r16`, so terminal symbol c will be sent to the environment. Thus we model instruction $j : (q_i \to q_l, write(c))$ of counter automaton M.

"Decrement" instruction:

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

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

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

c) Applying rule `1r4`:

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

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

d) Applying rule `2r6`:

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

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

"Test for zero" instruction:

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

Case e):

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

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

$$d'_j\boxed{d_jb_j\mathbf{g_j}\mathbf{a'_j}\$\boxed{q_ia_jg'_jq_l\#_1}} \Rightarrow^{\texttt{2r6}} d'_j\boxed{d_jb_j\$\boxed{q_ia_jg_j\mathbf{a'_j}\mathbf{g'_j}q_l\#_1}} \Rightarrow^{\texttt{2r8}}$$
$$d'_j\boxed{\mathbf{a'_j}\mathbf{g'_j}d_jb_j\$\boxed{q_ia_jg_jq_l\#_1}} \Rightarrow^{\texttt{1r11}} a'_j\mathbf{g'_j}\mathbf{d'_j}\boxed{d_jb_j\$\boxed{q_ia_jg_jq_l\#_1}} \Rightarrow^{\texttt{1r12}}$$
$$a'_j\boxed{d_jb_j\$\mathbf{g'_j}\mathbf{d'_j}\boxed{q_ia_jg_jq_l\#_1}} \Rightarrow^{\texttt{2r9}} a'_j\boxed{d_jb_j\$\boxed{q_ia_jg_jg'_j\mathbf{d'_j}\mathbf{q_l}\#_1}} \Rightarrow^{\texttt{2r10}}$$
$$a'_j\boxed{\mathbf{d'_j}\mathbf{q_l}\mathbf{d_j}b_j\$\boxed{q_ia_jg_jg'_j\#_1}} \Rightarrow^{\texttt{1r13,1r14}} a'_jd_jd'_jq_l\boxed{b_j\$\boxed{q_ia_jg_jg'_j\#_1}}$$

Thus, q_i is replaced by q_l.

Case f):

$$a'_jd_jd'_j\mathbf{q_i}\mathbf{a_j}\boxed{c_kb_j\$\boxed{g_jg'_jq_l\#_1}} \Rightarrow^{\texttt{1r1}} a'_jd_jd'_j\boxed{c_k\mathbf{q_i}\mathbf{a_j}\mathbf{b_j}\$\boxed{g_jg'_jq_l\#_1}} \Rightarrow^{\texttt{2r1,2r2}}$$
$$a'_jd_jd'_j\boxed{c_k\$\boxed{q_ia_j\mathbf{b_j}\mathbf{g_j}g'_jq_l\#_1}} \Rightarrow^{\texttt{2r3}} a'_jd_jd'_j\boxed{c_k\mathbf{b_j}\mathbf{g_j}\$\boxed{q_ia_jg'_jq_l\#_1}} \Rightarrow^{\texttt{1r2}}$$
$$d'_j\mathbf{b_j}\mathbf{d_j}\mathbf{g_j}\mathbf{a'_j}\boxed{c_k\$\boxed{q_ia_jg'_jq_l\#_1}} \Rightarrow^{\texttt{1r8,1r5}} d'_j\boxed{\mathbf{c_k}\mathbf{d_j}\mathbf{g_j}\mathbf{a'_j}b_j\$\boxed{q_ia_jg'_jq_l\#_1}}$$
$$\Rightarrow^{\texttt{1r9,2r6}}$$
$$c_kd_jd'_j\boxed{b_j\$\boxed{q_ia_jg_j\mathbf{a'_j}\mathbf{g'_j}q_l\#_1}} \Rightarrow^{\texttt{2r8}} c_kd_jd'_j\boxed{\mathbf{a'_j}\mathbf{g'_j}b_j\$\boxed{q_ia_jg_jq_l\#_1}} \Rightarrow^{\texttt{1r11}}$$
$$a'_jc_kd_j\mathbf{g'_j}\mathbf{d'_j}\boxed{b_j\$\boxed{q_ia_jg_jq_l\#_1}} \Rightarrow^{\texttt{1r12}} a'_jc_kd_j\boxed{b_j\$\mathbf{g'_j}\mathbf{d'_j}\boxed{q_ia_jg_jq_l\#_1}} \Rightarrow^{\texttt{2r9}}$$
$$a'_jc_kd_j\boxed{b_j\$\boxed{q_ia_jg_jg'_j\mathbf{d'_j}\mathbf{q_l}\#_1}} \Rightarrow^{\texttt{2r10}} a'_jc_kd_j\boxed{\mathbf{d'_j}q_lb_j\$\boxed{q_ia_jg_jg'_j\#_1}} \Rightarrow^{\texttt{1r13}}$$
$$a'_jc_kd_jd'_j\boxed{q_lb_j\$\boxed{q_ia_jg_jg'_j\#_1}}$$

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

3. END.

$$\begin{aligned} R_{1,f} &= \{\texttt{1f1} : (q_f a_n, in)\} \\ R_{2,f} &= \{\texttt{2f1} : (\#_2 g_n, in), \texttt{2f2} : (f a_n, in), \texttt{2f3} : (f g_n, out)\}. \end{aligned}$$

Object $\#_2$ is moved to region 2, so we stop without continuing the loop.

Thus, at the end of a terminating computation terminal word $w \in T$ will be sent to the environment. □

4 Discussion

Consider P systems with two membranes and symport of weight at most two or symport/antiport rules of weight one. It has been shown in [5] for both classes that they generate all recursively enumerable sets of positive integers and some finite sets of non-negative integers containing zero. In the present work we look at generating languages by the same systems, taking sequences of terminal objects sent into the environment as the output of the system. We have shown that both classes are computationally complete.

The case of one membrane remains to be investigated. Unlike generating numbers, it is possible to generate infinite sets:

Example 2

$$\begin{aligned} \Pi_3 &= (O = \{a, b\}, T = \{a\}, E = O, \mu = [_1 \ ]_1, w_1 = b, R_1), \\ R_1 &= \{(b, out; a, in), (a, out; b, in), (a, out)\}; \end{aligned}$$

$$\begin{aligned} \Pi_4 &= (O = \{a, b\}, T = \{a\}, E = O, \mu = [_1 \ ]_1, w_1 = ba, R_1), \\ R_1 &= \{(ba, out), (ba, in), (b, in)\}; \end{aligned}$$

$$L(\Pi_1) = L(\Pi_2) = a^+$$

Let us return to the way the behaviour of the system is defined. Notice that $T \subseteq E$ holds for both proofs (the output symbols are available in the environment in unbounded quantity). Therefore, when a terminal symbol is ejected into the environment, it is not important whether it is allowed to re-enter the system (it has been registered and "released") or not (it remains on the "tape"). In the next paragraph, however, we assume the first case (the other one is more restricted).

We come back to systems with multiple membranes. Clearly, such computational power is only possible when we register just the terminal symbols. What happens if we require

$$T = \{a \in O \mid |u|_a > 0,\ (u, out) \in R_1 \text{ or } (u, out; v, in) \in R_1\},$$

i.e., all symbols that may be sent into the environment constitute the terminal alphabet? It turns out that the total number of objects present inside the system cannot exceed the initial value plus the size of the output word (multiplied by a constant if we drop the restriction on the weight of symport/antiport rules). Following the argument that the total number of configurations with at most N objects is a polynomial with respect to N (depending on the size of the alphabet and the number of membranes), it is not difficult to see that the condition above bounds the power of such systems by $LOGSPACE$ [15] (i.e., that of Turing machines with a working tape of size $O(\log n)$, where n is the size of the output word).

Finally, it is possible to define P systems accepting words by registering the terminal symbols that enter the skin membrane from the environment. In principal, similar computational completeness results may be obtained. We did not include such results in this article because it would be non-deterministic acceptance and because there are multiple ways one could define the behaviour of such P systems.

The authors thank Dr. Rudolf Freund for the useful discussions of the definitions.

References

[1] A. Alhazov, R. Freund, Yu. Rogozhin. *Computational Power of Symport/Antiport: History, Advances, and Open Problems.* International Workshop in Membrane Computing (WMC6) (R. Freund, G. Lojka, M. Oswald, Gh. Păun, Eds.) Vienna Institute of Technology, Vienna (2005) 44–78 and also Membrane Computing, International Workshop, WMC 2005, Vienna, 2005, Revised Selected and Invited Papers (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), *Lecture Notes in Computer Science*, Springer **3850** (2006) 1–30.

[2] A. Alhazov, R. Freund, Yu. Rogozhin. *Some Optimal Results on Communicative P Systems with Minimal Cooperation.* Cellular Computing (Complexity Aspects), ESF PESC Exploratory Workshop (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, Eds.), Fénix Editora, Sevilla, (2005) 23–36.

[3] A. Alhazov, M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan. *Communicative P Systems with Minimal Cooperation.* Membrane Computing, International Workshop, WMC 2004, Milan, 2004, Revised Selected and Invited Papers (G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, Eds.) *Lecture Notes in Computer Science*, Springer **3365** (2005) 161–177.

[4] A. Alhazov, Yu. Rogozhin. *Minimal Cooperation in Symport/ Antiport P Systems with One Membrane.* Third Brainstorming Week on Membrane Computing (M.A. Gutiérrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, Eds.) RGNC TR *01/2005*, University of Seville, Fénix Editora, Sevilla (2005) 29–34.

[5] A. Alhazov, Yu. Rogozhin. *Towards a Characterization of P Systems with Minimal Symport/Antiport and Two Membranes.* In: Pre-proc. of the 7th Workshop on Membrane Computing, WMC7, 17 – 21 July, 2006, Lorentz Center, Leiden (2006) 102–117. Revised Selected and Invited Papers in: *Lecture Notes in Computer Science*, Springer **4361** (2006) 135–153.

[6] A. Alhazov, Yu. Rogozhin, S. Verlan. *Symport/Antiport Tissue P Systems with Minimal Cooperation.* Cellular Computing

(Complexity Aspects), ESF PESC Exploratory Workshop (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, Eds.), Fénix Editora, Sevilla (2005) 37–52.

[7] F. Bernardini, M. Gheorghe. *On the Power of Minimal Symport/Antiport.* Workshop on Membrane Computing, WMC 2003 (A. Alhazov, C. Martín-Vide, Gh. Păun, Eds.), Tarragona, 2003, TR *28/03*, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona (2003) 72–83.

[8] F. Bernardini, A. Păun. *Universality of Minimal Symport/ Antiport: Five Membranes Suffice.* Membrane Computing, International Workshop, WMC 2003, Tarragona, Revised Papers (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.), *Lecture Notes in Computer Science*, Springer **2933** (2004) 43–45.

[9] R. Freund, M. Oswald. *GP Systems with Forbidding Context.* Fundamenta Informaticae, IOS Press **49**, 1–3 (2002) 81–102.

[10] R. Freund, M. Oswald. *P Systems with Activated/Prohibited Membrane Channels.* Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Argeş, 2002. Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science*, Springer **2597** (2003) 261–268.

[11] R. Freund, A. Păun. *Membrane Systems with Symport/Antiport: Universality Results.* Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Argeş, 2002. Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science*, Springer **2597** (2003) 270–287.

[12] P. Frisco. *About P Systems with Symport/Antiport.* Second Brainstorming Week on Membrane Computing (Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, Eds), TR *01/2004*, Research Group on Natural Computing, University of Seville (2004) 224–236.

[13] P. Frisco, H.J. Hoogeboom. *P Systems with Symport/Antiport Simulating Counter Automata.* Acta Informatica, Springer **41**, 2–3 (2004) 145–170.

[14] P. Frisco, H.J. Hoogeboom. *Simulating Counter Automata by P Systems with Symport/Antiport.* Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Argeş, 2002. Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, Eds.), *Lecture Notes in Computer Science*, Springer **2597** (2003) 288–301.

[15] J.Gruska. *Foundations of Computing.* International Thomson Computer Press (1997).

[16] L. Kari, C. Martín-Vide, A. Păun. *On the Universality of P Systems with Minimal Symport/Antiport Rules.* Aspects of Molecular Computing - Essays dedicated to Tom Head on the occasion of his 70th birthday, *Lecture Notes in Computer Science*, Springer **2950** (2004) 254–265.

[17] M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan. *About P Systems with Minimal Symport/Antiport Rules and Four Membranes.* Fifth Workshop on Membrane Computing (WMC5), (G. Mauri, Gh. Păun, C. Zandron, Eds.), Universitá di Milano-Bicocca, Milan (2004) 283–294.

[18] C. Martín-Vide, A. Păun, Gh. Păun. *On the Power of P Systems with Symport Rules,* Journal of Universal Computer Science, **8**, 2 (2002) 317–331.

[19] M.L. Minsky. *Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey (1967).

[20] A. Păun, Gh. Păun. *The Power of Communication: P Systems with Symport/Antiport.* New Generation Computing, Japan, **20** (2002) 295–305.

[21] Gh. Păun. *Computing with Membranes.* Journal of Computer and Systems Science, **61** (2000) 108–143.

[22] Gh. Păun. *Membrane Computing. An Introduction.* Springer-Verlag (2002).

[23] Gh. Păun. *Further Twenty Six Open Problems in Membrane Computing (2005).* Third Brainstorming Week on Membrane Computing (M.A. Gutiérrez-Naranjo, A. Riscos-Núñez, F.J. Romero-

Campero, D. Sburlan, Eds.) RGNC TR *01/2005*, University of Seville, Fénix Editora, Sevilla (2005) 249–262.

[24] Gh.Păun. *2006 Research Topics in Membrane Computing.* Fourth Brainstorming Week on Membrane Computing, vol. *1* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, Eds.), Fénix Edit., Sevilla (2006), 235–251.

[25] G. Rozenberg, A. Salomaa (Eds.). *Handbook of Formal Languages* (3 volumes). Springer-Verlag, Berlin (1997).

[26] Gy. Vaszil. *On the Size of P Systems with Minimal Symport/ Antiport.* Fifth Workshop on Membrane Computing (WMC5) (G. Mauri, Gh. Păun, C. Zandron, Eds.), Universitá di Milano-Bicocca, Milan (2004) 422–431.

[27] S. Verlan. *Optimal Results on Tissue P Systems with Minimal Symport/ Antiport.* Presented at EMCC meeting, Lorentz Center, Leiden (2004).

[28] S. Verlan. *Tissue P Systems with Minimal Symport/Antiport.* Developments in Language Theory, DLT 2004 (C.S. Calude, E. Calude, M.J. Dinneen, Eds), *Lecture Notes in Computer Science*, Springer **3340**, (2004) 418–430.

[29] P Systems Webpage, `http://psystems.disco.unimib.it`

Artiom Alhazov, Yurii Rogozhin, Received December 1, 2006

Dr. Artiom Alhazov
Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
5 Academiei str., Chişinău, MD-2028, Moldova
email: *artiom@math.md*
Research Group on Mathematical Linguistics
Rovira i Virgili University, Tarragona, Spain
email: *artiome.alhazov@estudiants.urv.cat*

Dr.hab. Yurii Rogozhin
Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
5 Academiei str., Chişinău, MD-2028, Moldova
email: *rogozhin@math.md*