# Symport/Antiport Tissue P Systems with Minimal Cooperation

Artiom Alhazov[1,2], Yurii Rogozhin[1], Sergey Verlan[3]

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

[2] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: artiome.alhazov@estudiants.urv.es

[3] Laboratoire d'Informatique Théorique et Appliquée
Université de Metz
Ile du Saulcy, 57045 Metz Cedex, France
E-mail: verlan@sciences.univ-metz.fr

**Summary.** We investigate tissue P systems with symport/antiport with minimal cooperation, i.e., when only two objects may interact. We show that 2 cells are enough in order to generate all recursively enumerable sets of numbers. Moreover, the constructed systems simulate register machines and have a purely deterministic behavior. We also investigate systems with one cell and we show that they may generate only finite sets of numbers.

## 1 Introduction

P systems were introduced by Gh. Păun in [12] as distributed parallel computing devices of biochemical inspiration. These systems are inspired from the structure and the functioning of a living cell. The cell is considered as a set of compartments (membranes) nested one in another and which contain objects and evolution rules. The basic model specifies neither the nature of these objects, nor the nature of rules. Numerous variants specify these two parameters by obtaining a lot of different models of computing (we refer to [18] for a comprehensive bibliography). One of these variants, P systems with *symport/antiport*, was introduced in [11]. This variant uses one of the most important properties of P systems: the communication. This property 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, they may compute all Turing computable sets of numbers [11]. This result was improved with respect to the number of used membranes and/or the weight of symport/antiport rules ([5], [7], [9], [13], [4]).

Rather unexpectedly, minimal symport/antiport membrane systems, *i.e.*, when one uses only one object in symport rules and a pair of objects in antiport rules are universal. The proof of this result may be found in [3] and the corresponding system has 9 membranes. This result was improved first by reducing the number of membranes to six [8], after that to five [4] and four [6]. In [16] G. Vaszil showed that three membranes are sufficient to generate all recursively enumerable sets of numbers. Another proof of the same result that was obtained independently may be found in [2].

Similarly, when minimal symport is used, i.e., when only symport rules dealing with at most two objects are permitted, three membranes suffice to generate all recursively enumerable sets of numbers [2]. A recent result shows that this number can be decreased down to two for both minimal symport and minimal antiport cases [1].

The inspiration for *tissue P systems* comes from two sides. From one hand, P systems previously introduced may be viewed as transformations of labels associated to nodes of a tree. Therefore, it is natural to consider same transformations on a graph. On the other hand, they may be obtained by following the same reflections as for P systems, but starting from a tissue of cells and no more from a single cell.

Tissue P systems were first considered by Gh. Păun and T. Yokomori in [14] and [15]. They have richer possibilities and the advantages of the new topology have to be investigated. Tissue P systems with symport/antiport were first considered in [13] where several results having different values of parameters (graph size, maximal size of connected component, weight of symport and antiport rules) are presented.

In this paper we consider symport and antiport rules with minimal cooperation, i.e., we consider minimal antiport rules (of weight 1) that permit to exchange two objects, as well as minimal symport rules (of weight 2) that permit to send two objects together. Systems with minimal antiport were first investigated in [17] where it was shown that 3 cells are sufficient to generate any recursively enumerable set of numbers.

In this article, we investigate both variants and we show that in both cases we can construct systems defined on a graph with 3 nodes, i.e., 2 cells, that simulate any (non-)deterministic register machine. Moreover, in the deterministic case, the obtained systems are also deterministic and only one evolution is possible at any time. Therefore, if the computation stops, then we are sure that the corresponding register machine stops on the provided input. Another difference from previous proofs is that we use a very small amount of symbols present in an infinite number of copies in the environment.

We also show that if only one cell is considered, then the corresponding systems may generate no more than a finite language. The minimal symport case was already discussed in [7], so we present only the antiport one.

As a consequence, we found a borderline between the decidability and the undecidability for the corresponding systems.

## 2 Definitions

We denote by $\mathbb{N}$ the set of all non-negative natural numbers: $\{0, 1, \ldots\}$ and by $\mathbb{N}'$ the set $\{1, 2, \ldots\}$.

A *multiset* $S$ over $O$ is a mapping $f_S : O \longrightarrow \mathbb{N}$. The mapping $f_S$ specifies the number of occurrences of each element of $S$. The size of the multiset $S$ is $|S| = \sum_{x \in O} f_S(x)$. The multiset defined by the mapping $a \rightarrow 3, b \rightarrow 1, c \rightarrow 0$ will be specified as $\{a^3, b\}$ or $a^3b$.

The sum of two multisets $P$ and $Q$ over $O$ is a multiset $S$ such that $f_S(a) = f_P(a) + f_Q(a)$ for all $a$ in $O$. Similarly, the difference of two multisets $P$ and $Q$ is a multiset $S$ having $f_S(a) = f_P(a) \ominus f_Q(a)$ where $\ominus$ is the positive subtraction.

Multisets defined as above are called *finite* multisets. If we consider that mapping $f_S$ is of form $f_S : O \longrightarrow \mathbb{N} \cup \{\infty\}$, i.e., elements of $S$ may have an infinite multiplicity, then we obtain *infinite* multisets.

A deterministic *register machine* is the following construction:

$$M = (Q, R, q_0, q_f, P),$$

where $Q$ is a set of states, $R = \{r_1, \ldots, r_k\}$ is the set of registers (called also counters), $q_0 \in Q$ is the initial state, $q_f \in Q$ is the final state and $P$ is a set of instructions (called also rules) of the following form:

1. $(p, A+, q) \in P$, $p, q \in Q, p \neq q, A \in R$ (being in state $p$, increase register $A$ and go to state $q$).
2. $(p, A-, q, s) \in P$, $p, q, s \in Q, A \in R$ (being in state $p$, decrease register $A$ and go to $q$ if successful or to $s$ if $A$ is zero).
3. STOP (may be associated only to the final state $q_f$).

We note that for each state $p$ there is only one instruction of the type above.

A configuration of a register machine is given by the $k+1$-tuple $(q, n_1, \ldots, n_k)$ describing the current state of the machine as well as the contents of all registers. A transition of the register machine consists in updating/checking the value of a register according to an instruction of one of types above and by changing the current state to another one.

We say that $M$ computes a value $y \in \mathbb{N}$ on the *input* $x \in \mathbb{N}$ if starting from the initial configuration $(q_0, x, 0, \ldots, 0)$ it reaches the final configuration $(q_f, y, 0, \ldots, 0)$.

We say that $M$ recognizes the set $S \subseteq \mathbb{N}$ if for any input $x \in S$ the machine stops and for any $y \notin S$ the machine performs an infinite computation. It is known that register machines recognize all recursively enumerable sets of numbers [10].

We may also consider non-deterministic register machines where the first type of instruction is of the form $(p, A+, q, s)$ and with the following meaning: if the machine is in state $p$, then the counter $A$ is increased and the current state is changed to $q$ or $s$ non-deterministically. In this case the result of the computation is the set of all values of the first counter when the computation halts. We assume that the machine empties all counters except the first counter before stopping. It is known that non-deterministic register machines generate all recursively enumerable sets of non-negative natural numbers starting from empty counters.

A *tissue P system with simport/antiport* of degree $m \geq 1$ is a construct

$$\Pi = (O, G, w_1, \ldots, w_m, E, R, i_0),$$

where $O$ is the alphabet of objects and $G$ is the underlying directed labelled graph of the system. The graph $G$ has $m+1$ nodes and the nodes are numbered from 0 to $m$. We shall also call nodes from 1 to $m$ cells and node 0 the environment. There is an edge between each cell $i$, $1 \leq i \leq m$ and the environment. Each node has a label that contains a multiset of objects. The environment is a special node whose label may contain an infinite multiset. The symbols of the multiset labelling the environment which are present with an infinite multiplicity are given by the set $E$. The symbols $w_1, \ldots, w_m$ are multisets over $O$ that give initial labels of nodes of $G$. The symbol $i_0$ is the output node, and $R$ is a finite set of rules (associated to edges) of the following forms:

1. $(i, x, j)$, $1 \leq i \leq m, 0 \leq j \leq m, i \neq j$, $x \in O^+$ (symport rules),
2. $(i, x/y, j)$, $0 \leq i, j \leq m, i \neq j$, $x, y \in O^+$(antiport rules).

The first rule sends a multiset of objects $x$ from node $i$ to node $j$. The second rule exchanges multisets $x$ and $y$ situated in nodes $i$ and $j$ respectively. The weight of symport rule $(i, x, j)$ is equal to $|x|$, while the weight of an antiport rule is equal to $\max\{|x|, |y|\}$.

A computational step is made by applying all rules in a non-deterministic maximal parallel way. A configuration of the system is the following $m+1$-tuple $(z_0, z_1, \ldots, z_m)$ where each $z_i, 1 \leq i \leq m$ represents the label of vertex $i$ and $z_0$ represents objects that appear with a finite multiplicity in the environment (initially $z_0$ is an empty multiset). The computation stops when no rule may be applied. The result of a computation is given by the number of objects situated in cell $i_0$, i.e., by the size of the multiset labelling vertex $i_0$.

We denote by $NOtP_{m,n}(sym_p, anti_q)$ the family of all sets of numbers computed by tissue P systems with symport/antiport of degree at most $m$ with the maximal size of the connected component of the restriction of the graph to cells equal to $n$ and which have symport rules of weight at most $p$ and antiport rules of weight at most $q$.

## 3 Universality Results

We consider first the case of symport/antiport of weight 1.

**Lemma 1** *For any deterministic register machine $M$ and for any input $I_n$ there is a tissue P system with symport/antiport of degree 2 having symport and antiport rules of weight* 1*, which simulates $M$ on this input and produces the same result.*

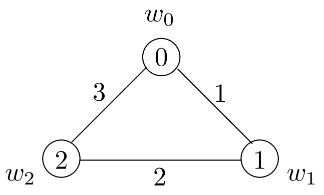
*Proof.* We consider an arbitrary deterministic register machine $M = (Q, R, q_0, q_f, P)$ and we construct a tissue P system with symport/antiport that will simulate this machine on the input $I_n$. We consider a more general problem: we shall simulate $M$ on any initial configuration $(q_0, N_1, \dots, N_k)$.

We assume, for simplicity, that we may have objects initially present in a finite number of copies in the environment and we denote this multiset by $w_0$. We shall show later that this assumption is not necessary.

We define the system as follows.

$\Pi = (O, G, w_0, w_1, w_2, R, R', 2)$,
$O = Q \cup R \cup \{A^+_{pq} \mid (p, A+, q) \in P\} \cup \{p', p'', Q^-_{pqs}, Q^0_{pqs} \mid (p, A-, q, s) \in P\}$.

We consider the following underlying graph $G$:



Below we give in tables rules and objects of our system. In fact, each $w_i$, $0 \le i \le 2$, as well as $R$ is the union of corresponding cells in all tables.

Rule numbers follow the following convention: the second number is the number of the edge where the rule is located, the first number indicates which instruction is simulated using this rule (1 for incrementing, 2 for decrementing, 3 for stop and 0 for common parts) and the third is the number in group.

Encoding of initial configuration of $M$ $(1 \le j \le k)$:

| Node | Object(s) |
|---|---|
| 0. | |
| 1. | |
| 2. | $r_j^{N_j}, q_0$ |

Common rules and objects ($q \in Q$, $A \in R$, $p \in Q \setminus \{q_0\}$):

| Node | Object(s) |
|---|---|
| 0. | $A^\infty$ |
| 1. | $p$ |
| 2. | |

| Edge | Rules |
|---|---|
| 1. | |
| 2. | |
| 3. | 0.3.1 : $(0, q, 2)$ |

For any rule $(p, A+, q) \in P$ we have following rules and objects:

| Node | Object(s) |
|---|---|
| 0. | |
| 1. | $A^+_{pq}$ |
| 2. | |

| Edge | Rules |
|---|---|
| 1. | $1.1.1 : (0, A^+_{pq}/q, 1)$ |
| 2. | $1.2.1 : (2, p/A^+_{pq}, 1)$ |
| 3. | $1.3.1 : (2, A^+_{pq}/A, 0)$ |

For any rule $(p, A-, q, s) \in P$ we have following rules and objects:

| Node | Object(s) |
|---|---|
| 0. | $p''$ |
| 1. | $p', Q'_{pqs}, Q^0_{pqs}$ |
| 2. | |

| Edge | *Rules* | |
|---|---|---|
| 1. | $2.1.1 : (0, p'/Q'_{pqs}, 1)$ | $2.1.2 : (1, p'', 0)$ |
| | $2.1.3 : (0, Q'_{pqs}/Q^0_{pqs}, 1)$ | $2.1.4 : (0, Q^0_{pqs}/s, 1)$ |
| 2. | $2.2.1 : (2, p/p', 1)$ | $2.2.2 : (2, p''/Q^0_{pqs}, 1)$ |
| | $2.2.3 : (2, Q'_{pqs}/q, 1)$ | $2.2.4 : (2, Q^0_{pqs}, 1)$ |
| 3. | $2.3.1 : (2, p'/p'', 0)$ | $2.3.2 : (2, A/Q'_{pqs}, 0)$ |

Rules and objects associated to the STOP instruction:

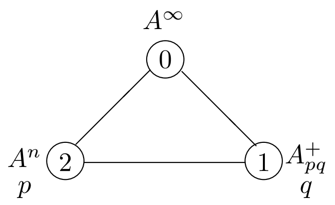
| Edge | *Rules* |
|---|---|
| 1. | |
| 2. | $3.2.1 : (2, q_f, 1)$ |
| 3. | |

We organize the system $\Pi$ as follows. Node 2 contains the current configuration of machine $M$. Node 1 contains one copy of objects that correspond to each state. In the same node, there are additional symbols used for the simulation of the decrementing operation and which are present in one copy. Node 0 (the environment) contains symbols $r_j$, $1 \leq j \leq k$, which are used to increment registers. These symbols are present in an infinite number of copies.

Each configuration $(p, n_1, \dots, n_k)$ of machine $M$ is encoded as follows. Cell 2 contains objects $r_j$ of the multiplicity $n_j$, $1 \leq j \leq k$ as well as the object $p$. It is easy to observe that the initial configuration of $\Pi$ corresponds to an encoding of the initial configuration of $M$.

Now we shall discuss the simulation of instructions of $M$.

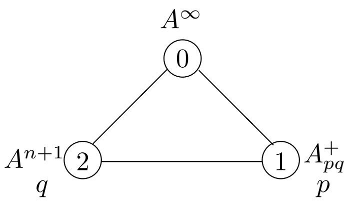
**Incrementing.** Suppose that $M$ is in configuration $(p, n_1, \dots, n_k)$ and that there is a rule $(\mathbf{p}, \mathbf{A}+, \mathbf{q})$ in $P$ $(A = r_j)$. Suppose that the value of $A$ is $n$ $(n_j = n)$. This corresponds to the following configuration of $\Pi$ (see below) where we indicate only symbols that we effectively use.



Now we present the evolution of the system in this case.

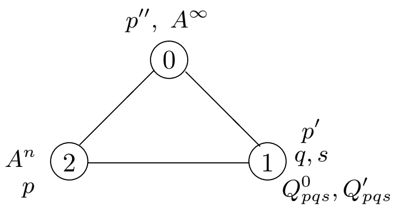
The symbol $p$ in the second node that encodes the current state of $M$ triggers the application of rule 1.2.1 and it exchanges with $A^+_{pq}$ which comes to the second node.

After that, the last symbol brings an object $A$ to node 2, which corresponds to an incrementing of register $A$. Further, symbol $A^+_{pq}$ goes to node 1 and brings from there to node 0 the new state $q$. After that, symbol $q$ moves to node 2. The configuration is as follows:



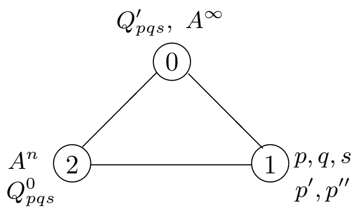
The last configuration differs from the first one by the following. In node 2, there is one more copy of object $A$ and the object $p$ was replaced by the object $q$. All other symbols remained on their places and the symbol $p$ was moved to node 1 which contains as before one copy of each state of the register machine that is not current. This corresponds to the following configuration of $M$: $(q, n_1, \ldots, n_j + 1, \ldots, n_k)$, i.e., we simulated the corresponding instruction of $M$.

**Decrementing.** Suppose that $M$ is in configuration $(p, n_1, \ldots, n_k)$ and that there is a rule $(\mathbf{p}, \mathbf{A}-, \mathbf{q}, \mathbf{s})$ in $P$ $(A = r_j)$. Suppose that the value of $A$ is $n$ $(n_j = n)$. This corresponds to the following configuration of $\Pi$, where we indicate only symbols that we effectively use:



The idea of the decrementing is very simple. First we duplicate the signal that the current state is $p$ ($p'$ and $p''$). After that, these signals are propagated and if they may be synchronized, then this means that the value of the corresponding counter is zero. In the other case, the corresponding counter is decremented and the synchronization is no more possible.

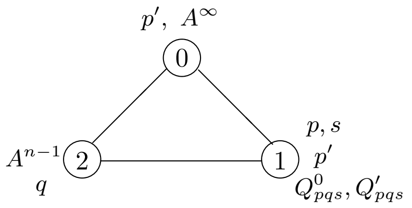
Now we shall give more details on the evolution of the system. The symbol $p$ in the second node that encodes the current state of $M$ triggers the application of rule 2.2.1 and it exchanges with $p'$ which comes to the second node. After that, the last symbol goes to node 0 bringing at the same time the symbol $p''$ in node 2. During next step, symbol $p'$ exchanges with $Q'_{pqs}$ and symbol $p''$ exchanges with $Q^0_{pqs}$. The obtained configuration is as follows:



Now there are two cases, $n > 0$ and $n = 0$, and the system behaves differently in each case.

## CASE A: $n > 0$

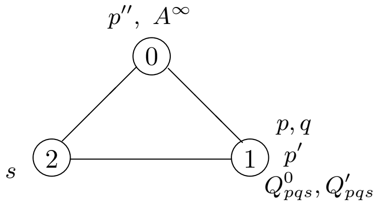
First, suppose that $n > 0$. In this case, $Q'_{pqs}$ goes to node 2 and brings a symbol $A$ to node 0, hence decrementing the counter. At the same time symbol $p''$ returns to the environment. Finally, the symbol $Q'_{pqs}$ brings the symbol $q$ to node 2 (see the configuration below).



We can see that the obtained configuration differs from the first one by the following. In node 2, there is one copy of object $A$ less and the object $p$ was replaced by the object $q$. All other symbols remained on their places and the symbol $p$ was moved to node 1 which contains as before one copy of each state of the register machine that is not current. This corresponds to the following configuration of $M$: $(q, n_1, \dots, n_j - 1, \dots, n_k)$, i.e., we simulated the corresponding instruction of $M$.

## CASE B: $n = 0$

Now suppose that $n = 0$. In this case, $Q'_{pqs}$ remains in node 0 for one more step. After that, it exchanges with $Q^0_{pqs}$ which returns to node 1. After that $Q^0_{pqs}$ brings in node 0 the symbol $s$ which moves after that to node 2 (see configuration at the right). We remark that the first exchange takes place only if the value of counter $A$ is equal to zero, otherwise rule 2.3.2 is applied and the exchange above cannot happen.



We can see that the obtained configuration differs from the first one by the following. In node 2, the object $p$ was replaced by the object $s$. All other symbols remained on their places and the symbol $p$ was moved to node 1 which contains as before one copy of each state of the register machine that is not current. This corresponds to the following configuration of $M$: $(s, n_1, \ldots, n_{j-1}, 0, n_{j+1}, \ldots, n_k)$, i.e., we simulated the corresponding instruction of $M$.

**Stop.** If the system is in halting state $q_f$, then rule 3.2.1 moves the symbol $q_f$ to node 1. Since node 2 contain only symbols corresponding to the value of the output counter, the system cannot evolve any more and the computation stops.

**Remarks.** It is clear that we simulate the behavior of $M$. Indeed, we simulate an instruction of $M$ and all additional symbols return to their places what permits to simulate the next instruction of $M$. Moreover, this permits to reconstruct easily a computation in $M$ from a successful computation in $\Pi$. For this it is enough to look for configurations which have a state symbol $p$ in node 2. We stop the computation when rule 3.2.1 is used and symbol $q_f$ goes to node 1. In this case, node 2 contains the result of the computation.

We remark that the assumption that $w_0 = \{p''\}$ for all decrementing instructions $(p, A-, q, s)$ is not necessary. Indeed, we may initially place $p''$ in node 1. □

**Theorem 1** *$NOtP_{2,2}(sym_1, anti_1) = NRE$.*

*Proof.* It is easy to observe that the system of previous lemma simulates the starting non-deterministic register machine. Indeed, in order to simulate a rule $(p, A+, q, s)$ of such machine, we use the same rules and objects as for rule $(p, A+, q)$. We only need to add a rule $1.1.1' : (0, A^+_{pq}/s, 1)$ to edge 1. □

We pass now to considering symport of weight 2.

**Lemma 2** *For any deterministic register machine $M$ and for any input $I_n$ there is a tissue P system with symport/antiport of degree 2 having only symport rules of weight at most 2, which simulates $M$ on this input and produces the same result.*

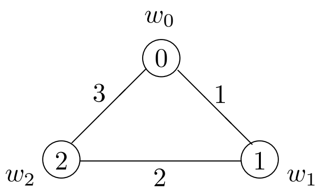
*Proof.* As in previous case we consider an arbitrary deterministic register machine $M = (Q, R, q_0, q_f, P)$ and we construct a tissue P system with symport/antiport that will simulate this machine on any initial configuration $(q_0, N_1, \ldots, N_k)$.

We assume, for simplicity, that we may have objects initially present in a finite number of copies in the environment and we denote this multiset by $w_0$. We shall show later that this assumption is not necessary. Let $m$ be the number of instructions of $M$ and $m_1$ the number of incrementing instructions. Let $1 \leq j \leq n+1$, where $n = m + 5(m - m_1 - 1) + m_1 = 2m - 4m_1 - 5$.

We define the system as follows.

$$\begin{aligned}
&\Pi = (O, G, w_0, w_1, w_2, E, R, 2),\\
&O = \{q, X_q \mid q \in Q\} \cup \{A^+_{pq} \mid (p, A+, q) \in P\} \cup \{Y, E, E_Y, E', S, V\}\\
&\quad \cup \{p', p'', p^0, Z_{pqs}, Z'_{pqs}, D_{pqs}, D'_{pqs}, X_{pqs}, X_{p^0} \mid (p, A-, q, s) \in P\} \cup \{E_j, E'_j\} \cup R,\\
&E = R.
\end{aligned}$$

We consider the following underlying graph $G$:



Below we give in tables rules and objects of our system. In fact, each $w_i$, $0 \leq i \leq 2$, as well as $R$ is the union of corresponding cells in all tables.

Rule numbers follow the following convention: the second number is the number of the edge where the rule is located, the first number indicates which instruction is simulated using this rule (1 for incrementing, 2 for decrementing, 3 for stop and 0 for common part) and the third is the number in group.

Encoding of initial configuration of $M$ ($1 \leq j \leq k$):

| Node | Object(s) |
|---|---|
| 0. | |
| 1. | |
| 2. | $r_j^{N_j}, q_0$ |

Common rules and objects ($q \in Q$, $A \in R$, $p \in Q \setminus \{q_0\}$):

| Node | Object(s) |
|---|---|
| 0. | $A^\infty, Y$ |
| 1. | $p$ |
| 2. | |

| Edge | *Rules* | |
|---|---|---|
| 1. | $0.1.1 : (1, A, 0)$ | |
| 2. | | |
| 3. | $0.3.1 : (0, qY, 2)$ | $0.3.2 : (2, Y, 0)$ |

For any rule $(p, A+, q) \in P$ we have following rules and objects:

| Node | Object(s) |
|---|---|
| 0. | |
| 1. | |
| 2. | $A^+_{pq}$ |

| Edge | Rules |
|---|---|
| 1. | $1.1.1 : (0, A^+_{pq}q, 1)$ |
| 2. | $1.2.1 : (2, pA^+_{pq}, 1)$ |
| 3. | $1.3.1 : (2, A^+_{pq}A, 0)$ |

For any rule $(p, A-, q, s) \in P$ we have following rules and objects:

| Node | Object(s) |
|---|---|
| 0. | $p'', D_{pqs}, D''_{pqs}, Z'_{pqs}$ |
| 1. | $p^0, D'_{pqs}, Z_{pqs}$ |
| 2. | $p'$ |

| Edge | *Rules* | |
|---|---|---|
| 1. | $2.1.1 : (1, p'p^0, 0)$ | $2.1.2 : (0, p''p^0, 1)$ |
| | $2.1.3 : (1, D_{pqs}D'_{pqs}, 0)$ | $2.1.4 : (1, D''_{pqs}q, 0)$ |
| | $2.1.5 : (0, Z_{pqs}Z'_{pqs}, 1)$ | $2.1.6 : (1, Z'_{pqs}s, 0)$ |
| 2. | $2.2.1 : (2, pp', 1)$ | $2.2.2 : (2, D_{pqs}A, 1)$ |
| | $2.2.3 : (1, p''Z_{pqs}, 2)$ | $2.2.4 : (2, D'_{pqs}Z_{pqs}, 1)$ |
| | $2.2.5 : (2, D''_{pqs}V, 1)$ | $2.2.6 : (1, V, 2)$ |
| 3. | $2.3.1 : (0, p'D_{pqs}, 2)$ | $2.3.2 : (0, D'_{pqs}D''_{pqs}, 2)$ |
| | $2.3.3 : (2, D_{pqs}Z_{pqs}, 0)$ | $2.3.4 : (2, p'', 0)$ |

Consider the following multisets over $O$:
$V_1 = \{X_q \mid q \in Q\}$, $n_1 = |V_1|$
$V_2 = \{X_{pqs} \mid (p, A-, q, s) \in P$, $n_2 = |V_2|$,
$V_3 = \{X_{p^0} \mid (p, A-, q, s) \in P$, $n_3 = |V_3|$,
$V_4 = \{p', A^+_{st} \mid (p, A-, q, l) \in P, (s, B+, t) \in P\}$, $n_4 = |V_4|$, and
$V = V_1 + V_2 + V_2 + V_3 + V_3 + V_4$, $n = |V|$ $(n = n_1 + 2 * n_2 + 2 * n_3 + n_4)$.

We number elements of $V = \{a_1, \dots, a_n\}$ as follows. Elements of $V_1$ will receive numbers from 1 to $n_1$. Elements of $V_2$ will receive numbers from $n_1+1$ to $n_1+2*n_2$. Elements of $V_3$ will receive numbers from $n_1 + 2 * n_2 + 1$ to $n_1 + 2 * n_2 + 2 * n_3$. Elements of $V_4$ will receive numbers from $n_1 + 2 * n_2 + 2 * n_3$ to $n$.

Below we give rules and objects associated to the STOP instruction ($1 \le i \le n$, $1 \le j \le n + 1$:

| Node | Object(s) |
|---|---|
| 0. | $E'_j, E'$ |
| 1. | $E_j, E_Y$ |
| 2. | $E, S, X_{pqs}, X_{pqs}, X_{p^0}, X_{p^0}, X_q$ |

| Edge | *Rules* | |
|---|---|---|
| 1. | $3.1.1 : (1, EE_Y, 0)$ | $3.1.2 : (0, E_YY, 1)$ |
| | $3.1.3 : (1, SE_1, 0)$ | $3.1.4 : (1, E'_iE_{i+1}, 0)$ |
| | $3.1.5 : (1, X_qq, 0)$ | $3.1.6 : (1, X_{pqs}Z_{pqs}, 0)$ |
| | $3.1.7 : (1, X_{p^0}p^0, 0)$ | |
| 2. | $3.2.1 : (2, q_fE, 1)$ | $3.2.2 : (2, E'S, 1)$ |
| | $3.2.3 : (2, E_ia_i, 1)$ | $3.2.4 : (2, E'_iV, 1)$ |
| | $3.2.5 : (2, E_{n+1}E, 1)$ | |
| 3. | $3.3.1 : (0, EE', 2)$ | $3.3.2 : (0, E_jE'_j, 2)$ |
| | $3.3.3 : (2, VE_{n+1}, 0)$ | |

We organize the system $\Pi$ as in Lemma 1. Node 2 contains the current configuration of machine $M$. All nodes contain additional symbols used for the simulation of $M$.

The simulation of $M$ is similar to Lemma 1. The incrementing of a register is done directly using the symbol $A^+_{pq}$. The decrementing operation is based on the same ideas as for Lemma 1. More exactly, the signal that the system has to perform the decrementing operation is duplicated ($p'$ and $p''$) and one symbol tries to decrement the corresponding register, while the other one is delayed in order to make the zero check. The key point consists in the fact that if the register is non-empty, then rules 2.2.2 and 2.2.3 are applied simultaneously and symbols $D_{pqs}$ and $Z_{pqs}$ cannot meet in node 2. Contrarily, if the corresponding register is empty, then these symbols meet in node 2 permitting the simulation of the corresponding branch.

The key difference from the previous lemma consists in the termination procedure. Since the system contains a lot of additional symbols, in particular in node 2, this node must be cleaned at the end of the computation avoiding at the same time possible conflicts with other rules. This is done in several stages. More precisely, the main problem is to move symbols $p'$ from node 2 to node 1. In order to solve this, it is sufficient to disable the group of rules 2.1.1. This might be done in several stages.

1. Move symbol $Y$ to node 1 (thus disabling rule 0.3.1).
2. Move all state symbols ($q$) to node 0 (disabling rule 2.1.6).
3. Move symbols $Z'_{pqs}$ to node 1 (disabling rule 2.1.5).
4. Move symbols $Z_{pqs}$ to node 0 (disabling rule 2.2.3).
5. Move symbols $p''$ to node 1 (disabling rule 2.1.2).
6. Move symbols $p^0$ to node 0 (disabling rule 2.1.1).
7. Finally move symbols $p'$ to node 1.

Rules 3.2.1, 3.1.1 and 3.1.2 permit to realize the first stage. Rules 3.3.2, 3.2.3, 3.2.4 and 3.1.4 permit to move symbols $a_i$ from node 2 to node 1 one after another. Now the main idea is to use existing rules from the decrementing phase and a special numeration of symbols in a way that will permit to accomplish all stages. First symbols $X_q$ are processed. After being moved to node 1 these symbols move together with corresponding states $q$ to the environment. Since the symbol $Y$ is no more present, symbols $q$ will remain there.

Next, symbols $X_{pqs}$ are processed. These symbols are present in two copies. When the first copy arrives in node 1, it moves the corresponding symbol $Z_{pqs}$ to node 0. After that, symbol $Z_{pqs}$ brings symbol $Z'_{pqs}$ to node 1 (rule 2.1.5). Now, the second copy of $X_{pqs}$ will definitively move symbols $Z_{pqs}$ to node 0, hence realizing stages 3 and 4.

In a similar way, symbols $X_{p^0}$ permit to move symbols $p''$ to node 1 and symbols $p^0$ to node 0.

Finally, all remaining symbols from node 2 are transferred to node 1 or 0.

**Remarks.** Following the same arguments like in the proof of Lemma 1 it is clear that we simulate the behavior of $M$.

Now we shall show that the assumption that $w_0$ is not empty is not necessary. Indeed, it is enough to place initially all symbols $s_i$ that are in $w_0$, except $E'$, in node 2. In the same node $|w_0| - 1$ symbols $U$ shall be placed. After that, rules $(2, s_iU, 0)$ shall be added. It is clear that during the first step all symbols $s_i$ will move to the environment. The remaining symbol $E'$ shall be placed in node 1, as well as a copy of the symbol $U$. A similar rule $(1, E'U, 0)$ will move $E'$ to the environment at the first step of the computation. Finally, in order to avoid a possible application of rules 2.2.5 and 3.2.4, the symbol $V$ shall be initially placed in node 1. □

**Theorem 2** $NOtP_{2,2}(sym_2, anti_0) = NRE.$

*Proof.* It is easy to observe that the system of previous lemma simulates the starting non-deterministic register machine. Indeed, in order to simulate a rule $(p, A+, q, s)$ of such machine, we use the same rules and objects as for rule $(p, A+, q)$. We only need to add a rule $1.1.1' : (0, A^+_{pq}/s, 1)$ to the edge 1. □

**Descriptional Complexity**

Let $M$ be a register machine having $\mathsf{n}$ states, $\mathsf{k}$ registers and $\mathsf{n}_1$ incrementing instructions, i.e., $M$ has $\mathsf{n} - \mathsf{n}_1 - 1$ decrementing instructions. In this case the system constructed as in Lemma 1 need at most $n_1+4(n-n_1-1)+n = 5\mathsf{n} - 3\mathsf{n}_1 - 4$ symbols present in one copy, $\mathsf{k}$ symbols present in infinite number of copies, $3n_1 + 8(n-n_1-1) = 8\mathsf{n} - 5\mathsf{n}_1 - 8$ antiport rules and $n+2(n-n_1-1)+1 = 3\mathsf{n} - 2\mathsf{n}_1 - 1$ symport rules. The system constructed as in Lemma 2 need at most $2n + n_1 + 6 + 9(n - n_1 - 1) + 2(n + 5(n - n_1 - 1) + n_1) = 14\mathsf{n} - 16\mathsf{n}_1 - 13$ symbols present in one copy, $\mathsf{k}$ symbols present in infinite number of copies $n + 3n_1 + 14(n - n_1 - 1) + 8 + n + 2(n - n_1 - 1) + 4(n + 5(n - n_1 - 1) + n_1) + 1 = 42\mathsf{n} - 29\mathsf{n}_1 - 28$ symport 2 rules and $k + 1 + 1 + (n - n_1 - 1) = \mathsf{n} - \mathsf{n}_1 + \mathsf{k} + 1$ symport 1 rules.

## 4 Decidability Results

In this section we show that systems with symport/antiport of weight 1 having only one cell are not very powerful. A similar result concerning systems with minimal symport may be found in [7].

**Theorem 3** $NOtP_{1,1}(sym_1, anti_1) \subseteq NFIN.$

*Proof.* Consider an arbitrary P system $\Pi$ with 1 membrane and symport/antiport rules of weight 1, $\Pi = (O, E, [_1\ \ ]_1, w, R)$. Consider also an arbitrary halting computation, ending in some configuration $C$ with $w_1 \in (O - E)^*$ and $w_e \in E^*$ in membrane 1 and $w_0 \in (O - E)^*$ in the environment. We are claiming that $|w_1| + |w_e| \leq |w|$.

We shall prove this assertion by contradiction. Let us assume the contrary. Since the number of objects in the membrane can only increase by symport rules, some rule $p_0 : (s_0, in)$ had to be applied at some step (by definition $s_0 \in O - E$). This implies that $s_0$ has been brought to the environment. We can assume that rules $p_i : (s_i; s_{i-1}, in)$, $1 \leq i < n$, have been applied $(n \geq 0)$, $s_i \in O - E$, $1 \leq i \leq n$. Suppose also that $n$ is maximal ($s_n$ was not brought to the environment by antiport with another object from $O - E$). Thus $R$ contains either a rule $p : (s_n, out)$, or $p' : (s_n, out, a, in)$, $a \in O$.

Now let us examine the final configuration. If $s_0$ is in $w_1$, then $p_0$ can be applied, hence the configuration is not final. Therefore $s_0$ is in $w_0$. For all $1 \leq i \leq n$, given $s_{i-1}$ in $w_0$, if $s_i$ is in $w_1$, then $p_i$ can be applied, hence the configuration is not final. Consequently, $s_i$ is in $w_0$ as well. By induction, we obtain that $s_n$ is in $w_0$. However, this implies that either $p \in R$ and $p$ can be applied, or some $p' \in R$ and $p'$ can be applied, therefore the configuration is not final. This implies that any computation where number of objects inside the membrane is increased cannot halt. Therefore, $\Pi$ can only generate numbers not exceeding $|w|$. The statement of the theorem follows directly from here. □

## 5 Conclusions

In this article we studied tissue P systems with symport/antiport. We considered variants using minimal cooperation (symport of weight 2 or antiport of weight 1) and we showed that they are able to generate any recursively enumerable set of numbers with only 2 cells. We remark that contrarily to other proofs concerning P systems with symport/antiport, the system that we constructed is deterministic. Consequently, if we have a deterministic register machine $M$ and an initial configuration $(q_0, n_1, \ldots, n_k)$ of this machine, then the corresponding system constructed as described in Lemma 1 and Lemma 2 will have a deterministic behavior, i.e., there will be only one possible evolution at each step and it will halt if and only if $M$ halts on the above configuration. We highlight this deterministic evolution because it makes the behavior of the system more predictable and makes such systems good candidates for possible implementations.

Another advantage of our proof technique with respect to other proofs is that we need only $k$ symbols present in an infinite number of copies, where $k$ is the number of registers of $M$. All other symbols are present in one or two copies.

We also showed that if only one cell is considered, then corresponding systems generate at most finite languages. Therefore, we completely solved the open question about the minimal antiport from [17], as well as about the minimal symport. We remark that the proof of Theorem 3 holds in the case of ordinary P systems. However, the difference between the two models in this case is minimal.

An open problem concerns the number of symport rules used for minimal antiport systems. In our system we used $3n - 2n_1 - 1$ symport rules. Using ideas from [17] it is possible to decrease the number of symport rules down to 4. The question if we may obtain the same result using a smaller number of symport rules remains open. We remark that because we deal with antiport rules of size one, we need at least one symport rule, otherwise we cannot change the number of objects in the system.

**Acknowledgements**

The first author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona. The second author is supported by the NATO project PST.CLG.976912. The first and the second authors also acknowledge the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034.

## References

1. A. Alhazov, R. Freund, Y. Rogozhin: Some optimal results on symport/antiport P systems with minimal cooperation. In the present volume, 23–36.
2. A. Alhazov, M. Margenstern, V. Rogozhin, Y. Rogozhin, S.Verlan: Communicative P systems with minimal cooperation. In *Membrane Computing, Fifth International Workshop, WMC5* (G. Mauri, Gh. Păun, M.J. Pérez Jiménez, G. Rozenberg, A. Salomaa, eds.), LNCS 3365 (2005), 161–177.
3. F. Bernardini, M. Gheorghe: On the power of minimal symport/antiport. In *Pre–proceedings of the Workshop on Membrane Computing* (A. Alhazov, C. Martín-Vide, Gh. Păun, eds.), Tarragona, July 17-22, 2003, 72–83.
4. F. Bernardini, A. Păun: Universality of minimal symport/antiport: Five membranes suffice. In *Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain, July, 17-22, 2003, Revised Papers* (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 2933 (2004), 43–54.
5. R. Freund, A. Păun: Membrane systems with symport/antiport: Universality results. In *Membrane Computing: International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002. Revised Papers.* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597 (2003), 270–287.
6. P. Frisco: About P systems with symport/antiport. In *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, University of Seville, 2004, 224–236.
7. P. Frisco, H.J. Hoogeboom: P systems with symport/antiport simulating counter automata. *Acta Informatica*, 41, 2-3 (2004), 145–170.
8. L. Kari, C. Martín-Vide, A. Păun: On the universality of P systems with minimal symport/antiport rules. In *Aspects of Molecular Computing* (N. Jonoska, Gh. Păun, G. Rozenberg, eds.), LNCS 2950 (2004), 254–265.
9. 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.
10. M.L. Minsky: *Computations: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffts, New Jersey, 1967.
11. A. Păun, G. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3 (2002), 295–305.
12. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 1, 61 (2000), 108–143. Also TUCS Report No. 208, 1998.
13. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.
14. Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graphs of restricted forms. *Publicationes Mathematicae*, 60 (2002), 635–660.
15. Gh. Păun, T. Yokomori: Membrane computing based on splicing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, 54 (1999), 217–232.

16. G. Vaszil: On the size of P systems with minimal symport/antiport. In *Preliminary proceedings of WMC 2004, Workshop on Membrane Computing, Milano, Italy, June, 14-16, 2004* (G. Mauri, Gh. Păun, C. Zandron, eds.), Universitá di Milano-Bicocca, 2004, 422–431.
17. S. Verlan: Tissue P systems with minimal symport/antiport. In *Proceedings of Developments in Language Theory: 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004* (C.S. Calude, E. Calude, M.J. Dinneen, eds.), LNCS 2240 (2004), 418–430.
18. The P systems web page: `http://psystems.disco.unimib.it/`.