Computational Power of Symport/Antiport: History, Advances, and Open Problems

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

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

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

[3] Faculty of Informatics, Vienna University of Technology,
Favoritenstr. 9–11, A–1040 Vienna, Austria
rudi@emcc.at

Abstract. We first give a historical overview of the most important results obtained in the area of P systems and tissue P systems with *symport/antiport* rules, especially with respect to the development of computational completeness results improving descriptional complexity parameters. We consider the number of membranes (cells in tissue P systems), the weight of the rules, and the number of objects. Then we establish our newest results: P systems with only one membrane, symport rules of weight three, and with only seven additional objects remaining in the skin membrane at the end of a halting computation are computationally complete; P systems with minimal cooperation, i.e., P systems with symport/antiport rules of size one and P systems with symport rules of weight two, are computationally complete with only two membranes with only three and six, respectively, superfluous objects remaining in the output membrane at the end of a halting computation.

1 Introduction

P systems with *symport/antiport* rules, i.e., P systems with *pure communication rules assigned to membranes*, were introduced in [38]. 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 [15], [22], [17].

After establishing the necessary definitions, we first give a historical overview of the most important results obtained in the area of P systems and tissue P systems with *symport/antiport* rules and review the development of computational completeness results improving descriptional complexity parameters, especially concerning the number of membranes and cells, respectively, and the weight of

R. Freund et al. (Eds.): WMC 2005, LNCS 3850, pp. 1–30, 2006.

the rules as well as the number of objects. Moreover, we establish our newest results: first we prove that P systems with only one membrane and symport rules of weight three can generate any Turing computable set of numbers with only seven additional symbols remaining in the skin membrane at the end of a halting computation, which improves the result of [21] where thirteen superfluous symbols remained. Then we show that P systems with minimal cooperation, i.e., P systems with symport/antiport rules of weight one and P systems with symport rules of weight two, are computationally complete with only two membranes modulo some initial segment. In P systems with symport/antiport rules of weight one, only three superfluous objects remain in the output membrane at the end of a halting computation, whereas in P systems with symport rules of weight two six additional objects remain. For both variants, in [5] it has been shown that two membranes are enough to obtain computational completeness modulo a terminal alphabet; in this paper, we now show that the use of a terminal alphabet can be avoided for the price of superfluous objects remaining in the output membrane at the end of a halting computation. So far we were not able to completely avoid these additional objects, hence, it remains as an interesting question how to reduce their number.

2 Basic Notions and Definitions

For the basic elements of formal language theory needed in the following, we refer to [45]. 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 $\mathbb{N}_k RE$ we denote the family of recursively enumerable sets of natural numbers excluding the initial segment 0 to $k-1$. Equivalently, $\mathbb{N}_k RE = \{k + L \mid L \in \mathbb{N}RE\}$, where $k + L = \{k + n \mid n \in L\}$.

Let $\{a_1, \ldots, a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $|x|_{a_i}$; the *Parikh vector* associated with x with respect to $a_1, \ldots, a_n$ is $\left(|x|_{a_1}, \ldots, |x|_{a_n}\right)$. The *Parikh image* of a language L over $\{a_1, \ldots, a_n\}$ is the set of all Parikh vectors of strings in L. A (finite) multiset $\langle m_1, a_1\rangle \ldots \langle m_n, a_n\rangle$ with $m_i \in \mathbb{N}$, $1 \leq i \leq n$, can be represented by any string x the Parikh vector of which with respect to $a_1, \ldots, a_n$ is $(m_1, \ldots, m_n)$.

The family of recursively enumerable sets of vectors of natural numbers is denoted by $PsRE$.

2.1 Register Machines and Counter Automata

The proofs of the main results discussed in this paper are based on the simulation of register machines or counter automata, respectively; with respect to register machines, we refer to [37] for original definitions, and to [13] for definitions like those we use in this paper.

A (non-deterministic) *register machine* is a construct

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

where:

- d is the number of registers,
- Q is a finite set of label for the instructions of M,
- q_0 is the initial label,
- q_f is the final label, and
- P is a finite set of instructions injectively labelled with elements from Q. The labelled instructions are of the following forms:
  1. $q_1 : (A(r), q_2, q_3)$;
     add 1 to the contents of register r and proceed to one of the instructions (labelled with) q_2 and q_3 ("ADD"-instruction).
  2. $q_1 : (S(r), q_2, q_3)$;
     if register r is not empty, then subtract 1 from its contents and go to instruction q_2, otherwise proceed to instruction q_3 ("SUBTRACT"-instruction).
  3. $q_f : halt$;
     stop the machine; the final label q_f is only assigned to this instruction.

A (non-deterministic) register machine M is said to generate a vector of natural numbers $(s_1, \dots, s_k)$ if, starting with the instruction with label q_0 and all registers containing the number 0, the machine stops (it reaches the instruction $q_f : halt$) with the first k registers containing the numbers $s_1, \dots, s_k$ (and all other registers being empty).

The register machines are known to be computationally complete, equal in power to (non-deterministic) Turing machines: they generate exactly the sets of vectors of natural numbers which can be generated by Turing machines, i.e., the family $PsRE$. More precisely, from the main result in [37] that the actions of a Turing machine can be simulated by a register machine with two registers (using a prime number encoding of the configuration of the Turing machine) we know that any recursively enumerable set of k-vectors of natural numbers can be generated by a register machine with $k+2$ registers where only "ADD"-instructions are needed for the first k registers.

A non-deterministic *counter automaton* is a construct

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

where:

- d is the number of counters, and we denote $D = \{1, \dots, d\}$;
- 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 $F = \{0, 1, \dots, f\}$,
- $q_0 \in Q$ is the initial state,
- $q_f \in Q$ is the final state, and
- P is a finite set of instructions of the following forms:

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 register 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. As for register machines, we know that any set of k-vectors of natural numbers from $PsRE$ can be generated by a counter automaton with $k+2$ counters where only "increment"-instructions are needed for the first k counters.

A special variant of counter automata uses a set C of pairs $\{i, j\}$ with $i, j \in Q$ and $i \neq j$. As a part of the semantics of the *counter automaton with conflicting counters* $M = (d, Q, q_0, q_f, P, C)$, the automaton stops without yielding a result whenever it reaches a configuration where, for any pair of conflicting counters, both are non-empty.

Given an arbitrary counter automaton, we can easily construct an equivalent counter automaton with conflicting counters: For every counter i which shall also be tested for zero, we add a conflicting counter $\bar{\imath}$; then we replace all "test for zero"-instructions $(l \to l', i = 0)$ by the sequence of instructions $(l \to l'', \bar{\imath}+)$, $(l'' \to l', \bar{\imath}-)$. Thus, in counter automata with conflicting counters we only use "increment"-instructions and "decrement"-instructions, whereas the "test for zero"-instructions are replaced by the special conflicting counters semantics.

Another special variant of a counter automaton is called *partially blind (multi) counter automaton* (or machine, [23]); we shall use the abbreviation PBCA for this restricted type of counter automata which consists of a finite number (we call the number m) of counters that can add one and subtract one, but cannot test for zero. If there is an attempt to decrement a zero counter, the system aborts and does not accept. The first k counters (for some $k \leq m$) are input counters. The system is started with some nonnegative integers $(n_1, \ldots, n_k)$ in the input counters and the other counters set to zero. The input tuple is accepted if the system reaches a halting state and all the counters are zero. Hence, the language accepted by a PBCA is the set of k-tuples of nonnegative integers accepted by the system.

Formally a PBCA is defined as $M = (m, B, l_0, l_h, R)$ where m is the number of partially blind counters in the system, B is the set of instruction labels, l_0 is the starting instruction, l_h is the halting instruction, and R is the set of labelled instructions. These labelled instructions in R are of the forms:

- $l_i : (ADD(r), l_j)$,
- $l_i : (SUB(r), l_j)$,
- $l_i : HALT$,

where l_i and l_j are instruction labels and r is the counter that should be added/ subtracted.

For notational convenience, we will denote the family of sets of tuples of natural numbers accepted by some PBCA as $aPBLIND$ and the family of sets of tuples of natural numbers accepted by PBCAs with m counters as m-$aPBLIND$.

A related model called *blind (multi)counter automaton* (or machine, see [23]) is a (multi)counter automaton that can add one and subtract one from a counter, but cannot test a counter for zero. The difference between this model and a partially blind counter automaton is that a blind counter automaton does not abort when a zero counter is decremented. Thus, the counter can store negative numbers. Again, an input is accepted if the computation reaches an accept state and all the counters are zero.

We note that blind counter automata are equivalent in power to reversal bounded counter automata [23] which are equivalent to semilinear sets [30]. Partially blind counter automata are strictly more powerful than blind counter automata [23].

We have defined a PBCA as an acceptor for k-tuples of nonnegative integers. One can also define a partially blind counter automaton that is used as a generator of k-tuples of nonnegative integers [29]. A *partially blind counter generator* (PBCG) M consists of m counters, where the first $k \leq m$ counters are distinguished as the *output counters*. M starts with all counters set to zero. Again, at each step, each counter can be incremented/decremented by 1 (or left unchanged), but if there is an attempt to decrement a zero counter, the system aborts and does not generate anything. If the system halts in a final state with zero in counters $k+1, \ldots, m$, then the tuple $(n_1, \ldots, n_k)$ in the first k counters is said to be generated by M.

A restricted variant of a counter automaton is called *linear-bounded multicounter automaton* (or machine).

A *deterministic* multicounter automaton Z is linear-bounded if, when given an input n in one of its counters (called the input counter) and zeros in the other counters, it computes in such a way that the sum of the values of the counters at any time during the computation is at most n. One can easily normalize the computation so that every increment is preceded by a decrement (i.e., if Z

wants to increment a counter C_j, it first decrements some counter C_i and then increments C_j) and every decrement is followed by an increment. Thus we can assume that every instruction of Z, which is not "Halt", is of the form:

p: If $C_i \neq 0$, decrement C_i by 1, increment C_j by 1,
and go to k else go to state l

where p, k, l are labels (states). We do not require that the contents of the counters is zero when the automaton halts.

If in the instruction as defined above there is a "choice" for states k and/or l, then the automaton is called *non-deterministic*.

2.2 P Systems with Symport/Antiport Rules

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [40]; comprehensive information can be found on the P systems web page `http://psystems.disco.unimib.it`.

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

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

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 $1, 2, \dots, k$;
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 an elementary 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 (only the terminal objects as it done in [5], if in addition we specify a subset of O as the set of terminal symbols) 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 (terminal) object is counted separately, then a vector of natural numbers is obtained, denoted by $Ps(\Pi)$, see [40]. For short, we shall also speak of a *P system* only when dealing with a *P system with symport/antiport rules* as defined above.

By

$$\mathbb{N}O_nP_m(sym_s, anti_t)$$

we denote the family of sets of natural numbers (non-negative integers) that are generated by a P system with symport/antiport rules having at most $n > 0$ objects in O, at least $m > 0$ membranes, symport rules of size at most $s \geq 0$, and antiport rules of size at most $t \geq 0$. By

$$\mathbb{N}_kO_nP_m(sym_s, anti_t)$$

we denote the corresponding families of recursively enumerable sets of natural numbers without initial segment $\{0, 1, \ldots, k-1\}$. If we replace numbers by vectors, then in the notations above $\mathbb{N}$ is replaced by Ps. When any of the parameters m, n, s, t is not bounded, it is replaced by $*$; if the number of objects n is unbounded, we also may just omit n. If $s = 0$, then we may even omit sym_s; if $t = 0$, then we may even omit $anti_t$.

It may happen that P systems with symport/antiport (symport) rules can simulate deterministic register machines (i.e., register machines where in each ADD-instruction $q_1 : (A(r), q_2, q_3)$ the labels q_2 and q_3 are equal) in a deterministic way, i.e., from each configuration of the P system we can derive at most one other configuration. Then, when considering these P systems as accepting devices (the input from a set in $PsRE$ is put as an additional multiset into some specified membrane of the P system), we can get deterministic accepting P systems; the corresponding families of recursively enumerable sets of natural numbers then are denoted in the same way as before, but with the prefix aD; e.g., from the results proved in [18] and [14] we immediately obtain

$$PsRE = aDPsOP_1(anti_2).$$

Sometimes, the results we recall use the intersection with a terminal alphabet, in that way avoiding superfluous symbols to be counted as a result of a halting computation. In that case, we add the suffix $_T$ at the end of the corresponding notation.

2.3 Tissue P Systems with Symport/Antiport Rules

Tissue P systems were introduced in [34], and tissue-like P systems with channel states were investigated in [19]. Here we deal with the following type of systems (omitting the channel states).

A tissue P system (of degree $m \geq 1$) with symport/antiport rules is a construct

$$\Pi = \left(m, O, w_1, \ldots, w_m, ch, \left(R_{(i,j)}\right)_{(i,j)\in ch}\right),$$

where:

- m is the number of cells,
- O is the alphabet of *objects*,
- $w_1, \ldots, w_m$ are strings over O representing the *initial* multisets of *objects* present in the cells of the system (it is assumed that the m cells are labelled with $1, 2, \ldots, m$) and, moreover, we assume that all objects from O appear in an unbounded number in the environment,
- $ch \subseteq \{(i,j) \mid i,j \in \{0,1,2,\ldots,m\}, (i,j) \neq (0,0)\}$ is the set of links (*channels*) between cells (these were called *synapses* in [19]; 0 indicates the environment), $R_{(i,j)}$ is a finite set of symport/antiport rules associated with the channel $(i,j) \in ch$.

A *symport/antiport rule* of the form y/λ, λ/x, or y/x, respectively, $x, y \in O^+$, from $R_{(i,j)}$ for the ordered pair (i,j) of cells means moving the objects specified by y from cell i (from the environment, if $i = 0$) to cell j, at the same time moving the objects specified by x in the opposite direction. For short, we shall also speak of a *tissue P system* only when dealing with a *tissue P system with symport/antiport rules* as defined above.

The computation starts with the multisets specified by $w_1, \ldots, w_m$ in the m cells; in each time unit, a rule is used on each channel for which a rule can be used (if no rule is applicable for a channel, then no object passes over it). Therefore, the use of rules is sequential at the level of each channel, but it is parallel at the level of the system: all channels which can use a rule must do it (the system is synchronously evolving). The computation is successful if and only if it halts.

The result of a halting computation is the number described by the multiplicity of objects present in cell 1 (or in the first k cells) in the halting configuration. The set of all (vectors of) natural numbers computed in this way by the system Π is denoted by $N(\Pi)$ (resp., $Ps(\Pi)$). The family of sets $N(\Pi)$ $(Ps(\Pi))$ of (vectors of) natural numbers computed as above by systems with at most $n > 0$ symbols and $m > 0$ cells as well as with symport rules of weight $s \geq 0$ and antiport rules of weight $t \geq 0$ is denoted by

$$\mathbb{N}O_n t' P_m(sym_s, anti_t) \quad (\text{resp., } PsO_n t' P_m(sym_s, anti_t)).$$

When any of the parameters m, n, s, t is not bounded, it is replaced by $*$.

In [19], only channels (i, j) with $i \neq j$ are allowed, and, moreover, for any i, j only one channel out of $\{(i, j), (j, i)\}$ is allowed, i.e., between two cells (or one cell and the environment) only one channel is allowed (this technical detail may influence considerably the computational power). The family of sets $N(\Pi)$ (resp., $Ps(\Pi)$) of (vectors of) natural numbers computed as above by systems with at most $n > 0$ symbols and $m > 0$ cells as well as with symport rules of weight $s \geq 0$ and antiport rules of weight $t \geq 0$ is denoted by

$$\mathbb{N}O_n tP_m(sym_s, anti_t) \ \ (\text{resp., } PsO_n tP_m(sym_s, anti_t)).$$

3 Descriptional Complexity – A Historic Overview

In this section we review the development of computational completeness results with respect to descriptional complexity parameters, especially concerning the number of membranes (cells in tissue P systems), the weight of the rules, and the number of objects.

3.1 Rules Involving More Than Two Objects

We first recall results where rules involving more than two objects are used. As it was shown in [38], two membranes are enough for getting computational completeness when rules involving at most four objects, moving up to two objects in each direction, are used, i.e.,

$$\mathbb{N}RE = \mathbb{N}OP_2(sym_2, anti_2).$$

Using antiport. The result stated above was independently improved in [15], [17], and [22] – one membrane is enough:

$$\mathbb{N}RE = \mathbb{N}OP_1(sym_1, anti_2).$$

In fact, only one symport rule is needed; this can be avoided for the price of one additional object in the output region:

$$\mathbb{N}_1RE = \mathbb{N}_1OP_1(anti_2).$$

It is worth mentioning that the only antiport rules used are those exchanging one object by two objects.

Using symport. The history of P systems with symport only is longer. In [33] the results

$$\mathbb{N}RE = \mathbb{N}OP_2(sym_5) = \mathbb{N}OP_3(sym_4) = \mathbb{N}OP_5(sym_3)$$

were proved, whereas in [21]

$$\mathbb{N}_{13}RE = \mathbb{N}_{13}OP_1(sym_3)$$

was shown; the additional symbols can be avoided if a second membrane is used:

$$\mathbb{N}RE = \mathbb{N}OP_2(sym_3).$$

In this paper we now will show that we can bound the number of additional symbols by 7:

$$\mathbb{N}_7RE = \mathbb{N}_7OP_1(sym_3).$$

Determinism. It is known that deterministic P systems with one membrane using only antiport rules of weight at most 2 (actually, only the rules exchanging one object for two objects are needed, see [18], [11]) or using only symport rules of weight at most 3 (see [18]) can accept all sets of vectors of natural numbers (in fact, this is only proved for sets of numbers, but the extension to sets of vectors is straightforward), i.e.,

$$PsRE = aDPsOP_1(anti_2) = aDPsOP_1(sym_3).$$

3.2 Minimal Cooperation

Already in [38] it was shown that

$$\mathbb{N}RE = \mathbb{N}OP_5(sym_2, anti_1),$$

i.e., five membranes are already enough when only rules involving two objects are used. However, both types of rules involving two objects are used: symport rules moving up to two objects in the same direction, and antiport rules moving two objects in different directions.

Minimal cooperation by antiport. We now consider P systems where symport rules move only one object and antiport rules move only two objects across the a membrane in different directions. The first proof of the computational completeness of such P systems can be found in [9]:

$$\mathbb{N}RE = \mathbb{N}OP_9(sym_1, anti_1),$$

i.e., these P systems have nine membranes. This first result was improved by reducing the number of membranes to six [31], five [10], and four [20, 32], and finally in [46] it was shown that

$$\mathbb{N}_5RE = \mathbb{N}_5OP_3(sym_1, anti_1),$$

i.e., three membranes are sufficient to generate all recursively enumerable sets of numbers (with five additional objects in the output membrane).

In [6], a stronger result was shown where the output membrane did not contain superfluous symbols:

$$PsRE = PsOP_3(sym_1, anti_1).$$

In [5] it was shown that even two membranes are enough to obtain computational completeness, yet only modulo a terminal alphabet:

$$PsRE = PsOP_2(sym_1, anti_1)_T.$$

In this paper we now will show that we can bound the number of additional symbols by 3:

$$\mathbb{N}_3RE = \mathbb{N}_3OP_2(sym_1, anti_1).$$

Minimal cooperation by symport. We now consider P systems moving only one or two objects by a symport rule; these systems were shown to be computationally complete with four membranes in [22]:

$$\mathbb{N}RE = \mathbb{N}OP_4(sym_2).$$

In [6], this result was improved down to three membranes even for vectors of natural numbers:

$$PsRE = PsOP_3(sym_2).$$

Moreover, in [6] it was also shown that even two membranes are enough to obtain computational completeness (modulo a terminal alphabet):

$$PsRE = PsOP_2(sym_2)_T.$$

In this paper we will show that the number of additional objects in the output region can be bound by six:

$$\mathbb{N}_6RE = \mathbb{N}_6OP_2(sym_2).$$

The tissue case. If we do not restrict the graph of communication to be a tree, certain advantages appear. It was shown in [48] that

$$\mathbb{N}RE = \mathbb{N}OtP_3(sym_1, anti_1),$$

i.e., three cells are enough when using symport/antiport rules of weight one. This result was improved in [8] to two cells, again without additional objects in the output cell, and an equivalent result holds if antiport rules of weight one are replaced by symport rules of weight two:

$$PsRE = PsOtP_2(sym_1, anti_1) = PsOtP_2(sym_2).$$

Moreover, it was shown in the same article that accepting can be done deterministically:

$$PsRE = aDPsOtP_2(sym_1, anti_1) = aDPsOtP_2(sym_2).$$

A nice aspect of the proof is that it not only holds true for P systems with channels operating sequentially (as it is usually defined for tissue P systems), but also for P systems with channels operating in a maximally parallel way (like in standard P systems, generalizing the region communication structure of P systems to the arbitrary graph structure of tissue P systems).

Below computational completeness. In [8], it was also shown that

$$\mathbb{N}OP_1(sym_1, anti_1) \cup \mathbb{N}OtP_1(sym_1, anti_1) \subseteq \mathbb{N}FIN.$$

Together with the counterpart results for symport systems,

$$\mathbb{N}OP_1(sym_2) \cup \mathbb{N}OtP_1(sym_2) \subseteq \mathbb{N}FIN$$

obtained in [21], this is enough to state the optimality of the computational completeness results for the two-membrane/two-cell systems.

The most interesting open questions remaining in the cases considered so far concern the possibility to reduce the number of extra objects in the output region in some of the results stated above.

3.3 Small Number of Objects

In the preceding subsections, a survey of computational completeness results depending on the number of *membranes* or *cells* and the *weights* of the rules has been given. We now follow another direction of descriptional complexity: we try to keep the number of *membranes* or *cells* and especially the number of *objects* small, yet on the other hand allow rules of unbounded weight.

P Systems. A quite surprising result was presented in [42]: using symport/antiport rules of unbounded weight, P systems with four membranes are computationally complete even when the alphabet contains only three symbols:

$$\mathbb{N}RE = \mathbb{N}O_3P_4(sym_*, anti_*).$$

Then it has been shown in [1] that

$$\mathbb{N}RE = \mathbb{N}O_5P_1(sym_*, anti_*),$$

i.e., for P systems with one membrane, even five objects are enough for getting computational completeness.

The original result was improved in [3]; in sum, the actual computational completeness results for P systems can be found there:

$$\begin{array}{c}\mathbb{N}RE = \mathbb{N}O_nP_m(sym_*, anti_*) = a\mathbb{N}O_nP_m(sym_*, anti_*) \\ \text{for } (n,m) \in \{(5,1),(4,2),(3,3),(2,4)\}.\end{array}$$

The results mentioned above are presented as part of a general picture ("complexity carpet"), including results for generating/accepting/computing functions on vectors of specified dimensions.

Below computational completeness. The same article ([3]) presents undecidability results for the families

$$(a)\mathbb{N}O_2P_3(sym_*, anti_*),\ (a)\mathbb{N}O_3P_2(sym_*, anti_*),\ (a)\mathbb{N}O_4P_1(sym_*, anti_*);$$

moreover, it was shown that

$$\begin{array}{l}\mathbb{N}O_1P_2(sym_*, anti_*) \cap \mathbb{N}O_2P_1(sym_*, anti_*) \supseteq \mathbb{N}REG; \\ a\mathbb{N}O_3P_1(sym_*, anti_*) \cap a\mathbb{N}O_2P_2(sym_*, anti_*) \supseteq \mathbb{N}REG; \\ \mathbb{N}O_1P_1(sym_*, anti_*) = \mathbb{N}FIN; \\ a\mathbb{N}O_2P_1(sym_*, anti_*) \supseteq \mathbb{N}FIN.\end{array}$$

The last result has been improved in [29]; in the same article, also some results on one-symbol P systems are presented:

$$\begin{array}{l}a\mathbb{N}O_2P_1(sym_*, anti_*) \supsetneq \mathbb{N}REG; \\ a\mathbb{N}O_1P_{5m+3}(sym_*, anti_*) \supsetneq am\text{-}PBLIND; \\ \mathbb{N}O_1P_{5m+3}(sym_*, anti_*) \supseteq m\text{-}PBLIND.\end{array}$$

The parameter $5m+3$ in the last two results can even be reduced to $2m+3$, i.e., $2m+3$ membranes are enough to simulate partially blind counter automata/generators (these results will appear in the final version of [29].

Several questions are still open; the most interesting one is to determine the computational power of P systems with one symbol (we conjecture that they are not computationally complete, even if we can use an unbounded number of membranes and symport/antiport rules of unbounded weight).

Tissue P Systems. The question concerning systems with only one object has been answered in a positive way in [16] for tissue P systems:

$$\mathbb{N}RE = \mathbb{N}O_1tP_7(sym_*, anti_*) = \mathbb{N}O_1t'P_6(sym_*, anti_*).$$

In [2] the "complexity carpet" for tissue P systems was completed:

$$\mathbb{N}RE = \mathbb{N}O_ntP_m(sym_*, anti_*) \\ \text{for } (n,m) \in \{(4,2),(2,3),(1,7)\},$$

but

$$\mathbb{N}REG = \mathbb{N}O_*tP_1(sym_*, anti_*) = \mathbb{N}O_2tP_1(sym_*, anti_*)$$

and

$$\mathbb{N}FIN = \mathbb{N}O_1tP_1(sym_*, anti_*) = \mathbb{N}O_1t'P_1(sym_*, anti_*).$$

Using two channels between a cell and the environment, one cell can sometimes be saved, and one-cell systems become computationally complete:

$$\mathbb{N}RE = \mathbb{N}O_nt'P_m(sym_*, anti_*) \\ \text{for } (n,m) \in \{(5,1),(3,2),(2,3),(1,6)\}.$$

3.4 Computational Completeness - Summary

We now finish our historical review with repeating (some of) the best known results of computational completeness:

One membrane:
$aDPsOP_1(anti_2) = aDPsOP_1(sym_3) = PsRE$,
$\mathbb{N}_1RE = \mathbb{N}_1OP_1(anti_2)$,
$\mathbb{N}_7RE = \mathbb{N}_7OP_1(sym_3)$.

P systems - minimal cooperation:
$PsRE = PsOP_2(sym_1, anti_1)_T = PsOP_2(sym_2)_T$,
$\mathbb{N}_3RE = \mathbb{N}_3OP_2(sym_1, anti_1)$,
$\mathbb{N}_6RE = \mathbb{N}_6OP_2(sym_2)$.

Tissue P systems – minimal cooperation:
$PsRE = aDPsOtP_2(sym_1, anti_1) = aDPsOtP_2(sym_2)$,
$PsRE = PsOtP_2(sym_1, anti_1) = PsOtP_2(sym_2)$.

P systems – small number of objects:
$\mathbb{N}RE = \mathbb{N}O_nP_m(sym_*, anti_*)$
for $(n,m) \in \{(5,1),(4,2),(3,3),(2,4)\}$.

Tissue P systems – small number of objects:

$$\mathbb{N}RE = \mathbb{N}O_n tP_m(sym_*, anti_*) \quad \text{for } (n,m) \in \{(4,2),(2,3),(1,7)\},$$

$$\mathbb{N}RE = \mathbb{N}O_n t'P_m(sym_*, anti_*) \quad \text{for } (n,m) \in \{(5,1),(3,2),(2,3),(1,6)\}.$$

3.5 Bounded Symport/Antiport Systems

The question whether or not the deterministic version is weaker than the non-deterministic version of a specific variant of (tissue)P systems is an interesting and fundamental research issue in membrane computing, in particular for P systems with symport/antiport rules (see [41], [18], [26]).

Let us consider P systems that are used as acceptors. A symport/antiport P systems is called *bounded* if the only rules allowed are of the form $(u, out; v, in)$ such that u, v are multisets of objects with the restriction that $|u| = |v|$. (Note that all the rules are antiport rules). The power of these systems is exactly equivalent to that of linear-bounded (multi)counter automata or $log\ (n)$ space-bounded Turing machines (see [27]).

The deterministic and non-deterministic versions of such systems are equivalent if and only if deterministic and non-deterministic linear-bounded automata are equivalent, the latter problem being a long-standing open problem in complexity theory (see [27, 28]). This is in contrast to the fact that deterministic and non-deterministic 1-membrane unrestricted symport/antiport systems are equivalent and are universal (see, for example, Subsection 3.1 of this paper).

4 New Results

We first improve the result $\mathbb{N}_{13}OP_1(sym_3) = \mathbb{N}_{13}RE$ from [21]. For the proof, we use the variant of counter automata with conflicting counters and implement the semantics that if two conflicting counters are non-empty at the same time, then the computation is blocked without producing a result.

Theorem 1. *$\mathbb{N}_7OP_1(sym_3) = \mathbb{N}_7RE$.*

Proof. Let L be an arbitrary set from $\mathbb{N}_7RE$ and consider a counter automaton $M = (d, Q, q_0, q_f, P, C)$ with conflicting counters generating $L - 7$ $(= \{n - 7 \mid n \in L\})$; C is a finite set of pair sets of conflicting counters $\{i, \bar{\imath}\}$. We construct a P system simulating M:

$$\begin{aligned}
\Pi &= (O, E, [_1\ \ ]_1, w_1, R_1, 1),\\
O &= \{x_i \mid 1 \le i \le 6\} \cup Q \cup \{(p,j) \mid p \in P, 1 \le j \le 6\}\\
&\quad \cup \{a_i, A_i \mid i \in C\} \cup \{\#, b, d\},\\
E &= \{a_i, A_i \mid i \in C\} \cup \{x_2, x_3, \#\}\\
&\quad \cup Q \cup \{(p,j) \mid p \in P, j \in \{2,4,5,6\}\},\\
w_1 &= l_0 d x_1 x_4 x_5 x_6 \prod_{p \in P} (p,1)\,(p,3)\, b.
\end{aligned}$$

The following rules allow us to simulate the counter automaton M:

- The rules $(da_ia_{\bar{\imath}}, out)$ implement the special semantics of conflicting counters $\{i, \bar{\imath}\}$ with leading to an infinite computation by applying the rules $(d\#, out)$ and $(d\#, in)$.
- The simulation of the instructions of M is initiated by also sending out x_1 in the first step; the rules $(x_1x_2x_3, in)$ as well as $(x_2x_4x_5, out)$ and (x_3x_6, out) then allow us to send out the specific signal variables x_4, x_5, and x_6 which are needed to guide the sequence of rules to be applied.
- The instruction $p : (l \to l', i-)$ is simulated by the sequence of rules

$$\begin{array}{l}(l(p,1)x_1, out),\\ ((p,1)x_4(p,2), in),\\ ((p,2)(p,3)a_i, out),\ ((p,2)(p,3)d, out),\\ ((p,3)x_5(p,4), in),\\ ((p,4)(p,5), out),\\ ((p,5)x_6l', in).\end{array}$$

  In case that no symbol a_i is present (which corresponds to the fact that counter i is empty), the rule $((p,2)(p,3)d, out)$ leads to an infinite computation by applying the rules $(d\#, out)$ and $(d\#, in)$. Otherwise, decrementing is successfully accomplished by applying the rule $((p,2)(p,3)a_i, out)$.
- The instruction $p : (l \to l', i+)$ is simulated by the sequence of rules

$$\begin{array}{l}(l(p,1)x_1, out),\\ ((p,1)x_4(p,2), in),\\ ((p,2)(p,3)A_i, out),\\ ((p,3)x_5l', in),\\ (A_ix_6a_i, in).\end{array}$$

  The symbol A_i is sent out to take exactly one symbol a_i in.
- A simulation of M by Π terminates with sending out the symbols from $\{(p,1), (p,3) \mid p \in P\} \cup \{A_i \mid i \in C\}$ which were used during the simulation of the instructions of M as soon as the halting label l_h of M appears:
  (l_hbx, out),
  $x \in \{(p,1), (p,3) \mid p \in P\} \cup \{A_i \mid i \in C\}$,
  (l_hb, in).
  If the system halts, the objects inside correspond with the contents of the output registers, and the extra symbols are $l_h, d, b, x_1, x_4, x_5, x_6$, i.e., seven in total. □

We now show that two membranes are enough to obtain computational completeness with symport/antiport rules of minimal size 1 with only three additional objects remaining in halting computations.

Theorem 2. $\mathbb{N}_3 OP_2(sym_1, anti_1) = \mathbb{N}_3 RE$.

Proof. We simulate a counter automaton $M = (d, Q, q_0, q_f, P)$ which starts with empty counters. We also suppose that all instructions from P are labelled in a one-to-one manner with elements of $\{1, \ldots, n\} = I$; 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. Additionally we suppose, without loss of generality, that on the first counter of the counter automaton M only "increment" instructions – of the form $(q_i \to q_l, c_1+)$ – are operating.

We construct the P system Π_1 as follows:

$$\begin{aligned}
\Pi_1 &= (O, [_1\ [_2\ \ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2),\\
O &= E \cup \{I_c, q_0', F_1, F_2, F_3, F_4, F_5, \#_1, \#_2, b_j, b_j' \mid j \in I\},\\
E &= Q \cup \{a_j, a_j', a_j'' \mid j \in I\} \cup C \cup \{F_2, F_3, F_4, F_5\},\\
w_1 &= q_0' I_c \#_1 \#_1 \#_2 \#_2,\\
w_2 &= F_1 F_1 F_1 \prod_{j \in I} b_j \prod_{j \in I} b_j',\\
R_i &= R_{i,s} \cup R_{i,r} \cup R_{i,f}, \quad i = 1, 2.
\end{aligned}$$

The functioning of this system may be split into two stages:

1. simulating the instructions of the counter automaton;
2. terminating the computation.

We code the counter automaton as follows:

Region 1 will hold the current state of the automaton, represented by a symbol $q_i \in Q$; region 2 will hold the value of all counters, represented by the number of occurrences of symbols $c_k \in C$, $k \in D$, where $D = \{1, \ldots, d\}$. We also use the following idea realized by the phase "START" below: from the environment, we bring symbols c_k into region 1 all the time during the computation. This process may only be stopped if all stages finish correctly; otherwise, the computation will never stop.

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 (stage 1);
2. RUN (stage 1);
3. END (stage 2).

The parts of the computations illustrated in the following describe different stages of the evolution of the P system given in the corresponding theorem. For simplicity, we focus on explaining a particular stage and omit the objects that do not participate in the evolution at that time. Each rectangle represents a membrane, each variable represents a copy of 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.

$$\begin{aligned} R_{1,s} &= \{\texttt{1s1} : (I_c, in),\ \ \texttt{1s2} : (I_c, out; c_k, in),\ \ \texttt{1s3} : (c_k, out) \mid c_k \in C\} \\ &\quad \cup \{\texttt{1s4} : (q_0', out; q_0, in)\}, \\ R_{2,s} &= \emptyset \end{aligned}$$

Symbol I_c brings one symbol c_k from the environment into region 1 (rules `1s1`, `1s2`), where it may be used immediately during the simulation of the "increment" instruction and then moved to region 2. Otherwise symbol c_k returns to the environment (rule `1s3`). Rule `1s4` is used for synchronizing the appearance of the symbols c_k and q_i in region 1.

We illustrate the beginning of the computation as follows:

$$\mathbf{c_{k_1}}\mathbf{q_0}a_j c_{k_2}\boxed{\mathbf{q_0'}\mathbf{I_c}\boxed{b_j}} \Rightarrow^{\texttt{1s2,1s4}} \mathbf{I_c}q_0'\mathbf{a_j}c_{k_2}\boxed{\mathbf{q_0}\mathbf{c_{k_1}}\boxed{b_j}} \Rightarrow^{\texttt{1s1,1s3,1r1}}$$
$$q_0' q_0 c_{k_1}\mathbf{c_{k_2}}\boxed{\mathbf{a_j}\mathbf{I_c}\boxed{\mathbf{b_j}}} \Rightarrow^{\texttt{1s2,2r1}} q_0' q_0 c_{k_1}\mathbf{I_c}\boxed{c_{k_2}\mathbf{b_j}\boxed{\mathbf{a_j}}} \cdots$$

2. RUN.

$$\begin{aligned} R_{1,r} &= \{\texttt{1r1} : (q_i, out; a_j, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, =0\}\} \\ &\quad \cup \{\texttt{1r2} : (b_j, out; a_j', in),\ \ \texttt{1r3} : (a_j, out; b_j, in), \\ &\qquad \texttt{1r4} : (\#_1, out; b_j, in) \mid j \in I\} \\ &\quad \cup \{\texttt{1r5} : (a_j', out; a_j'', in) \mid j \in I_+ \cup I_-\} \cup \{\texttt{1r6} : (\#_1, out; \#_1, in)\} \\ &\quad \cup \{\texttt{1r7} : (b_j', out; a_j'', in),\ \ \texttt{1r8} : (a_j', out; b_j', in), \\ &\qquad \texttt{1r9} : (\#_1, out; b_j', in) \mid j \in I_{=0}\} \\ &\quad \cup \{\texttt{1r10} : (a_j'', out, q_l, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, =0\}\} \\ &\quad \cup \{\texttt{1r11} : (b_j, out),\ \ \texttt{1r12} : (b_j', out) \mid j \in I\}, \\ R_{2,r} &= \{\texttt{2r1} : (b_j, out; a_j, in) \mid j \in I\} \\ &\quad \cup \{\texttt{2r2} : (a_j, out; c_k, in) \mid (j : q_i \to q_l, c_k+) \in P\} \\ &\quad \cup \{\texttt{2r3} : (a_j', in) \mid j \in I_+\} \\ &\quad \cup \{\texttt{2r4} : (a_j', out; b_j, in) \mid j \in I_+ \cup I_-\} \\ &\quad \cup \{\texttt{2r5} : (a_j, out) \mid j \in I_- \cup I_{=0}\} \\ &\quad \cup \{\texttt{2r6} : (c_k, out; a_j', in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{-, =0\}\} \\ &\quad \cup \{\texttt{2r7} : (b_j', out; b_j, in),\ \ \texttt{2r8} : (b_j', in) \mid j \in I_{=0}\} \\ &\quad \cup \{\texttt{2r9} : (a_j, out; \#_2, in) \mid j \in I_+\} \cup \{\texttt{2r10} : (\#_2, out; \#_2, in)\}. \end{aligned}$$

"Increment"-instruction:

$$\mathbf{a_j} a_j' a_j'' q_l \boxed{\mathbf{q_i} c_k \#_1 \#_1 \boxed{b_j}} \Rightarrow^{\mathtt{1r1}} a_j' a_j'' q_i q_l \boxed{\mathbf{a_j} c_k \#_1 \#_1 \boxed{\mathbf{b_j}}} \Rightarrow^{\mathtt{2r1}}$$

$$\mathbf{a_j'} a_j'' q_i q_l \boxed{\mathbf{b_j c_k} \#_1 \#_1 \boxed{\mathbf{a_j}}} \Rightarrow^{\mathtt{1r2,2r2}} \mathbf{b_j} a_j'' q_i q_l \boxed{\mathbf{a_j a_j'} \#_1 \#_1 \boxed{c_k}}$$

Now there are two possibilities: we may either apply
a) rule `1r5` or
b) rule `2r3`.

It is easy to see that **case a)** leads to an infinite computation:

$$\mathbf{b_j a_j''} q_i q_l \boxed{\mathbf{a_j a_j'} \#_1 \#_1 \boxed{c_k}} \Rightarrow^{\mathtt{1r5,1r3}}$$

$$a_j \mathbf{a_j'} q_i \mathbf{q_l} \boxed{\mathbf{b_j a_j''} \#_1 \#_1 \boxed{c_k}} \Rightarrow^{\mathtt{1r2,1r10}} a_j \mathbf{b_j} q_i \mathbf{a_j''} \boxed{\mathbf{a_j' q_l \#_1} \#_1 \boxed{c_k}}$$

After that rule `1r4` will eventually be applied, object $\#_1$ will be moved to the environment and then applying rule `1r6` leads to an infinite computation.

Now let us consider **case b)**:

$$\mathbf{b_j} a_j'' q_i q_l \boxed{\mathbf{a_j a_j'} \#_1 \#_1 \boxed{c_k}} \Rightarrow^{\mathtt{1r3,2r3}} a_j a_j'' q_i q_l \boxed{\mathbf{b_j} \#_1 \#_1 \boxed{\mathbf{a_j'} c_k}}$$

We cannot apply rule `1r2` as this leads to an infinite computation (see above). Hence, rule `2r4` has to be applied:

$$a_j a_j'' q_i q_l \boxed{\mathbf{b_j} \#_1 \#_1 \boxed{\mathbf{a_j'} c_k}} \Rightarrow^{\mathtt{2r4}} a_j \mathbf{a_j''} q_i q_l \boxed{\mathbf{a_j'} \#_1 \#_1 \boxed{b_j c_k}} \Rightarrow^{\mathtt{1r5}}$$

$$a_j a_j' q_i \mathbf{q_l} \boxed{\mathbf{a_j''} \#_1 \#_1 \boxed{b_j c_k}} \Rightarrow^{\mathtt{1r10}} a_j a_j' a_j'' q_i \boxed{\mathbf{q_l} \#_1 \#_1 \boxed{b_j c_k}}$$

In that way, q_i is replaced by q_l and c_k is moved from region 1 into region 2.

"Decrement"-instruction:

$$\mathbf{a_j} a_j' a_j'' q_l \boxed{\mathbf{q_i} \#_1 \#_1 \boxed{b_j c_k}} \Rightarrow^{\mathtt{1r1}} a_j' a_j'' q_i q_l \boxed{\mathbf{a_j} \#_1 \#_1 \boxed{\mathbf{b_j} c_k}} \Rightarrow^{\mathtt{2r1}}$$

$$\mathbf{a_j'} a_j'' q_i q_l \boxed{\mathbf{b_j} \#_1 \#_1 \boxed{\mathbf{a_j} c_k}} \Rightarrow^{\mathtt{1r2,2r5}} \mathbf{b_j} a_j'' q_i q_l \boxed{\mathbf{a_j a_j'} \#_1 \#_1 \boxed{c_k}} \Rightarrow^{\mathtt{1r3,2r6}}$$

$$a_j a_j'' q_i q_l \boxed{\mathbf{b_j} c_k \#_1 \#_1 \boxed{\mathbf{a_j'}}} \Rightarrow^{\mathtt{2r4}} a_j \mathbf{a_j''} q_i q_l \boxed{\mathbf{a_j'} c_k \#_1 \#_1 \boxed{b_j}} \Rightarrow^{\mathtt{1r5}}$$

$$a_j a_j' q_i \mathbf{q_l} \boxed{\mathbf{a_j''} c_k \#_1 \#_1 \boxed{b_j}} \Rightarrow^{\mathtt{1r10}} a_j a_j' a_j'' q_i \boxed{\mathbf{q_l} c_k \#_1 \#_1 \boxed{b_j}}$$

In the way described above, q_i is replaced by q_l and c_k is removed from region 2 to region 1.

“Test for zero”-instruction:

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

(i) *There is no c_k in region 2:*

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

Now there are two possibilities: we apply either

a) rule `2r7` or

b) rule `1r2`.

It is easy to see that **case a)** leads to an infinite computation:

$$a'_j a''_j q_i q_l \boxed{\mathbf{b_j} \#_1 \#_1 \boxed{\mathbf{a_j} \mathbf{b'_j}}} \Rightarrow^{\texttt{2r7,2r5}} a'_j a''_j q_i q_l \boxed{\mathbf{a_j} \mathbf{b'_j} \#_1 \#_1 \boxed{\mathbf{b_j}}} \Rightarrow^{\texttt{2r1,2r8}}$$
$$a'_j a''_j q_i q_l \boxed{\mathbf{b_j} \#_1 \#_1 \boxed{\mathbf{a_j} \mathbf{b'_j}}} \Rightarrow^{\texttt{2r7,2r5}} \dots \Rightarrow^{\texttt{2r1,2r8}} \mathbf{a'_j} a''_j q_i q_l \boxed{\mathbf{b_j} \#_1 \#_1 \boxed{\mathbf{a_j} b'_j}}$$
$$\Rightarrow^{\texttt{1r2,2r5}} \mathbf{b_j} a''_j q_i q_l \boxed{\mathbf{a_j} a'_j \#_1 \#_1 \boxed{b'_j}} \Rightarrow^{\texttt{1r3}} a_j a''_j q_i q_l \boxed{\mathbf{b_j} a'_j \#_1 \#_1 \boxed{b'_j}}$$

Again there are two possibilities: we can apply either

c) rule `1r2` or

d) rule `2r7`.

Case c) leads to an infinite computation (rules `1r4` and `1r6`).

Now let us consider **case d)**:

$$a_j a''_j q_i q_l \boxed{\mathbf{b_j} a'_j \#_1 \#_1 \boxed{\mathbf{b'_j}}} \Rightarrow^{\texttt{2r7}} a_j \mathbf{a''_j} q_i q_l \boxed{\mathbf{b'_j} a'_j \#_1 \#_1 \boxed{b_j}} \Rightarrow^{\texttt{1r7}}$$
$$a_j \mathbf{b'_j} q_i \mathbf{q_l} \boxed{\mathbf{a''_j} \mathbf{a'_j} \#_1 \#_1 \boxed{b_j}} \Rightarrow^{\texttt{1r8,1r10}} a_j a'_j a''_j q_i \boxed{\mathbf{q_l} \mathbf{b'_j} \#_1 \#_1 \boxed{b_j}}$$

There are two possibilities: we can apply either

e) rule `1r7` or

f) rule `2r8`.

Case e) leads to infinite computation (rules `1r9` and `1r6`).

In **case f)**, the object b'_j comes back to region 2.

(b) *There is some c_k in region 2:*

Consider again **case d)**:

$$a_j a''_j q_i q_l \boxed{\mathbf{b_j} \mathbf{a'_j} \#_1 \#_1 \boxed{\mathbf{b'_j} \mathbf{c_k}}} \Rightarrow^{\texttt{2r7,2r6}} a_j \mathbf{a''_j} q_i q_l \boxed{\mathbf{b'_j} c_k \#_1 \#_1 \boxed{a'_j b_j}} \Rightarrow^{\texttt{1r7}}$$
$$a_j \mathbf{b'_j} q_i \mathbf{q_l} \boxed{\mathbf{a''_j} c_k \#_1 \#_1 \boxed{a'_j b_j}} \Rightarrow^{\texttt{1r9,1r10}} a_j a''_j \#_1 q_i \boxed{\mathbf{q_l} \mathbf{b'_j} c_k \#_1 \boxed{a'_j b_j}}$$

Now the application of rule `1r6` leads to an infinite computation.

Finally, let us notice that applying the rules `1r11` and `1r12` during the phase RUN leads to infinite computation. Hence, we model correctly the "test for zero" instruction.

3. END.

$$\begin{aligned}
R_{1,f} = \{ & \mathtt{1f1} : (F_1, out; F_2, in), \quad \mathtt{1f2} : (F_2, out; F_3, in), \\
& \mathtt{1f3} : (F_3, out; F_4, in), \quad \mathtt{1f4} : (F_4, out; F_5, in)\}, \\
R_{2,f} = \{ & \mathtt{2f1} : (F_1, out; q_f, in), \quad \mathtt{2f2} : (q_f, out; I_c, in), \\
& \mathtt{2f3} : (q_f, out; \#_1, in), \quad \mathtt{2f4} : (q_f, out; \#_2, in), \quad \mathtt{2f5} : (F_5, out), \\
& \mathtt{2f6} : (b_j, out; F_5, in), \quad \mathtt{2f7} : (b'_j, out; F_5, in)\}.
\end{aligned}$$

We illustrate the end of computations as follows:

$$\begin{array}{l}
F_2F_3F_4F_5I_cc_{k_1}c_{k_2}\boxed{q_f\#_1\#_1\#_2\#_2\boxed{F_1F_1F_1b_{j_1}b'_{j_2}}} \Rightarrow^{\mathtt{2f1,1s1}} \\
F_2F_3F_4F_5c_{k_1}c_{k_2}\boxed{I_c\#_1\#_1\#_2\#_2F_1\boxed{q_fF_1F_1b_{j_1}b'_{j_2}}} \Rightarrow^{\mathtt{2f3,1s2,1f1}} \\
F_2F_3F_4F_5I_cc_{k_2}F_1\boxed{F_2c_{k_1}\#_1\#_2\#_2q_f\boxed{\#_1F_1F_1b_{j_1}b'_{j_2}}} \Rightarrow^{\mathtt{1s1,1s4,1f2,2f1}} \\
F_2F_3F_4F_5c_{k_1}c_{k_2}F_1\boxed{F_3I_c\#_1\#_2\#_2F_1\boxed{q_f\#_1F_1b_{j_1}b'_{j_2}}} \Rightarrow^{\mathtt{1s2,1f1,1f3,2f3}} \\
F_2F_3F_4F_5c_{k_1}I_cF_1F_1\boxed{F_2F_4c_{k_2}\#_2\#_2q_f\boxed{\#_1\#_1F_1b_{j_1}b'_{j_2}}} \Rightarrow^{\mathtt{1s1,1s4,1f2,1f4,2f1}} \\
F_2F_3F_4F_5c_{k_1}c_{k_2}F_1F_1\boxed{F_3F_5I_c\#_2\#_2F_1\boxed{q_f\#_1\#_1b_{j_1}b'_{j_2}}}
\end{array}$$

Notice that now rule `2f2` will eventually be applied, as otherwise the application of rule `2f4` will lead to an infinite computation (rule `2r10`). Hence, we continue as follows:

$$\begin{array}{l}
F_2F_3F_4F_5c_{k_1}c_{k_2}F_1F_1\boxed{F_3F_5I_c\#_2\#_2F_1\boxed{q_f\#_1\#_1b_{j_1}b'_{j_2}}} \Rightarrow^{\mathtt{1f1,1f3,2f2,2f6}} \\
F_2F_3F_4F_5c_{k_1}c_{k_2}F_1F_1F_1\boxed{F_2F_4\#_2\#_2b_{j_1}q_f\boxed{I_c\#_1\#_1F_5b'_{j_2}}} \Rightarrow^{\mathtt{1f2,1f4,1r11,2f5}} \\
F_2F_3F_4F_5c_{k_1}c_{k_2}F_1F_1F_1b_{j_1}\boxed{F_3F_5F_5\#_2\#_2q_f\boxed{I_c\#_1\#_1b'_{j_2}}}
\end{array}$$

We continue in this manner until all objects $b_j, b'_j,\ j \in I$ from the elementary membrane 2 have been moved to the environment. Notice that the result in the elementary membrane 2 (multiset c_1^t) cannot be changed during phase END, as object I_c now is situated in the elementary membrane and cannot bring symbols c_1 from the environment. Recall that the counter automaton can only increment the first counter c_1, so all other computations of P system Π_1 cannot change the number of symbols c_1 in the elementary membrane. Thus, at the end of a terminating computation, in the elementary membrane there are the result (multiset c_1^t) and only the three additional objects $I_c, \#_1, \#_1$. $\square$

A "dual" class of systems with minimal cooperation is the class where two objects are moved across the membrane in the same direction rather than in the opposite ones. We now prove a similar result for this class using six additional symbols.

Theorem 3. $\mathbb{N}_6OP_2(sym_2) = \mathbb{N}_6RE$.

Proof. As in the proof of Theorem 1 we simulate a counter automaton $M = (d, Q, q_0, q_f, P)$ that 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. Moreover, we define $I' = \{1, 2, \ldots, n+4\}$, $Q_k = \{q_{i,k}\}$, $1 \le k \le 5$, $i \in K$, $K = \{0, 1, \ldots, f\}$, and $C = \{c_i \mid 1 \le i \le d\}$.

We construct the P system Π_2 as follows:

$$\Pi_2 = (O, [_1\ [_2\ \ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2),$$

$$O = \{\#_0, \#_1, \#_2, \$_1, \$_2, \$_3, \hat{a}, \hat{b}, I_c\} \cup \{a_k \mid 1 \le k \le 5\} \cup Q \bigcup_{1 \le k \le 5} Q_k$$

$$\cup\, C \cup \{a_j, a'_j, \check{a}_j, \hat{a}_j, b_j, d_j, d'_j, d''_j \mid j \in I\} \cup \{e_t, h_t \mid t \in I'\},$$

$$E = \{a_1, a_3, a_5, \#_0\} \cup \{a_j, a'_j \mid j \in I\} \cup \{h_t \mid t \in I'\} \cup Q \cup Q_2 \cup Q_4 \cup C,$$

$$w_1 = \#_1 \hat{a}\hat{b} a_2 a_4 \$_3 \prod_{j \in I} \check{a}_j \prod_{j \in I} d'_j \prod_{j \in I} d''_j \prod_{t \in I'} e_t \prod_{i \in K} \hat{q}_i \prod_{i \in K} q_{i,1} \prod_{i \in K} q_{i,3} \prod_{i \in K} q_{i,5},$$

$$w_2 = \#_2 \$_1^{n+1} \$_2 \prod_{j \in I} \hat{a}_j \prod_{j \in I} b_j \prod_{j \in I} d_j,$$

$$R_i = R_{i,s} \cup R_{i,r} \cup R_{i,f}, i \in \{1, 2\}.$$

The functioning of this system again may be split into two stages:

1. simulating the instructions of the counter automaton;
2. terminating the computation.

We code the counter automaton as in Theorem 1 above: region 1 will hold the current state of the automaton, represented by a symbol $q_i \in Q$; region 2 will hold the value of all counters, represented by the number of occurrences of symbols $c_k \in C$, $k \in D$, where $D = \{1, \ldots, d\}$. We also use the following idea (called "*circle*") realized by phase "START" below: from the environment, we bring symbols c_k into region 1 all the time during the computation. This process may only be stopped if all stages finish correctly; otherwise, the computation will never stop.

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 again are given by three phases:

1. START (stage 1);
2. RUN (stage 1);
3. END (stage 2).

1. START.

$$R_{1,s} = \{\mathtt{1s1} : (I_c, out),\quad \mathtt{1s2} : (I_c c_k, in),\quad \mathtt{1s3} : (c_k, out) \mid k \in D\},$$
$$R_{2,s} = \emptyset.$$

Symbol I_c brings one symbol $c \in C$ from the environment into region 1 (rules 1s1, 1s2) where it may be used immediately during the simulation of an "increment"-instruction and moved to region 2. Otherwise symbol c returns to the environment (rule 1s3).

2. RUN.

$$\begin{aligned}
R_{1,r} = &\{\mathtt{1r1} : (q_i\hat{q}_i, out) \mid i \in K\}\\
&\cup \{\mathtt{1r2} : (a_j\hat{q}_i, in) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{+, -, =0\}, k \in D\}\\
&\cup \{\mathtt{1r3} : (a_j\hat{a}, out) \mid j \in I_+ \cup I_-\} \cup \{\mathtt{1r4} : (a_j\hat{b}, out) \mid j \in I_{=0}\}\\
&\cup \{\mathtt{1r5} : (\#_2, out),\quad \mathtt{1r6} : (\#_2, in)\} \cup \{\mathtt{1r7} : (b_j\check{a}_j, out) \mid j \in I\}\\
&\cup \{\mathtt{1r8} : (b_j\#_1, out) \mid j \in I\} \cup \{\mathtt{1r9} : (\hat{a}_j\#_1, out) \mid j \in I\}\\
&\cup \{\mathtt{1r10} : (\#_0\#_1, in),\quad \mathtt{1r11} : (\#_0\hat{b}, in)\} \cup \{\mathtt{1r12} : (a'_jb_j, in) \mid j \in I\}\\
&\cup \{\mathtt{1r13} : (\hat{a}a_1, in),\quad \mathtt{1r14} : (a_1a_2, out),\quad \mathtt{1r15} : (a_2a_3, in)\}\\
&\cup \{\mathtt{1r16} : (a_3a_4, out),\quad \mathtt{1r17} : (a_4a_5, in),\quad \mathtt{1r18} : (a_5, out)\}\\
&\cup \{\mathtt{1r19} : (a'_jq_{l,1}, out) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{+, -, =0\}, k \in D\}\\
&\cup \{\mathtt{1r20} : (q_{i,1}q_{i,2}, in),\quad \mathtt{1r21} : (q_{i,2}q_{i,3}, out),\quad \mathtt{1r22} : (q_{i,3}q_{i,4}, in) \mid i \in K\}\\
&\cup \{\mathtt{1r23} : (q_{i,4}q_{i,5}, out),\quad \mathtt{1r24} : (q_{i,5}q_i, in) \mid i \in K\}\\
&\cup \{\mathtt{1r25} : (d_j\hat{a}, out),\quad \mathtt{1r26} : (d_j\#_0, in) \mid j \in I_+ \cup I_-\}\\
&\cup \{\mathtt{1r27} : (d_j\check{a}_j, in) \mid j \in I\} \cup \{\mathtt{1r28} : (d_j\#_1, out) \mid j \in I_+ \cup I_-\}\\
&\cup \{\mathtt{1r29} : (d_jd'_j, out) \mid j \in I_{=0}\} \cup \{\mathtt{1r30} : (d'_j\hat{b}, in) \mid j \in I_{=0}\},
\end{aligned}$$

$$\begin{aligned}
R_{2,r} = &\{\mathtt{2r1} : (a_j\check{a}_j, in) \mid j \in I\} \cup \{\mathtt{2r2} : (b_j\check{a}_j, out) \mid j \in I\}\\
&\cup \{\mathtt{2r3} : (a_jc_k, out) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{-, =0\}, k \in D\}\\
&\cup \{\mathtt{2r4} : (a_j\#_2, out) \mid j \in I_-\} \cup \{\mathtt{2r5} : (a_j\hat{a}_j, out) \mid j \in I_+\}\\
&\cup \{\mathtt{2r6} : (\#_0, in), \mathtt{2r7} : (\#_0, out)\}\\
&\cup \{\mathtt{2r8} : (c_k\hat{a}_j, in) \mid (j : q_i \to q_l, k+) \in P, k \in D\}\\
&\cup \{\mathtt{2r9} : (a'_jb_j, in) \mid j \in I\} \cup \{\mathtt{2r10} : (a'_jd_j, out) \mid j \in I\}\\
&\cup \{\mathtt{2r11} : (d_ja_5, in) \mid j \in I_+ \cup I_-\} \cup \{\mathtt{2r12} : (a_5, out)\}\\
&\cup \{\mathtt{2r13} : (d_jd''_j, in) \mid j \in I_{=0}\} \cup \{\mathtt{2r14} : (a_jd''_j, out) \mid j \in I_{=0}\}.
\end{aligned}$$

"Increment"-instruction:

$$a_j c \boxed{\mathbf{I_c q_i \hat{q}_i} \breve{a}_j \hat{a} \boxed{b_j \hat{a}_j}} \Rightarrow^{\mathtt{1r1,1s1}} q_i \mathbf{\hat{q}_i a_j I_c c} \boxed{\breve{a}_j \hat{a} \boxed{b_j \hat{a}_j}} \Rightarrow^{\mathtt{1r2,1s2}}$$

$$q_i \boxed{I_c c \hat{q}_i a_j \breve{a}_j \hat{a} \boxed{b_j \hat{a}_j}} \text{ where } c \in C$$

Now there are two variants of computations (depending on the application of rule `2r1` or rule `1r3`). It is easy to see that the application of rule `1r3` leads to an infinite computation (by *"circle"*). Consider applying rule `2r1`:

$$q_i c_k \boxed{\mathbf{I_c c} \hat{q}_i \mathbf{a_j \breve{a}_j} \hat{a} \boxed{b_j \hat{a}_j}} \Rightarrow^{\mathtt{2r1,1s1,1s3}}$$

$$q_i \mathbf{I_c c_k} c \boxed{\hat{q}_i \hat{a} \boxed{\mathbf{b_j \breve{a}_j a_j \hat{a}_j}}} \Rightarrow^{\mathtt{2r2,2r5,1s2}}$$

$$q_i c \boxed{I_c c_k \hat{q}_i \hat{a} b_j \breve{a}_j a_j \hat{a}_j \boxed{}}$$

Notice that object $\hat{a}_j$ cannot be idle, as the application of the rules `1r9`, `1r10`, `2r6`, `2r7` leads to an infinite computation. Hence, rule `2r8` will be applied and object c_k will be moved to region 2 (thus, we increase the number of objects c_k in region 2 by one and model the increment-instruction of the counter automaton). In an analogous way, object b_j cannot be idle, as applying rules `1r8`, `1r10`, `2r6`, `2r7` leads to an infinite computation. Thus, rule `2r1` cannot be applied and rule `1r7` will eventually be applied.

$$c a'_j a_1 a_3 a_5 \boxed{\mathbf{I_c c_k} \hat{q}_i \hat{a} \mathbf{b_j \breve{a}_j a_j \hat{a}_j} a_2 a_4 q_{l,1} \boxed{}}$$

$$\mathbf{I_c c a'_j b_j} \breve{a}_j a_j \mathbf{\hat{a} a_1} a_3 a_5 \boxed{\hat{q}_i a_2 a_4 q_{l,1} \boxed{\hat{a}_j c_k}} \Rightarrow^{\mathtt{1r12,1r13,1s2}}$$

$$\breve{a}_j a_j a_3 a_5 \boxed{I_c c \hat{q}_i \hat{a} a_1 a_2 a_4 q_{l,1} a'_j b_j \boxed{\hat{a}_j c_k}}$$

Notice that applying rule `1r19` leads to an infinite computation, as object b_j cannot be idle. Thus, rule `2r9` will eventually be applied.

$$\breve{a}_j a_j a_3 a_5 q_{l,2} q_{l,4} \boxed{\mathbf{I_c c} \hat{q}_i \hat{a} \mathbf{a_1 a_2} a_4 q_{l,1} \mathbf{a'_j b_j} q_{l,3} q_{l,5} \boxed{d_j \hat{a}_j c_k}}$$

$$\Rightarrow^{\mathtt{2r9,1r14,1s1,1s3}}$$

$$\mathbf{I_c c} \breve{a}_j a_j a_1 \mathbf{a_2 a_3} a_5 q_{l,2} q_{l,4} \boxed{\hat{q}_i \hat{a} a_4 q_{l,1} q_{l,3} q_{l,5} \boxed{\mathbf{d_j a'_j} b_j \hat{a}_j c_k}}$$

$$\Rightarrow^{\mathtt{2r10,1r15,1s2}}$$

$$\breve{a}_j a_j a_1 a_5 q_{l,2} q_{l,4} \boxed{\mathbf{I_c c} \hat{q}_i a_2 \mathbf{a_3 a_4 \hat{a} d_j a'_j q_{l,1}} q_{l,3} q_{l,5} \boxed{b_j \hat{a}_j c_k}}$$

$$\Rightarrow^{\mathtt{1r19,1r25,1r16,1s1,1s3}}$$

$$\mathbf{I_c c} a_j \mathbf{\breve{a}_j d_j \hat{a} a_1} a_3 \mathbf{a_4 a_5} a'_j \mathbf{q_{l,1} q_{l,2}} q_{l,4} \boxed{\hat{q}_i a_2 q_{l,3} q_{l,5} \boxed{b_j \hat{a}_j c_k}}$$

$$\Rightarrow^{\mathtt{1r27,1r13,1r17,1r20,1s2}}$$

$$a_j a_3 a'_j q_{l,4} \boxed{I_c c \hat{q}_i \hat{a} a_1 a_2 a_4 \breve{a}_j d_j a_5 q_{l,1} q_{l,2} q_{l,3} q_{l,5} \boxed{b_j \hat{a}_j c_k}}$$

Now we can apply the rules `1r25`, `1r18` or `2r11`. It is easy to see that applying rule `1r25` leads to an infinite computation (rules `1r26`, `2r6`, `2r7`), which is true for rule `1r18`, too (rules `1r28`, `1r10`, `2r6`, `2r7`). Hence, now consider applying rule `2r11`.

$$a_j a_3 a'_j q_{l,4} q_l \boxed{\mathbf{I_c c}\hat{q}_l\hat{q}_i\hat{a}\mathbf{a_1 a_2} a_4 \breve{a}_j \mathbf{d_j a_5} q_{l,1}\mathbf{q_{l,2} q_{l,3}} q_{l,5} \boxed{b_j \hat{a}_j c_k}}$$
$$\Rightarrow^{\texttt{2r11,1r21,1r14,1s1,1s3}}$$
$$\mathbf{I_c c} a_j a_1 \mathbf{a_2 a_3} a'_j q_{l,2}\mathbf{q_{l,3} q_{l,4}} q_l \boxed{\hat{q}_l \hat{q}_i \hat{a} a_4 \breve{a}_j q_{l,1} q_{l,5} \boxed{d_j \mathbf{a_5} b_j \hat{a}_j c_k}}$$
$$\Rightarrow^{\texttt{2r12,1r15,1r22,1s2}}$$
$$a_j a_1 a'_j q_{l,2} q_l \boxed{\mathbf{I_c c}\hat{q}_l\hat{q}_i \hat{a} a_2 \mathbf{a_3 a_4 a_5}\breve{a}_j q_{l,1} q_{l,3}\mathbf{q_{l,4} q_{l,5}} \boxed{d_j b_j \hat{a}_j c_k}}$$
$$\Rightarrow^{\texttt{1r16,1r18,1r23,1s1,1s3}}$$
$$\mathbf{I_c c} a_j a_1 a_3 \mathbf{a_4 a_5} a'_j q_{l,2} q_{l,4} \mathbf{q_{l,5} q_l} \boxed{\hat{q}_l \hat{q}_i \hat{a} a_2 \breve{a}_j q_{l,1} q_{l,3} \boxed{d_j b_j \hat{a}_j c_k}}$$
$$\Rightarrow^{\texttt{1r17,1r24,1s2}}$$
$$a_j a_1 a_3 a'_j q_{l,2} q_{l,4} \boxed{\mathbf{I_c c q_l \hat{q}_l} \hat{q}_i \hat{a} a_2 a_4 \mathbf{a_5} \breve{a}_j q_{l,1} q_{l,3} q_{l,5} \boxed{d_j b_j \hat{a}_j c_k}}$$
$$\Rightarrow^{\texttt{1r1,1r18,1s1,1s3}}$$
$$I_c c a_j a_1 a_3 a_5 a'_j q_{l,2} q_{l,4} q_l \hat{q}_l \boxed{\hat{q}_i \hat{a} a_2 a_4 \breve{a}_j q_{l,1} q_{l,3} q_{l,5} \boxed{d_j b_j \hat{a}_j c_k}}$$

Thus, we begin a new circle of modelling.

"Decrement"-instruction.

If there is an object c_k in region 2, we obtain the following computation:

$$a_j \boxed{\mathbf{q_i \hat{q}_i}\breve{a}_j \hat{a} \boxed{b_j c_k \#_2}} \Rightarrow^{\texttt{1r1}} q_i \mathbf{\hat{q}_i a_j}\boxed{\breve{a}_j \hat{a} \boxed{b_j c_k \#_2}} \Rightarrow^{\texttt{1r2}}$$
$$q_i \boxed{\hat{q}_i a_j \breve{a}_j \hat{a} \boxed{b_j c_k \#_2}}$$

Now there are two variants of computations (depending on the application of rule `2r1` or rule `1r3`). It is easy to see that the application of rule `1r3` leads to an infinite computation (by *"circle"*). Now consider applying rule `2r1`:

$$q_i \boxed{\hat{q}_i \mathbf{a_j \breve{a}_j} \hat{a} \boxed{b_j c_k \#_2}} \Rightarrow^{\texttt{2r1}} q_i \boxed{\hat{q}_i \hat{a} \boxed{\mathbf{b_j \breve{a}_j a_j c_k} \#_2}} \Rightarrow^{\texttt{2r2,2r3}}$$
$$q_i \boxed{\hat{q}_i b_j \breve{a}_j \hat{a} a_j c_k \boxed{\#_2}}$$

Thus, object c_k is moved from region 2 to region 1 (i.e., we decrease the number of objects c_k in region 2 by one and in that way model the "decrement"-instruction of the counter automaton).

The case when there is no object c_k in region 2 leads to an infinite computation (rules `2r4`, `1r5`, `1r6`), hence, again we correctly model the "decrement"-instruction. The further behavior of the system is the same as in the case of modelling the "increment"-instruction.

"Test for zero"-instruction:

q_i is replaced by q_l if there is no c_k in region 2 (case a)), otherwise the computation will never stop (case b)).

Case a):

$$a_j \boxed{\mathbf{q_i}\hat{\mathbf{q}}_\mathbf{i}\breve{a}_j\hat{b}d'_jd''_j\boxed{b_jd_j\#_2}} \Rightarrow^{\texttt{1r1}} q_i\hat{\mathbf{q}}_\mathbf{i}\mathbf{a_j}\boxed{\breve{a}_j\hat{b}d'_jd''_j\boxed{b_jd_j\#_2}} \Rightarrow^{\texttt{1r2}}$$

$$q_i\boxed{\hat{q}_ia_j\breve{a}_j\hat{b}d'_jd''_j\boxed{b_jd_j\#_2}}$$

Now there are two variants of computations (depending on the application of rule `2r1` or rule `1r4`). It is easy to see that the application of rule `1r4` leads to an infinite computation (by *"circle"*). Consider the application of rule `2r1`:

$$q_iq_{l,2}q_{l,4}q_la'_j\boxed{\hat{q}_i\mathbf{a_j}\breve{\mathbf{a}}_\mathbf{j}q_{l,1}q_{l,3}q_{l,5}\hat{b}d'_jd''_j\boxed{b_jd_j\#_2}} \Rightarrow^{\texttt{2r1}}$$

$$q_iq_{l,2}q_{l,4}q_la'_j\boxed{\hat{q}_iq_{l,1}q_{l,3}q_{l,5}\hat{b}d'_jd''_j\boxed{a_j\breve{\mathbf{a}}_\mathbf{j}\mathbf{b_j}d_j\#_2}} \Rightarrow^{\texttt{2r2}}$$

$$q_iq_{l,2}q_{l,4}q_la'_j\boxed{\hat{q}_i\breve{\mathbf{a}}_\mathbf{j}\mathbf{b_j}q_{l,1}q_{l,3}q_{l,5}\hat{b}d'_jd''_j\boxed{a_jd_j\#_2}} \Rightarrow^{\texttt{1r7}}$$

$$q_iq_{l,2}q_{l,4}q_l\breve{a}_j\mathbf{b_j}\mathbf{a'_j}\boxed{\hat{q}_iq_{l,1}q_{l,3}q_{l,5}\hat{b}d'_jd''_j\boxed{a_jd_j\#_2}} \Rightarrow^{\texttt{1r12}}$$

$$q_iq_{l,2}q_{l,4}q_l\breve{a}_j\boxed{\hat{q}_ib_ja'_jq_{l,1}q_{l,3}q_{l,5}\hat{b}d'_jd''_j\boxed{a_jd_j\#_2}}$$

Again there are two variants of computations, depending on the application of rule `1r19` or rule `2r9`. Notice that applying rule `1r19` leads to an infinite computation, as object b_j cannot be idle (rules `1r8`, `1r10`, `2r6`, `2r7`). Hence, we only consider the case of applying rule `2r9`:

$$q_iq_{l,2}q_{l,4}q_l\breve{a}_j\boxed{\hat{q}_i\mathbf{b_j}\mathbf{a'_j}q_{l,1}q_{l,3}q_{l,5}\hat{b}d'_jd''_j\boxed{a_jd_j\#_2}} \Rightarrow^{\texttt{2r9}}$$

$$q_iq_{l,2}q_{l,4}q_l\breve{a}_j\boxed{\hat{q}_iq_{l,1}q_{l,3}q_{l,5}\hat{b}d'_jd''_j\boxed{a_jb_j\mathbf{a'_j}\mathbf{d_j}\#_2}} \Rightarrow^{\texttt{2r10}}$$

$$q_iq_{l,2}q_{l,4}q_l\breve{a}_j\boxed{\hat{q}_ia'_jq_{l,1}q_{l,3}q_{l,5}\hat{b}d_jd'_jd''_j\boxed{a_jb_j\#_2}}$$

Now there are two variants of computations, depending on the application of rule `2r13` and `1r29`. It is easy to see that applying rule `2r14` leads to an infinite computation (rules `2r14`, `1r4`, `1r11`, `2r6`, `2r7`). Hence, consider applying rule `1r29`:

$$q_i q_{l,2} q_{l,4} q_l \breve{a}_j \boxed{\hat{q}_i \mathbf{a'_j q_{l,1}} q_{l,3} q_{l,5} \hat{b} \mathbf{d_j d'_j} d''_j \boxed{a_j b_j \#_2}} \Rightarrow^{\texttt{1r29,1r19}}$$

$$q_i a'_j \mathbf{q_{l,1} q_{l,2}} q_{l,4} q_l \mathbf{\breve{a}_j d_j} d'_j \boxed{\hat{q}_i q_{l,3} q_{l,5} \hat{b} d''_j \boxed{a_j b_j \#_2}} \Rightarrow^{\texttt{1r20,1r27}}$$

$$q_i a'_j q_{l,4} q_l d'_j \boxed{\hat{q}_i q_{l,1} \mathbf{q_{l,2} q_{l,3}} q_{l,5} \hat{b} \breve{a}_j \mathbf{d_j d''_j} \boxed{a_j b_j \#_2}} \Rightarrow^{\texttt{1r21,2r13}}$$

$$q_i a'_j q_{l,2} \mathbf{q_{l,3} q_{l,4}} q_l d'_j \boxed{\hat{q}_i q_{l,1} q_{l,5} \hat{b} \breve{a}_j \boxed{d_j \mathbf{d''_j a_j} b_j \#_2}} \Rightarrow^{\texttt{1r22,2r14}}$$

$$q_i a'_j q_{l,2} q_l d'_j \boxed{\hat{q}_i q_{l,1} q_{l,3} \mathbf{q_{l,4} q_{l,5}} d''_j \mathbf{a_j \hat{b}} \breve{a}_j \boxed{d_j b_j \#_2}} \Rightarrow^{\texttt{1r4,1r23}}$$

$$q_i a'_j q_{l,2} q_{l,4} \mathbf{q_{l,5} q_l} a_j \mathbf{\hat{b} d'_j} \boxed{\hat{q}_i q_{l,1} q_{l,3} d''_j \breve{a}_j \boxed{d_j b_j \#_2}} \Rightarrow^{\texttt{1r24,1r30}}$$

$$q_i a'_j q_{l,2} q_{l,4} a_j \boxed{\hat{q}_i q_{l,1} q_{l,3} q_{l,5} q_l \hat{b} d'_j d''_j \breve{a}_j \boxed{d_j b_j \#_2}}$$

Thus, q_i is replaced by q_l in region 1.

Case b):

$$a_j \boxed{\mathbf{q_i \hat{q}_i} \breve{a}_j \hat{b} \boxed{c_k b_j d_j \#_2}} \Rightarrow^{\texttt{1r1}} q_i \mathbf{\hat{q}_i a_j} \boxed{\breve{a}_j \hat{b} \boxed{c_k b_j d_j \#_2}} \Rightarrow^{\texttt{1r2}}$$
$$q_i \boxed{\hat{q}_i a_j \breve{a}_j \hat{b} \boxed{c_k b_j d_j \#_2}}$$

Again there are two variants of computations (depending on the application of rule `2r1` or rule `1r4`). It is easy to see that the application of rule `1r4` leads to infinite computation (by *"circle"*). Consider the applying of rule `2r1`:

$$q_i \boxed{\hat{q}_i \mathbf{a_j \breve{a}_j} \hat{b} \boxed{c_k b_j d_j \#_2}} \Rightarrow^{\texttt{2r1}} q_i \boxed{\hat{q}_i \hat{b} \boxed{\mathbf{c_k a_j \breve{a}_j b_j} d_j \#_2}} \Rightarrow^{\texttt{2r2,2r3}}$$
$$q_i \boxed{\hat{q}_i \breve{a}_j b_j c_k a_j \hat{b} \boxed{d_j \#_2}}$$

There are two variants of computations, depending on the application of rule `2r1` or rule `1r4`. Notice that they both lead to infinite computations. Indeed, if rule `2r1` will be applied, then rules `1r8`, `1r10`, `2r6`, `2r7` will be applied (applying rules `2r6`, `2r7` leads to an infinite computation). If rule `1r4` will be applied, it again leads to an infinite computation (rules `1r11`, `2r6`, `2r7`). Thus, we correctly model a "test for zero"-instruction.

3. END.

$$\begin{aligned} R_{1,f} = & \{\texttt{1f1} : (\$_1 \breve{a}_j, out) \mid j \in I\} \\ & \cup \{\texttt{1f2} : (\$_2 e_1, out), \texttt{1f3} : (\$_1 \$_3, out)\} \\ & \cup \{\texttt{1f4} : (e_t h_t, in) \mid t \in I'\} \\ & \cup \{\texttt{1f5} : (h_t e_{t+1}, out) \mid 1 \le t \le n+3\}, \end{aligned}$$

$$\begin{aligned} R_{2,f} = \{ & \texttt{2f1} : (q_f, in), \texttt{2f2} : (q_f\$_1, out), \texttt{2f3} : (q_f\$_2, out)\} \\ & \cup \{\texttt{2f4} : (\$_1\hat{a}, in), \texttt{2f5} : (\$_1\#_1, in), \texttt{2f6} : (\$_1 I_c, in)\} \\ & \cup \{\texttt{2f7} : (h_{n+4}, in)\} \\ & \cup \{\texttt{2f8} : (h_{n+4}\hat{a}_j, out) \mid j \in I\} \\ & \cup \{\texttt{2f9} : (h_{n+4}b_j, out) \mid j \in I\} \\ & \cup \{\texttt{2f10} : (h_{n+4}d_j, out) \mid j \in I\}. \end{aligned}$$

At first, all objects $\check{a}_j$ will be moved to the environment and the objects $\hat{a}, \#_1, I_c$ to region 2 (thus, we stop without continuing the loop) and after that all objects $\hat{a}_j, b_j, d_j$ will be moved from region 2 to region 1. Hence, in region 2 now there are only the objects c_1 (representing the result of the computation) and the six additional objects $\#_1$, $\#_2$, $\hat{a}$, I_c, q_f, h_{n+4}. □

Both constructions from Theorem 2 and Theorem 3 can easily be modified to show that

$PsOP_2(sym_1, anti_1)_T = PsRE$ and
$PsOP_2(sym_2)_T = PsRE$,

i.e., the results proved in Theorem 2 and Theorem 3 can be extended from sets of natural numbers to sets of vectors of natural numbers.

5 Final Remarks

In this paper we have proved the new results that P systems with minimal cooperation, i.e., P systems with symport/antiport rules of size one, are computationally complete with only two membranes: they generate all recursively enumerable sets of vectors of nonnegative integers excluding (at most) the initial segment $\{0, 1, 2\}$. In an analogous manner, P systems with symport rules of size two are computationally complete with only two membranes: they generate all recursively enumerable sets of vectors of nonnegative integers excluding (at most) the initial segment $\{0, 1, 2, 3, 4, 5\}$. On the other hand it is known that systems with such rules in only one membrane cannot be universal, see [21, 47, 7]. Hence, the results we have proved in this paper are optimal with respect to the number of membranes. Notice that for *tissue* P systems with minimal cooperation this problem has already been solved successfully ([8]), i.e., it was proved that two cells are enough to generate all recursively enumerable sets of natural numbers.

Moreover, for P systems with symport rules of weight three we already obtain computational completeness with only one membrane modulo the initial segment $\{0, 1, 2, 3, 4, 5, 6\}$, which improves the result of [21], where thirteen objects remained in the skin membrane at the end of a halting computation.

As so far we have not been able to completely avoid additional symbols that remain after a computation has halted, the interesting open question remains to find the minimal numbers of these additional objects that permit to obtain computationally completeness in the cases described above.

Acknowledgements

The first author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona. The first author and the third author acknowledge the U.S. Civilian Research and Development Foundation (CRDF) and the Moldavian Research and Development Association (MRDA), Award No. MM2-3034 for providing a challenging and fruitful framework for cooperation. This article was written during the first author's stay at the Vienna University of Technology.

References

1. A. Alhazov, R. Freund: P systems with one membrane and symport/antiport rules of five symbols are computationally complete. In [25], 19–28.
2. A. Alhazov, R. Freund, M. Oswald: Tissue P systems with antiport rules and a small number of symbols and cells. In *Developments in Language Theory, 9th International Conference, DLT 2005* (C. De Felice, A. Restivo, eds.), Palermo, Italy, July 4 – 8, 2005, LNCS 3572, Springer, Berlin, 2005, 100–111.
3. A. Alhazov, R. Freund, M. Oswald: Symbol/membrane complexity of P systems with symport/antiport rules. In [12], 123–146.
4. A. Alhazov, R. Freund, Yu. Rogozhin: Computational power of symport/antiport: history, advances and open problems. In [12], 44–78.
5. A. Alhazov, R. Freund, Yu. Rogozhin: Some optimal results on communicative P systems with minimal cooperation. In [24], 23–36.
6. A. Alhazov, M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan: Communicative P systems with minimal cooperation. In [36], 161–177.
7. A. Alhazov, Yu. Rogozhin: Minimal cooperation in symport/antiport P systems with one membrane. In [25], 29–34.
8. A. Alhazov, Yu. Rogozhin, S. Verlan: Symport/antiport tissue P systems with minimal cooperation. In [24], 37 – 52.
9. F. Bernardini, M. Gheorghe: On the power of minimal symport/antiport. In *Pre-proceedings of Workshop on Membrane Computing, WMC-2003* (A. Alhazov, C. Martín-Vide, Gh. Păun, eds.), Tarragona, July 17–22, 2003, Technical Report RGML 28/03, Universitat Rovira i Virgili, Tarragona, 2003, 72–83.
10. F. Bernardini, A. Păun: Universality of minimal symport/antiport: five membranes suffice. In *Membrane Computing, International Workshop, WMC 2003, Tarragona, July 2003, Selected Papers* (C. Martin-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 2933, Springer, Berlin, 2004, 43–45.
11. C.S. Calude, Gh. Păun: Bio-steps beyond Turing. *BioSystems*, 77 (2004), 175–194.
12. R. Freund, G. Lojka, M. Oswald, Gh. Păun, eds.: *Pre-proceedings of Sixth International Workshop on Membrane Computing, WMC6*, Vienna, July 18–21, 2005.
13. R. Freund, M. Oswald: GP systems with forbidding context. *Fundamenta Informaticae*, 49 (2002), 81–102.
14. R. Freund, M. Oswald: A short note on analysing P systems with antiport rules. *Bulletin of the European Association for Theoretical Computer Science*, 78 (2002) 231–236.
15. R. Freund, M. Oswald: P systems with activated/prohibited membrane channels. In [44], 261–268.

16. R. Freund, M. Oswald: Tissue P systems with symport/antiport rules of one symbol are computationally universal. In [24], 187–200.
17. R. Freund, A. Păun: Membrane systems with symport/antiport: universality results. In [44], 270–287.
18. R. Freund, Gh. Păun: On deterministic P Systems. Manuscript, 2003 (available at `http://psystems.disco.unimib.it`).
19. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel states. In [43], 206–223, and *Theoretical Computer Science*, 330 (2005), 101–116.
20. P. Frisco: About P systems with symport/antiport. In [43], 224–236.
21. P. Frisco, H.J. Hoogeboom: P systems with symport/antiport simulating counter automata. *Acta Informatica*, 41 (2004), 145–170.
22. P. Frisco, H.J. Hoogeboom: Simulating counter automata by P systems with symport/antiport. In [44], 288–301.
23. S. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7 (1978), 311–324.
24. M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Cellular Computing. Complexity Aspects*. Fenix Editora, Sevilla, 2005.
25. M.A. Gutierrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan, eds.: *Proceedings of the Third Brainstorming Week on Membrane Computing*, Sevilla (Spain), January 31 – February 4, 2005.
26. O.H. Ibarra: On determinism versus nondeterminism in P systems. *Theoretical Computer Science*, to appear.
27. O.H. Ibarra, S.Woodworth: On bounded symport/antiport P systems. *Proc. DNA11*, UWO, London, Ontario, 2005, 37–48, and LNCS, to appear.
28. O.H. Ibarra: Some recent results concerning deterministic P systems. In [12], 24–25.
29. O. Ibarra, S. Woodworth: On symport/antiport P systems with one or two symbols. In *Pre-Proceedings of the Workshop on Theory and Applications of P Systems*, Timişoara, September 26-27, 2005, 75–82.
30. O.H. Ibarra, S. Woodworth, H. Yen, Z. Dang: On symport/antiport systems and semilinear sets. In [12], 312–335.
31. L. Kari, C. Martín-Vide, A. Păun: On the universality of P systems with minimal symport/antiport rules. In *Aspects of Molecular Computing. Essays Dedicated to Tom Head on the Occasion of His 70th Birthday* (N. Jonoska, Gh. Păun, G. Rozenberg, eds.), LNCS 2950, Springer, Berlin, 2004 254–265.
32. M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan: About P systems with minimal symport/antiport rules and four membranes. In [35], 283–294.
33. C. Martín-Vide, A. Păun, Gh. Păun: On the power of P systems with symport rules, *Journal of Universal Computer Science*, 8 (2002), 317–331.
34. C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Patón: Tissue P systems. *Theoretical Computer Science*, 296 (2003), 295–326.
35. G. Mauri, Gh. Păun, C. Zandron, eds.: *Pre-Proceedings of Fifth Workshop on Membrane Computing (WMC5)*, Universitá di Milano-Bicocca, Italy, June 14–16, 2004.
36. G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing. 5th Inter. Workshop, WMC5, Milan, Italy, June 2004, Revised Selected and Invited Papers*. LNCS 3365, Springer, Berlin, 2005.
37. M.L. Minsky: *Finite and infinite machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
38. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20 (2002), 295–305.

39. Gh. Păun: Computing with membranes. *Journal of Computer and Systems Science*, 61 (2000), 108–143.
40. Gh. Păun: *Membrane computing. An Introduction.* Springer-Verlag, 2002.
41. Gh. Păun: Further twenty six open problems in membrane computing. In [25], 249–262.
42. Gh. Păun, J. Pazos, M.J. Perez-Jimenez, A. Rodriguez-Paton: Symport/antiport P systems with three objects are universal. *Fundamenta Informaticae*, 64 (2005), 1–4.
43. Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds.: *Second Brainstorming Week on Membrane Computing.* Technical report of Research Group on Natural Computing, University of Seville, TR 01, 2004.
44. Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.: *Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, August 19–23, 2002. Revised Papers.* LNCS 2597, Springer, Berlin, 2003.
45. G. Rozenberg, A. Salomaa, eds.: Handbook of formal languages (3 volumes). Springer, Berlin, 1997.
46. Gy. Vaszil: On the size of P systems with minimal symport/antiport. In [35], 422–431.
47. S. Verlan: Optimal results on tissue P systems with minimal symport/antiport. Presented at *EMCC meeting*, Lorentz Center, Leiden, The Netherlands, 22–26 November, 2004.
48. S. Verlan: Tissue P systems with minimal symport/antiport. In *Developments in Language Theory, DLT 2004* (C.S. Calude, E. Calude, M.J. Dinneen, eds.), LNCS 3340, Springer, Berlin, 2004, 418–430.