Skin Output in P Systems with Minimal Symport/Antiport and Two Membranes*

Artiom Alhazov[1,2] and Yurii Rogozhin[1,3]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD-2028 Moldova
{artiom,rogozhin}@math.md

[2] Åbo Akademi University
Department of Information Technologies
Turku Center for Computer Science, FIN-20520 Turku, Finland
aalhazov@abo.fi

[3] Rovira i Virgili University,
Research Group on Mathematical Linguistics,
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain

Summary. It is known that symport/antiport P systems with two membranes and minimal cooperation can generate any recursively enumerable sets of natural numbers using exactly one superfluous object in the output membrane, where the output membrane is an elementary membrane. In this paper we consider symport/antiport P systems where the output membrane is the skin membrane. In this case we prove an unexpected characterization: symport/antiport P systems with two membranes and minimal cooperation generate exactly the recursively enumerable sets of natural numbers. The question about power of purely symport P systems with two membranes and minimal cooperation where the output membrane is the skin membrane is still open.

1 Introduction

P systems with *symport/antiport* rules, i.e., P systems with *pure communication rules assigned to membranes*, first were introduced in [21]; symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions. These operations are very powerful, i.e., P systems with symport/antiport rules have universal computational power with only one membrane, e.g., see [12], [15], [13].

* 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. The first author gratefully acknowledges the support by Academy of Finland, project 203667 and the third author gratefully acknowledges the support of European Commission, project MolCIP, MIF1-CT-2006-021666.

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].

For instance, in [3] one obtains the exact characterization of $\mathbb{N}RE$ for symport/antiport P systems with three membranes and minimal cooperation and for corresponding purely symport P systems.

In [5] one shows that if some P system with two membranes and with minimal cooperation, i.e., a P system with symport/antiport rules of weight one or a P system with symport rules of weight two, generates a set of numbers *containing zero*, then this set is **finite**. After that one proves that P systems with symport/antiport rules of weight one can generate any *recursively enumerable* set of natural numbers without zero (i.e., they are computationally complete with just **one superfluous object** remaining in the output membrane at the end of a halting computation). The same result is true also for purely symport P systems of weight two. Therefore, one superfluous object is both necessary and sufficient in case of two membranes.

The question about precise characterization of computational power of symport/antiport P systems (purely symport P systems) with two membranes and minimal cooperation is still open.

Interpreting the result of the computation as the sequence of terminal symbols sent to the environment, one shows that P systems with two membranes and symport rules of weight two or symport/antiport rules of weight one generate all recursively enumerable languages [6].

In this paper we show that P systems with minimal symport/antiport with two membranes characterize $\mathbb{N}RE$ when we consider the **output in the skin membrane** rather than the elementary membrane.

2 Basic Notations and Definitions

For the basic elements of formal language theory needed in the following, we refer to [26]. We just list a few notions and notations: $\mathbb{N}$ denotes the set of natural numbers (i.e., of non-negative integers). V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; by $\mathbb{N}RE$, $\mathbb{N}REG$, and $\mathbb{N}FIN$ we denote the family of recursively enumerable sets, regular sets, and finite sets of natural numbers, respectively. For $k \geq 1$, by $\mathbb{N}_kRE$ we denote the family of recursively enumerable sets of natural numbers excluding the initial segment 0 to $k-1$. Particularly, $\mathbb{N}_1RE = \{N \in \mathbb{N}RE \mid 0 \notin N\}$. The families of recursively enumerable sets of vectors of natural numbers are denoted by $PsRE$.

2.1 Counter Automata

A non-deterministic *counter automaton* (see [11], [1]) is a construct

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

- d is the number of counters, and we denote $D = \{1, ..., d\}$;
- Q is a finite set of states, and without loss of generality, we use the notation $Q = \{q_i \mid 0 \leq i \leq f\}$ and $F = \{0, 1, ..., f\}$,
- $q_0 \in Q$ is the initial state,
- $q_f \in Q$ is the final state, and
- P is a finite set of instructions of the following form:

1. $(q_i \to q_l, k+)$, with $i, l \in F,\ i \neq f,\ k \in D$ ("increment" -instruction). This instruction increments counter k by one and changes the state of the system from q_i to q_l.
2. $(q_i \to q_l, k-)$, with $i, l \in F,\ i \neq f,\ k \in D$ ("decrement" -instruction). If the value of counter 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 counter k is zero) the computation is blocked in state q_i.
3. $(q_i \to q_l, k = 0)$, with $i, l \in F,\ i \neq f,\ k \in D$ ("test for zero" -instruction). If the value of counter k is zero, then this instruction changes the state of the system from q_i to q_l. Otherwise (the value stored in counter k is greater than zero) the computation is blocked in state q_i.
4. *halt*. This instruction stops the computation of the counter automaton, and it can only be assigned to the final state q_f.

A transition of the counter automaton consists in updating/checking the value of a counter according to an instruction of one of the types described above and by changing the current state to another one. The computation starts in state q_0 with all counters being equal to zero. The result of the computation of a counter automaton is the value of the first k counters when the automaton halts in state $q_f \in Q$ (without loss of generality we may assume that in this case all other counters are empty). A counter automaton thus (by means of all computations) generates a set of k-vectors of natural numbers. If $k = 1$, then by $N(M)$ we denote the corresponding numeric set generated by M.

2.2 P Systems with Symport/Antiport Rules

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [23]; comprehensive information can be found in the P systems web page, [30].

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

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

1. O is a finite alphabet of symbols called *objects*;
2. μ is a *membrane structure* consisting of k 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 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 an infinite number of copies;
5. 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 $x, y \in O^+$;
6. i_0 is the label of a membrane of μ that identifies the corresponding output region.

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.

A computation is successful if starting from the initial configuration, the P system reaches a configuration where no rule can be applied anymore. The result of a successful computation is a natural number that is obtained by counting all objects present in region i_0. Given a P system Π, the set of natural numbers computed in this way by Π is denoted by $N(\Pi)$. If the multiplicity of each object is counted separately, then a vector of natural numbers is obtained, denoted by $Ps(\Pi)$, see [23].

By $\mathbb{N}OP_m(sym_s, anti_t)$ we denote the family of sets of natural numbers generated by P systems with symport/antiport rules with at most $m > 0$ membranes, symport rules of size at most $s \geq 0$, and antiport rules of size at most $t \geq 0$. In the papers on P systems, following [23], i_0 is assumed to be an elementary membrane. In this paper we will write $\mathbb{N}^{skin}OP_m(sym_s, anti_t)$ if i_0 is the skin membrane. Any unbounded parameter m, s, t is replaced by $*$. If $t = 0$, then we may omit $anti_t$.

3 Main result

Theorem 1. $\mathbb{N}^{skin}OP_2(sym_1, anti_1) = \mathbb{N}RE$.

Proof. We simulate a counter automaton $M = (d, Q, q_0, q_f, P)$. Recall that M starts with empty counters. We also suppose that all instructions from P are labeled in a one-to-one manner with elements of $\{1, \dots, n\} = I$, 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. We also use the following notation: $C = \{c_k\}, k \in D$ and $Q' = Q \setminus \{q_0\}$.

We construct the P system Π_1 as follows:

$$
\begin{aligned}
\Pi_1 &= (O, [_1\ [_2\ \ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 1),\\
O &= E \cup \{L, T_1, T_2, P_2, J_1, J_2, J_3\} \cup \{b_j \mid j \in I\} \cup \{d_j \mid j \in I'\},\\
E &= Q' \cup C \cup \{a_j \mid j \in I\} \cup \{a'_j, e_j \mid j \in I'\} \cup \{J_0, P_1\} \cup \{F_i \mid 0 \le i \le 9\},\\
w_1 &= q_0 L J_1 J_2 J_3,\\
w_2 &= T_1 T_2 P_2 \prod_{j \in I} b_j \prod_{j \in I'} d_j,\\
R_i &= R_{i,s} \cup R_{i,r} \cup R_{i,f}, \quad i = 1, 2.
\end{aligned}
$$

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 also 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 the following idea: in our system we have a symbol L which moves from region 1 to the environment and back in an infinite loop. This loop may be stopped only if all stages are completed correctly.

$$R_{1,s} = \{\mathtt{1s1} : (L, out), \mathtt{1s2} : (L, in)\}.$$
$$R_{2,s} = \emptyset.$$

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.

$$\begin{aligned} R_{1,r} = & \{\mathtt{1r1} : (q_i, out; a_j, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, =0\}\} \\ & \cup \{\mathtt{1r2} : (q_f, out; a_n, in)\} \\ & \cup \{\mathtt{1r3} : (b_j, out; a'_j, in) \mid j \in I'\} \\ & \cup \{\mathtt{1r4} : (a_j, out; J_0, in),\ \mathtt{1r5} : (J_1, out; b_j, in) \mid j \in I\} \\ & \cup \{\mathtt{1r6} : (J_0, out; J_1, in)\} \\ & \cup \{\mathtt{1r7} : (a'_j, out; c_k, in) \mid (j : q_i \to q_l, c_k+) \in P\} \\ & \cup \{\mathtt{1r8} : (a'_j, out) \mid j \in I_- \cup I_{=0}\} \\ & \cup \{\mathtt{1r9} : (d_j, in) \mid j \in I_+ \cup I_{=0}\} \\ & \cup \{\mathtt{1r10} : (c_k, out; d_j, in) \mid (j : q_i \to q_l, c_k-) \in P\} \\ & \cup \{\underline{\mathtt{1r11}} : (J_3, out; d_j, in) \mid \in I_-\} \\ & \cup \{\underline{\mathtt{1r12}} : (J_3, out; J_1, in)\} \\ & \cup \{\mathtt{1r13} : (d_j, out; e_j, in) \mid j \in I'\} \\ & \cup \{\mathtt{1r14} : (e_j, out, q_l, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, =0\}\} \\ & \cup \{\mathtt{1r15} : (b_n, out; F_0, in)\} \\ & \cup \{\underline{\mathtt{1r16}} : (\#, out), \underline{\mathtt{1r17}} : (\#, in)\}. \end{aligned}$$

$$\begin{aligned} R_{2,r} = & \{\mathtt{2r1} : (b_j, out; a_j, in), \mathtt{2r2} : (a_j, out; J_2, in) \mid j \in I\} \\ & \cup \{\underline{\mathtt{2r3}} : (a_j, out; J_1, in) \mid j \in I\} \\ & \cup \{\mathtt{2r4} : (d_j, out; a'_j, in) \mid j \in I'\} \\ & \cup \{\underline{\mathtt{2r5}} : (a'_j, out; c_k, in) \mid (j : q_i \to q_l, c_k = 0) \in P\} \\ & \cup \{\mathtt{2r6} : (a'_j, out; e_j, in) \mid j \in I_{=0}\} \\ & \cup \{\underline{\mathtt{2r7}} : (a'_j, out; J_1, in) \mid j \in I_{=0}\} \\ & \cup \{\mathtt{2r8} : (e_j, out; d_j, in) \mid j \in I_{=0}\} \end{aligned}$$

$\cup\, \{\underline{\texttt{2r9}} : (e_j, out; J_1, in) \mid j \in I_{=0}\}$
$\cup\, \{\texttt{2r10} : (d_j, in) \mid j \in I_+ \cup I_-\}$
$\cup\, \{\texttt{2r11} : (a'_j, out) \mid j \in I_+ \cup I_-\}$
$\cup\, \{\texttt{2r12} : (J_2, out; b_j, in) \mid j \in I'\}$
$\cup\, \{\underline{\texttt{2r13}} : (J_2, out; J_1, in), \underline{\texttt{2r14}} : (\#, out; J_0, in)\}.$

First of all, we mention that if during the phase RUN object J_3 comes to the environment by rules $\underline{\texttt{1r11}}$, $\underline{\texttt{1r12}}$ (**Scenario 0**), it remains there forever and cannot move object L to region 2 (during the phase END), thus to stop the infinite loop. So, the computation never halts.

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 `1r6` returns J_0 and J_1 to their original locations.

If a_j comes to the environment without b_j leaving it or b_j is in region 1 or 2 at that moment (it is possible after applying rules $\underline{\texttt{2r3}}$, $\underline{\texttt{2r7}}$, $\underline{\texttt{2r13}}$), J_1 remains in region 1 (or 2) and J_0 comes to region 1 and after that in region 2 by rules `1r4`, $\underline{\texttt{2r14}}$ (**Scenario 1**), thus causing an endless computation since $\underline{\texttt{1r16}}$ and $\underline{\texttt{1r17}}$ are 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 $\underline{\texttt{1r12}}$ is applied and J_3 comes to the environment. The computation never halts, see scenario 0.

Scenario 3 takes place when two symbols a_j and symbol $b_j, j \in I$ appear in region 1 and in the environment accordingly. In this case rules `1r4`,`1r5` will be applied, and rule `1r4` two times. Thus, two symbols J_0 appear in region 1 and rule $\underline{\texttt{2r14}}$ will be applied eventually. The computation never halts, see scenario 1.

We also mention that applying rule $\underline{\texttt{1r11}}$ causes scenario 0 (this is a case of modeling a "decrement"-instruction, there is no c_k in region 1); applying $\underline{\texttt{2r5}}$ leads to scenario 3 (this is a case of modeling a "test for zero"-instruction, there is some c_k in region 1), and applying $\underline{\texttt{2r7}}$ and $\underline{\texttt{2r9}}$ eventually causing 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. We explain the behavior of simulating the instruction $(j : q_i \to q_l, c_k\gamma)$. Index s stands for any possible instruction associated to state q_l.

"Increment" -instruction:

$$q_l \mathbf{a_j} a_s a'_j e_j c_k J_0 \boxed{\mathbf{q_i} J_1 J_2 J_3 \boxed{b_j d_j \#}} \Rightarrow^{\texttt{1r1}} q_l q_i a_s a'_j e_j c_k J_0 \boxed{\mathbf{a_j} J_1 J_2 J_3 \boxed{\mathbf{b_j} d_j \#}} \Rightarrow^{\texttt{2r1}}$$
$$q_l q_i a_s \mathbf{a'_j} e_j c_k J_0 \boxed{\mathbf{b_j} J_1 \mathbf{J_2} J_3 \boxed{\mathbf{a_j} d_j \#}} \Rightarrow^{\texttt{1r3,2r2}} q_l q_i a_s \mathbf{b_j} e_j c_k \mathbf{J_0} \boxed{\mathbf{a'_j J_1 a_j} J_3 \boxed{J_2 \mathbf{d_j} \#}}$$
$$\Rightarrow^{\texttt{1r4,1r5,2r4}} q_l q_i a_j a_s \mathbf{e_j} c_k \mathbf{J_1} \boxed{\mathbf{b_j d_j J_0} J_3 \boxed{\mathbf{J_2 a'_j} \#}} \qquad \text{(A)}$$

$$\Rightarrow^{\texttt{1r6,1r13,2r11,2r12}} \mathbf{q_l} q_i a_j a_s \mathbf{d_j} \mathbf{c_k} J_0 \boxed{J_1 J_2 \mathbf{a'_j} \mathbf{e_j} J_3 \boxed{b_j \#}} \Rightarrow^{\texttt{1r7,1r9,1r14}}$$

$$q_i a_j \mathbf{a_s} a'_j e_j J_0 \boxed{\mathbf{q_l} \mathbf{d_j} J_1 J_2 J_3 c_k \boxed{b_j \#}} \Rightarrow^{\texttt{1r1,2r10}} q_l q_i a_j a'_j e_j J_0 \boxed{\mathbf{a_s} J_1 J_2 J_3 c_k \boxed{b_j d_j \#}}$$

In that way, q_i is replaced by q_l and c_k is moved from the environment into region 1. Notice that symbols a_j, b_j, a'_j, d_j, e_j, J_0, J_1, J_2 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).

"Decrement" -instruction:

(i) *There is some* c_k *in region 1:*

We consider configuration (A) above with symbol c_k in region 1.

$$q_l q_i a_j a_s \mathbf{e_j} \mathbf{J_1} \boxed{\mathbf{b_j} \mathbf{d_j} \mathbf{J_0} J_3 c_k \boxed{\mathbf{J_2} \mathbf{a'_j} \#}} \Rightarrow^{\texttt{1r6,1r13,2r11,2r12}}$$

$$\mathbf{q_l} q_i a_j a_s \mathbf{d_j} J_0 \boxed{J_1 J_2 \mathbf{a'_j} \mathbf{e_j} J_3 \mathbf{c_k} \boxed{b_j \#}} \Rightarrow^{\texttt{1r8,1r10,1r14}} q_i a_j \mathbf{a_s} a'_j e_j c_k J_0 \boxed{\mathbf{q_l} J_1 J_2 J_3 \mathbf{d_j} \boxed{b_j \#}}$$

$$\Rightarrow^{\texttt{1r1,2r10}} q_l q_i a_j a'_j e_j c_k J_0 \boxed{\mathbf{a_s} J_1 J_2 J_3 \boxed{b_j d_j \#}}$$

In the way described above, q_i is replaced by q_l and c_k is removed from region 1 to the environment. Notice that symbols a_j, a'_j, b_j, d_j, e_j, J_0, J_1, J_2 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).

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

Again we start with configuration (A).

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

$$\Rightarrow^{\texttt{1r6,1r13,2r11,2r12}} \mathbf{q_l} q_i a_j a_s \mathbf{d_j} J_0 \boxed{J_1 J_2 \mathbf{a'_j} \mathbf{e_j} \mathbf{J_3} \boxed{b_j \#}} \Rightarrow^{\texttt{1r8,}\underline{\texttt{1r11}}\texttt{,1r14}}$$

Now rule <u>**`1r11`**</u> will be applied, leading to an infinite computation (see scenario 0).

"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:*

We consider configuration (A) above.

$$q_l q_i a_j a_s \mathbf{e_j J_1} \boxed{\mathbf{b_j d_j J_0} J_3 \boxed{\mathbf{J_2} a'_j \#}} \Rightarrow^{\mathtt{1r6,1r13,2r12}} q_l q_i a_j a_s \mathbf{d_j} J_0 \boxed{J_1 J_2 \mathbf{e_j} J_3 \boxed{\mathbf{a'_j} b_j \#}}$$

$$\Rightarrow^{\mathtt{1r9,2r6}} q_l q_i a_j a_s J_0 \boxed{\mathbf{d_j} J_1 J_2 J_3 \mathbf{a'_j} \boxed{\mathbf{e_j} b_j \#}} \Rightarrow^{\mathtt{1r8,2r8}} \mathbf{q_l} q_i a_j a_s a'_j J_0 \boxed{\mathbf{e_j} J_1 J_2 J_3 \boxed{b_j d_j \#}}$$

$$\Rightarrow^{\mathtt{1r14}} q_i a_j \mathbf{a_s} a'_j e_j J_0 \boxed{\mathbf{q_l} J_1 J_2 J_3 \boxed{b_j d_j \#}}$$

In this case, q_i is replaced by q_l. Notice that symbols a_j, a'_j, b_j, d_j, e_j, J_0, J_1, J_2 have returned to their original positions.

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

$$q_l q_i a_j a_s \mathbf{e_j J_1} \boxed{\mathbf{b_j d_j J_0} J_3 \mathbf{c_k} \boxed{\mathbf{J_2 a'_j} \#}} \Rightarrow^{\mathtt{1r6,1r13,\underline{2r5},2r12}}$$

Now applying rule $\underline{\mathtt{2r5}}$ leads to an infinite computation.

$$\mathbf{q_l} q_i a_j a_s a_s a'_s \mathbf{d_j} J_0 J_0 \boxed{J_1 J_2 \mathbf{a'_j e_j} J_3 \boxed{b_j b_s c_k \#}} \Rightarrow^{\mathtt{1r8,1r9,1r14}}$$

$$q_i a_j a_s \mathbf{a_s} a'_j a'_s \mathbf{e_j} J_0 J_0 \boxed{\mathbf{q_l d_j} J_1 J_2 J_3 \boxed{b_j b_s c_k \#}} \Rightarrow^{\mathtt{1r1,1r13}}$$

$$\mathbf{q_l} q_i a_j a_s a'_j a'_s \mathbf{d_j} J_0 J_0 \boxed{\mathbf{a_s e_j} J_1 J_2 J_3 \boxed{b_j \mathbf{b_s} \#}} \Rightarrow^{\mathtt{1r14,2r1}}$$

$$q_i a_j \mathbf{a_s} a'_j \mathbf{a'_s} e_j J_0 J_0 \boxed{\mathbf{b_s q_l} J_1 \mathbf{J_2} J_3 \boxed{\mathbf{a_s} d_j \#}} \Rightarrow^{\mathtt{1r1,1r3,2r2}}$$

$$q_l q_i a_j a'_j \mathbf{b_s} e_j \mathbf{J_0 J_0} \boxed{\mathbf{a_s a_s a'_s J_1} J_2 J_3 \boxed{d_j \#}}$$

So, scenario 3 takes place and the computation never halts.

3. END.

$$\begin{aligned} R_{1,f} &= \{\mathtt{1f1} : (T_1, out; F_1, in)\} \cup \{\mathtt{1f2} : (F_i, out; F_{i+1}, in) \mid 1 \le i \le 8\} \\ &\cup \{\mathtt{1f3} : (T_2, out; P_1),\ \mathtt{1f4} : (P_2, out),\ \mathtt{1f5} : (F_0, out; P_2, in)\}. \\ R_{2,f} &= \{\mathtt{2f1} : (T_1, out; F_0, in), \mathtt{2f2} : (F_0, out), \mathtt{2f3} : (T_2, out; F_0, in)\} \\ &\cup \{\mathtt{2f4} : (P_1, in), \mathtt{2f5} : (P_1, out; J_1, in), \mathtt{2f6} : (P_1, out; J_2, in)\} \\ &\cup \{\mathtt{2f7} : (P_1, out; J_3, in), \mathtt{2f8} : (J_3, out; L, in), \mathtt{2f9} : (P_2, out; F_9, in)\}. \end{aligned}$$

Once the counter automaton reaches the final state, q_f is in region 1 and it exchanges with object a_n (rule `1r2`) and object F_0 will be moved to region 1 in several steps (rules `1r15`).

It takes T_1 and T_2 to region 1, in either order. The duty of T_2 is to bring P_1 from the environment to region 2, where P_1 pumps objects J_1, J_2, J_3 from region 1 to region 2. If on the previous steps of simulation of counter automaton M object

J_3 was moved to the environment (by rules 1r11, 1r12), scenario 0 takes place and the computation never halts, as there is only one possibility to stop an infinite loop with object L, i.e. to move it to region 2 by rule 2f8.

T_1 starts a chain of exchanges of objects F_i, as a result object F_9 will be moved to region 1 and then object P_2 will be moved to the environment, where it pumps object F_0 to the environment. So, at the end of the computation there are only objects $c_k, k \in D$ in region 1. The entire simulation shows the inclusion $N(\Pi_1) \supseteq N(M)$.

The converse inclusion also holds because the system may only halt if it has correctly simulated a computation of the counter automaton (according to the design of the system) from state q_0 to state q_f, while if behavior of M is not simulated correctly, then the computation never halts and hence does not contribute to $N(\Pi_1)$. This shows that P systems with two membranes and symport/antiport rules of weight one with the output in the skin membrane generate all recursively enumerable sets of natural numbers. Since the power of such systems cannot exceed that of Turing machines, the statement of the theorem is an equality. □

4 Conclusions

In this paper we prove the new result that any recursively enumerable set of natural numbers is generated by symport/antiport P systems with two membranes and minimal cooperation where the output membrane is the skin membrane. It contrasts with the previous result where an elementary membrane is used as the output membrane, where at least one superfluous object is necessary in the output membrane in order to get universality. Thus we answered the question of Francesco Bernardini about computational power of symport/antiport P systems with two membranes and minimal cooperation where the output membrane is the skin membrane. The question about power of purely symport P systems with two membranes and minimal cooperation where the output membrane is the skin membrane is still open.

References

1. A. Alhazov, R. Freund, Yu. Rogozhin: Computational Power of Symport/Antiport: History, Advances, and Open Problems. 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 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 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 (H.J. Hoogeboom, Gh. Păun, G. Rozenberg, A. Salomaa, Eds.) *Lecture Notes in Computer Science 4361* (2006) 135–153.
6. A. Alhazov, Yu. Rogozhin: Generating Languages by P Systems with Minimal Symport/Antiport. *Computer Science Journal of Moldova*, *14*, 3(42) (2006) 299–323.
7. 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.
8. A. Alhazov, Yu. Rogozhin and S. Verlan: Minimal Cooperation in Symport/Antiport Tissue P Systems. *International Journal of Foundation of Computer Science*, *18*, 1 (2007) 163–179.
9. 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.
10. 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 2933* (2004) 43–45.
11. R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae 49*, 1–3 (2002) 81–102.
12. 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 2597* (2003) 261–268.
13. 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 2597* (2003) 270–287.
14. 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.
15. 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 2597* (2003) 288–301.

16. P. Frisco, H.J. Hoogeboom: P Systems with Symport/Antiport Simulating Counter Automata. *Acta Informatica 41*, 2–3 (2004) 145–170.
17. 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 2950* (2004) 254–265.
18. 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.
19. 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.
20. M.L. Minsky: *Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey (1967).
21. A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing 20* (2002) 295–305.
22. Gh. Păun: Computing with Membranes. *Journal of Computer and Systems Science 61* (2000) 108–143.
23. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag (2002).
24. 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.
25. 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.
26. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages* (3 volumes). Springer-Verlag, Berlin (1997).
27. 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.
28. S. Verlan: Optimal Results on Tissue P Systems with Minimal Symport/ Antiport. Presented at *EMCC meeting*, Lorentz Center, Leiden (2004).
29. 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 3340*, Springer-Verlag, Berlin (2004) 418–430.
30. P Systems Webpage, `http://psystems.disco.unimib.it`