Symport/Antiport P Systems
with Three Objects Are Universal

Gheorghe Păun[1,2]**, Juan Pazos**[3]**,**
Mario J. Pérez-Jiménez[2]**, Alfonso Rodríguez-Patón**[3]

[1]Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
E-mail: `george.paun@imar.ro`

[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: `gpaun/marper@us.es`

[3]Department of Artificial Intelligence, Faculty of Computer Science
Polytechnical University of Madrid
Campus de Montegancedo
Boadilla del Monte 28660, Madrid, Spain
E-mail: `jpazos/arpaton@fi.upm.es`

Abstract. The operations of symport and antiport, directly inspired from biology, are already known to be rather powerful when used in the framework of P systems. In this paper we confirm this observation with a quite surprising result: P systems with symport/antiport rules using only three objects can simulate any counter machine, while systems with only two objects can simulate any blind counter machine. In the first case, the universality (of generating sets of numbers) is obtained also for a small number of membranes, four.

1   Introduction

P systems with symport/antiport [8] use as rules for processing the symbol-objects operations coming from biology, namely simultaneous trans-membrane transportation of several chemicals, either in the same direction (and this is called symport) or in opposite directions (which is called antiport) – biochemical details can be found in [1]. Since a considerable degree of cooperation (context-sensitivity) among objects is available, it is

1

somewhat expected that the systems of this type are universal, equal in power with Turing machines, for various combinations of ingredients (number of membranes, size of symport or antiport rules). Not so expected is the fact that such systems are universal even when using minimal symport/antiport rules, moving at most one object in each direction; the currently strongest results of this type state the universality for systems with three membranes, [2], [12], without knowing whether two membranes suffice in order to reach the universality.

We start here a somewhat "orthogonal" direction of research: we let free the number of membranes and the size of rules, and concentrate on the number of objects used in a system. Can this number be bounded? If so, which is the price (which other parameters should grow)? Which is the smallest number of objects which ensures the universality?

That a bound on the number of objects can be imposed is again somewhat expected, for instance, having in mind that there are small universal Turing machines (with the size estimated in both the number of tape symbols and of states – which can be simultaneously bounded at rather low values), see, e.g., [10]. However, there is no direct simulation of a Turing machine by a P system with symport/antiport rules, hence the results about small universal Turing machines cannot be used in our framework. (Furthermore, Turing machines use the positional information provided by the tape, while here we do not have such a data structure, we work with multisets of symbol-objects.)

The bound we find here on the number of objects is surprisingly small: three objects suffice. The proof is based on a unary codification of all numbers we deal with in a P system (which simulates a counter machine), with two additional symbols necessary in order to control the work of the system. One of these additional objects can be saved when simulating blind counter machines.

2 Prerequisites

We introduce here both the definition of P systems with symport/antiport rules, the class of P systems we investigate, and that of counter machines, the tool used in the proof of our result – but the reader is supposed to have some familiarity with membrane computing, for instance, from [9]; details can be found at the web address http://psystems.disco.unimib.it.

A P system with symport/antiport rules is a construct of the form $\Pi = (O, H, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_o)$, where O is the alphabet of objects, H is the finite set of labels for membranes (in general, one uses natural numbers as labels), μ is the membrane structure (of degree $m \geq 1$, with the membranes labelled in a one-to-one manner with elements of H; as usual, we represent the membrane structures by strings of matching labelled parentheses), $w_1, \ldots, w_m$ are strings over O representing the multisets of objects present in the m compartments of μ in the initial configuration of the system, $E \subseteq O$ is the set of objects supposed to appear in the environment in arbitrarily many copies, $R_1, \ldots, R_m$ are the (finite) sets of rules associated with the m membranes of μ, and $i_o \in H$ is the label of a membrane of μ, which indicates the *output* region of the

system (note that we do not impose that the output region is enclosed by an elementary membrane – although in the main result of the paper, as well as in many papers from the literature, this is the case).

The rules from R can be of two types (by O^+ we denote the set of all non-empty strings over O, with λ denoting the empty string):

- *Symport rules*, of the forms (x, in) or (x, out), where $x \in O^+$. When using such a rule, the objects specified by x enter or exit, respectively, the membrane with which the rule is associated. In this way, objects are sent to or imported from the surrounding region – which is the environment in the case of the skin membrane. (The length of x in a symport rule is called the *weight* of the rule.)

- *Antiport rules*, of the form $(x, out; y, in)$, where $x, y \in O^+$. When using such a rule for a membrane i, the objects specified by x exit the membrane and those specified by y enter from the region surrounding membrane i; this is the environment in the case of the skin membrane. (The maximal length of x, y is called the *weight* of the rule.)

The rules are used in the non-deterministic maximally parallel manner, standard in membrane computing. In this way, we obtain transitions from a configuration of the system to the next configuration. A sequence of transitions constitutes a computation, and a computation halts when a configuration is reached where no rule can be applied. The number of objects present in region i_o in the halting configuration is said to be computed by the system along that computation; the set of all numbers computed in this way by Π is denoted by $N(\Pi)$. The family of all sets $N(\Pi)$ of numbers computed as above by P systems with at most m membranes, using symport rules of weight at most r and antiport rules of weight at most q, for $m \geq 1$ and $r, q \geq 0$, is denoted by $NOP_m(sym_r, anti_q)$.

By NRE we denote the family of Turing computable sets of natural numbers.

Many results of the form $NRE = NOP_m(sym_r, anti_q)$ are known, for various combinations of the parameters m, r, q – see [2, 3, 9, 12], etc.

Here we consider one further parameter: the cardinality of the set O of objects. By $NOP_m(obj_n, sym_r, anti_q)$ we denote the family of sets of numbers $N(\Pi)$ computed by P systems with at most m membranes, at most n objects, and symport and antiport rules of weight at most r, q, respectively. When one of these parameters is not bounded, we replace it with $*$. Thus, $NOP_m(sym_r, anti_q)$ corresponds to the family $NOP_m(obj_*, sym_r, anti_q)$.

In order to also compute sets of vectors, as done in several cases in membrane computing, we cannot proceed in the usual way, distinguishing the objects from the output membrane and considering their multiplicities, but we can consider *several output membranes*: instead of i_o, we take $i_{o,1}, i_{o,2}, \ldots, i_{o,k}$, for some $k \geq 1$. Thus, if we halt with multisets $z_1, z_2, \ldots, z_k$ in these membranes, then we have computed the vector $(|z_1|, |z_2|, \ldots, |z_k|)$ (where $|z|$ is the lenght of the string z, hence the total multiplicity of the multiset represented by z). In this case, we denote by $Ps(\Pi)$ the set of all vectors computed by a system

3

Π and by $PsOP_m(obj_j, sym_r, anti_q)$ the family corresponding to $NOP_m(obj_j, sym_r, anti_q)$. In turn, we denote by $PsRE$ the family of Turing computable sets of vectors of natural numbers.

In the proof from the next sections we will use counter machines as devices characterizing $PsRE$ (hence also NRE), hence the Turing computability. In various places, one uses various names for the same device, sometimes with different formalisms and slight differences in architecture and functioning: register machines, program machines, multi-counter machines, etc.; we adhere here to the following terminology and definition.

Informally speaking, a counter machine consists of a specified number of counters which can hold any natural number, and which are handled according to a program consisting of labelled instructions; the counters can be increased or decreased by 1 – the decreasing being possible only if a counter holds a number greater than or equal to 1 (we say that it is non-empty) –, and checked whether they are non-empty.

Formally, a (non-deterministic) *counter machine* is a device $M = (m, B, l_0, l_h, R)$, where $m \geq 1$ is the number of counters, B is the (finite) set of instruction labels, l_0 is the initial label, l_h is the halting label, and R is the finite set of instructions labelled (hence uniquely identified) by elements from B (R is also called the *program* of the machine). The labelled instructions are of the following forms:

- $l_1 : (\texttt{ADD}(r), l_2, l_3)$, $1 \leq r \leq m$ (add 1 to counter r and go non-deterministically to one of the instructions with labels l_2, l_3),

- $l_1 : (\texttt{SUB}(r), l_2, l_3)$, $1 \leq r \leq m$ (if counter r is not empty, then subtract 1 from it and go to the instruction with label l_2, otherwise go to the instruction with label l_3),

- $l_h : \texttt{HALT}$ (the halt instruction, which can only have the label l_h).

A counter machine generates a k-dimensional vector of natural numbers in the following manner: we distinguish k counters as output counters (without loss of generality, they can be the first k counters), and we start computing with all m counters being empty, with the instruction labelled by l_0; if the computation reaches the instruction $l_h : \texttt{HALT}$ (we say that it halts), then the values of counters $1, 2, \ldots, k$ is the vector generated by the computation. The set of all vectors from $\mathbf{N}^k$ generated in this way by M is denoted by $Ps(M)$. If we want to generate only numbers (1-dimensional vectors), then we have the result of a computation in counter 1, and the set of numbers computed by M in this way is denoted by $N(M)$. It is known (see [7], [3]) that non-deterministic counter machines with $k + 2$ counters can compute any set of Turing computable k-dimensional vectors of natural numbers (hence machines with three counters generate exactly the family NRE, of Turing computable sets of numbers).

In the case when a counter machine cannot check whether a counter is empty one say that it is *blind*: the counters are increased and decreased by one as usual, but if the machine tries to subtract from an empty counter, then the computation aborts without producing any result. It is known (see, e.g., [5]) that blind counter machines are strictly less powerful than general counter machines (hence than Turing machines).

3   Universality with Three Objects

We pass now to giving the universality result mentioned in the Introduction.

Theorem 3.1 $PsOP_*(obj_3, sym_*, anti_*) = PsRE.$

Proof. We only prove the relation $PsRE \subseteq NOP_*(obj_3, sym_*, anti_*)$, the converse inclusion being straightforward (we can also invoke the Turing-Church thesis for it).

Let us consider a counter machine $M = (m, B, l_0, l_h, R)$ with m counters. We assume that M is non-trivial, that is, $l_0 \neq l_h$ and R contains at least one instruction, and that the output counters are the first k, that is, $1, 2, \ldots, k$. We construct a P system Π with $m + 1$ membranes which will compute the same set of vectors as M.

Essential in the construction is the following *codification* of certain relevant information items related to M. Consider the following set:

$$
\begin{aligned}
J \;=\;& B \cup \{(l_1, add_r) \mid l_1 : (\texttt{ADD}(r), l_2, l_3) \in R\} \\
\cup\;& \{(l_1, sub_{r>0}),\; (l_1, sub_{r=0}) \mid l_1 : (\texttt{SUB}(r), l_2, l_3) \in R\}.
\end{aligned}
$$

Assume that J contains n elements, $J = \{\alpha_1, \ldots, \alpha_n\}$ (their ordering is not relevant). Clearly, $n \geq 3$: we have at least two labels and at least one instruction. Then, with each element α_i we associate the natural number

$$v(\alpha_i) = 8n + 8i, \quad 1 \leq i \leq n.$$

We denote $v(J) = \{v(\alpha_i) \mid 1 \leq i \leq n\}$.

Three facts about these numbers are important for what follows:

1. $v(\alpha_i) \geq 32$ for all $1 \leq i \leq n$ (because $n \geq 3$),

2. each number $v(\alpha_i)$ is strictly bigger than the half of any other number $v(\alpha_j), 1 \leq i, j \leq n$ (hence no number $v(\alpha_i)$ can be written as the sum of any two or more than two numbers from $v(J)$),

3. if we subtract any number $t, 1 \leq t \leq 7$, from any $v(\alpha_i)$, we do not get the value of any $v(\alpha_j), 1 \leq i, j \leq n$.

Now, the system we look for is

$$\Pi = (O, H, \mu, w_0, w_1, w_2, \ldots, w_m, E, R_0, R_1, R_2, \ldots, R_m, i_{o,1}, i_{o,2}, \ldots, i_{o,k}),$$

where:

$$
\begin{aligned}
O \;&=\; \{a, b, c\}, \\
H \;&=\; \{0, 1, 2, \ldots, m\}, \\
\mu \;&=\; [_0[_1 \;]_1[_2 \;]_2 \ldots [_m \;]_m]_0 \;\; (0 \text{ is the label of the skin membrane}), \\
w_0 \;&=\; a^{v(l_0)}c, \\
w_i \;&=\; \lambda, \;\text{ for all } 1 \leq i \leq m, \\
E \;&=\; O, \\
i_{o,j} \;&=\; i, \; 1 \leq j \leq k,
\end{aligned}
$$

and the sets of rules are constructed as follows.

Let us firstly explain shortly the idea behind the construction. All labels of instructions from R and the actions of these instructions – the operations of increasing or decreasing a counter – are codified in base 1, through the mapping v, in the number of occurrences of object a. The copies of object a representing a given element of the set J will always evolve together, otherwise the computation will never stop. In order to represent the values of counters, we use the object b: the number of copies of b from a membrane $r = 1, 2, \ldots, m$ represents the value stored in counter r. When simulating a subtraction instruction, we first guess whether the counter to modify is empty or not. When a wrong guess is made, as well as in any case when a step in Π does not correctly simulate a step in M, the computation will never end. To this aim we will use a third object, c, subject of a rule of the form $(cc, out; cccc, in)$, which can indefinitely "flood" the system with copies of c. When the label l_h is reached, the computation will stop.

These general ideas are implemented as follows.

1. For each instruction $l_1 : (\mathtt{ADD}(r), l_2, l_3) \in R$, the set R_0 contains the rules

    1. $(a^{v(l_1)}, out; a^{v(l_1, add_r)}b, in)$,

    2. $(a^{v(l_1, add_r)}, out; a^{v(l_2)}, in)$, if $l_2 \neq l_h$,

    2′. $(a^{v(l_1, add_r)}, out)$, if $l_2 = l_h$,

    3. $(a^{v(l_1, add_r)}, out; a^{v(l_3)}, in)$, if $l_3 \neq l_h$,

    3′. $(a^{v(l_1, add_r)}, out)$, if $l_3 = l_h$,

and the set R_r (corresponding to this instruction) contains the rules:

    4. $(a^{v(l_1, add_r)}b, in)$,

    5. $(a^{v(l_1, add_r)}, out)$.

2. For each instruction $l_1 : (\mathtt{SUB}(r), l_2, l_3) \in R$, we introduce the following rules in R_0

    6. $(a^{v(l_1)}, out; a^{v(l_1, sub_r>0)}, in)$,

    7. $(a^{v(l_1, sub_r>0)}b, out; a^{v(l_2)}, in)$, if $l_2 \neq l_h$,

    7′. $(a^{v(l_1, sub_r>0)}b, out)$, if $l_2 = l_h$,

    8. $(a^{v(l_1)}, out; a^{v(l_1, sub_r=0)}c, in)$,

    9. $(a^{v(l_1, sub_r=0)}c^{13}, out; a^{v(l_3)}, in)$, if $l_3 \neq l_h$,

    9′. $(a^{v(l_1, sub_r=0)}c^{13}, out)$, if $l_3 = l_h$,

and the set R_r (corresponding to this instruction) contains the rules:

    10. $(a^{v(l_1, sub_r>0)}, in)$,

    11. $(a^{v(l_1, sub_r>0)}b, out)$,

    12. $(a^{v(l_1, sub_r=0)}, in)$,

13. $(a^{v(l_1,sub_r=0)}b, out)$,

14. $(a^{v(l_1,sub_r=0)}, out; c, in)$.

3. We also introduce the following rules in R_0 (let us call them "flooding rules")

15. $(a, out; c^{27}, in)$,

16. $(cc, out; c^4, in)$,

17. $(b, out; c^{27}, in)$,

and the following rules in each set $R_r, 1 \leq r \leq m$:

18. $(a, out; c, in)$,

19. (c, out).

The simulation of the instruction $l_1 : (\text{ADD}(r), l_2, l_3) \in R$ proceeds as follows. When $a^{v(l_1)}$ is present in the skin region, rule 1 can be used, bringing inside $a^{v(l_1,add_r)}b$. Note that if not all copies of a were used by this rule (that is, a rule with a smaller number of a was applied), then the remaining copies of a should exit by means of the "flooding" rule 15, which brings inside 27 copies of c. The rule 16 will be used forever, bringing inside more and more copies of c.

It is important to note here that the only possibility to send out copies of c is by rules 9 and $9'$; each time we remove 13 copies of c. However, rules 15 and 17 bring inside 27 copies of c at once; even if 13 of them will be sent out, 14 will remain, hence more than a rule of types 9 and $9'$ can remove. Moreover, rules of types 9 and $9'$ cannot be used in consecutive steps, we need at least one step in between. Thus, the 14 copies of c have to evolve at least one step in the system, and they can evolve in various manners: get doubled by rule 16, or enter membrane r, provided that copies of a are present there; from membrane r they exit in the next step and have to repeat these operations. Moreover, any a brought outside membrane r will again bring 27 copies of c inside (if some copies of a remain in membrane r, they cannot be equal to any $v(\alpha)$ from $v(J)$, hence they cannot exit the system. In all cases, the computation will never stop, the number of copies of c will increase continuosly. (The number 27 from rules of types 15 and 17 is not the smallest one which can ensure the non-halting behavior of the system in case of "wrong" steps, but this number makes clearer the non-halting and the explanations about the behavior of Π.)

Now, by means of rule 4, $a^{v(l_1,add_r)}b$ can enter membrane r. Again, any other move leads to an endless computation, because any other rule will leave copies of a unused, hence rule 15 should be applied. Now, if $a^{v(l_1,add_r)}b$ is inside membrane r, we can use a rule of type 5 and send out of this membrane the multiset $a^{v(l_1,add_r)}$, which amounts at increasing by one the value of counter r, represented by the number of copies of b present in membrane r. Again, all copies of a should be used: any other rule than the one of type 5 which uses all copies of a will leave unused at eight three copies of a; one of them will exit by means of rule 18, the other seven will remain in the membrane. In this way, in

7

the skin region we cannot use any rule for all copies of a present here, hence at least one will exit by means of rule 15, and the computation will never stop.

In the case of the correct use of rules, $a^{v(l_1,add_r)}$ should now exit the system by a rule of type 2 or 3, non-deterministically chosen, but in such a way that all copies of a are used. Thus, we bring inside $a^{v(l_2)}$ or $a^{v(l_3)}$, which is the correct continuation of the computation in M.

If one of l_2 and l_3 is equal to l_h, then $a^{v(l_1,add_r)}$ should exit the system by a rule 2' or 3', and the computation halts.

Let us consider now a subtract instruction $l_1 : (\texttt{SUB}(r), l_2, l_3) \in R$. We start with $a^{v(l_1)}$ present in the skin region. We choose non-deterministically either a rule of type 6 or one of type 8. The first case corresponds to the guess that counter r is non-empty, the latter case corresponds to the guess that counter r is empty.

Assume that we have chosen the first path, hence we have brought inside $a^{v(l_1,sub_r>0)}$. The continuation is similar to that from the case of addition, with the difference that now we remove a copy of b from membrane r, instead of bringing here one additional copy of b. First, by means of rule 10, $a^{v(l_1,sub_r>0)}$ is introduced in membrane r; if there is no copy of b here (hence the guess that the counter is non-empty was wrong), then we cannot use the corresponding rule 11, hence at least eight copies of a remains inside; this means that rule 18 can be used, and the computation never ends. If the guess was correct, hence counter r (membrane r) is non-empty, then rule 11 is used, $a^{v(l_1,sub_r>0)}b$ arrives in the skin membrane, and from here it exits the system by rule 7, which brings inside the code of l_2, the correct continuation of the simulation. In all cases where a rule different from those mentioned above is used, at least one copy of a will remain in the skin region for rule 15, and the computation will continue forever. As above, this is ensured by the fact that all values $v(\alpha), \alpha \in J$, are different from each other by at least eight units. If the label l_2 is equal to l_h, then instead of rule 7 we use rule 7' and the computation stops.

More complicated is the case when the counter r is empty. We start by bringing $a^{v(l_1,sub_r=0)}c$ inside the system, by rule 8. The multiset $a^{v(l_1,sub_r=0)}$ cannot be sent out (without leaving copies of a inside, and triggering "flooding" rules) by any other rule than one of type 9, which needs 13 copies of c; however, this multiset cannot stay in the skin region until such copies are brought into the system by rule 16, because rule 15 should be used, flooding the system with copies of c. Therefore, we have to use rule 12, and introduce $a^{v(l_1,sub_r=0)}$ in membrane r. Simultaneously, rule 16 uses the remaining copy of c and the copy already existing, and brings inside the system four copies of c.

In the next step, if membrane r contains at least one copy of b, hence the guess was wrong, then we have to use rule 13, and bring $a^{v(l_1,sub_r=0)}b$ in the skin region. No rule can now use the object b than the one of type 17, and the computation will never stop.

If the membrane r contains no object b (hence the counter r was empty), then $a^{v(l_1,sub_r=0)}$ must remain inside, and this is possible without compromising the computation, because the copies of c from the skin region can be "kept busy" by rule 16. This rule will bring 8 copies of c in the system.

We have two possibilities: to use rule 14 or not. Assume that we do not use it. Then, the 8 copies of c should evolve by rules 16 and 18. Irrespective how many times we use rule 18, we can bring outside at most eight copies of a, which is not a number from $v(J)$, hence these copies of a will evolve in the next step by the flooding rule 15. If only rule 16 has been used, then the number of copies of c is doubled, we have now 16 in the skin region. The reasoning continues. If we do not use rule 14, then either we bring outside membrane r at most 16 copies of a – again less than any code $v(\alpha)$, hence the flooding rule 15 must be used – or we double the copies of c, that is, we get 32 copies; this is more than 27, the computation will never end.

Therefore, rule 14 must be used. Because no number from $v(J)$ can be written as the sum of other numbers, this rule can be used only once. Assume that we use if for a wrong instruction, that is, not removing all copies of a from membrane r. This means that at least 8 copies of a remains in membrane r.

If we are in the step when we had 8 copies of c in the skin membrane, simultaneously, the 7 copies which are not used by rule 14 either bring copies of a out of membrane r or, at most six of them, get doubled by rule 16. In any case, out of membrane r we cannot get the code of a correct instruction, hence we can only use rule 16 for at most three pairs of c and rule 18 for the remaining copy. We have now 13 available copies of c for using rule 9 (or $9'$), but one copy of a has remained in the skin region and will flood the system with copies of c.

In the case when we have 16 copies of c in the skin membrane, in the step when rule 14 is used (bringing a multiset $a^{v(l,sub_{p=0})}$ out of membrane r – remember that we discuss the case when rule 14 is used for a wrong instruction) we have 15 remaining copies of c in the skin membrane. If in membrane r we have exactly 8 objects, we can bring them out by means of 8 copies of c (rule 18), while 6 of the other 7 copies of c will use rule 16, leading to a total of 13 copies of c in the skin membrane. This means that in the skin membrane we have both completed the correct number of copies of a for recomposing the value of $v(l_1, sub_{r=0})$ and 13 copies of c; we can exit by a rule 9 or $9'$. However, 9 copies of c are inside membrane r. In the next step, they exit membrane r, while a rule of types 1, 6 or 8 is used. The continuations from rules of types 1 and 6 have no way to remove c (without leaving copies of a inside the system). Assume that we have used a rule of type 8, hence again dealing with a subtraction where we guess that a counter is empty. There is no way to remove 9 copies of c in the next step, hence the number of c will become 17. In the meantime, the multiset of copies of a has entered a membrane q.

Let us look to the possibilities for the 17 copies of c from the skin membrane in the next step. The number is odd, hence at least one copy of c cannot be used by rule 16, hence copies of a will be brought out of membrane p. If we use only rule 18, we cannot reach the value of any $v(\alpha)$, the copies of a from the skin region will flood the system with copies of c. Thus, we have to use rule 14. If we bring all copies of a outside membrane p (this happens irrespective whether there is any b here, which is not correct with respect to M), then the other 16 copies of c will get doubled; 13 copies of c can exit by a rule 9 or $9'$, but the other 19 which remain will get doubled by rule 16, and the computation will never stop.

It remains to use a rule of type 14 and several times rule 18, but in such a way to complete in the skin region a correct code – this means that we have to use rule 18 either 8 times or 16 times, the possible differences between codes from $v(J)$. If we use rule 18 for all 16 copies of c, then we will reach a configuration with 17 copies of c in membrane q; even if all copies of a from the skin region can exit, in the next step the 17 copies of c exit membrane q and, one step later, when using one rule of types 1, 6 or 8, we have to use rule 16 for all pairs of copies of c and the system will contain 33 copies of c. The only remaining case is to use only 8 times rule 18, which means that 9 copies of c remain in the skin region, to be used by rule 16; the result is that we get 17 copies of c in the skin region. (If any a remains in membrane q, at least one should be brought out by a rule 18 and the computation will never stop.) In the next step, all copies of a should exit the system (otherwise a floods the system with copies of c), and this is possible only together with 13 copies of c. This means that 4 copies remain, and are doubled by rule 16. These 8 copies of c together with the 9 ones from membrane q (they exit at the same time with the doubling of the 4 copies of c from the skin region) make a total of 17 copies, which will be doubled in the next step, leading again to 33 copies. The computation never ends.

Therefore, the only continuation which is not immediately compromising the computation is to use rule 14 for all copies of a from membrane r, thus introducing one copy of c in membrane r and sending $a^{v(l_1, sub_r=0)}$ to the skin region; simultaneously, 6 out of the 7 copies of c remained in the skin region will be used by rule 16, hence we bring 6 more copies of c into the system (in total, we have now 13 copies of c in the skin region, and one in membrane r). In the next step, the only continuation which does not lead to an endless computation (a cannot stay unused in the skin region) is to use rule 9 and send $a^{v(l_1, sub_r=0)} c^{13}$ out of the system, in exchange of $a^{v(l_3)}$; at the same time, c exits membrane r, hence the configuration returns to a form as that from which we have started. (Of course, when $l_3 = l_h$, no object is introduced, and the computation stops.)

We can continue by simulating instructions of the counter machine M. When the halting instruction is reached in M, no multiset $a^{v(l_h)}$ is introduced, hence the computation stops.

Consequently, any halting computation in M can be simulated by a halting computation in Π, and, conversely, if a computation in Π stops, then it corresponds to a halting computation in M; in both cases, the number of copies of object b from membranes $1, 2, \ldots, k$ is equal to the value of counters $1, 2, \ldots, k$ of M, hence $Ps(\Pi) = Ps(M)$. $\square$

Corollary 3.1 $NOP_n(obj_3, sym_*, anti_*) = NRE$, for all $n \geq 4$.

Proof. Counter machines with three counters characterize NRE, hence four membranes suffice in the previous proof. $\square$

4   What Can We Do with One or Two Objects?

The previous question remains as a research topic, as here we only have some partial answers to it.

First, let us observe that in the proof of Theorem 3.1 the object c is involved mainly in the simulation of the subtraction instruction, for the case of checking whether the counter is empty. In blind counter machines we do not have the check for zero (however, if we try to subtract from an empty counter the computation does not stop, but it aborts). The object c is also used as a "guardian" for using all copies of a in membranes $1, 2, \ldots, m$ and for ensuring that no a and no b waits unused in the skin region. We can avoid using c for these operations in the following way.

Consider the following "subsystem", called $trap(Q)$, where Q is a set of objects:

$$
\begin{aligned}
trap(Q) &= (Q, \{t, t'\}, [_t[_{t'} \ ]_{t'}]_t, \lambda, \lambda, -, R_t, R_{t'}, -), \\
R_t &= \{(q, in) \mid q \in Q\}, \\
R_{t'} &= \{(q, in), \ (q, out) \mid q \in Q\}.
\end{aligned}
$$

(The set of objects from the environment and the output membrane are not specified, as such a system will be used as a subsystem of a usual, complete, system.)

If any object q from Q will be free (not used by any other rules) in the region surrounding $trap(Q)$, then $trap(Q)$ will bring q inside region t and then q will oscillate forever across membrane t'.

Now, let us take the system Π from the proof of Theorem 3.1, and remove all elements related to the checking for zero of counters (we remove $(l_1, sub_{r=0})$ from J as well as all rules using such numbers) and all rules involving c (we remove c from O and all rules containing a c from all sets of rules). The object c is removed also from the initial contents of membrane 0. Then, let us place $trap(\{a, b\})$ in the skin region and $trap(\{a\})$ in all membranes $1, 2, \ldots, m$. The system Π' obtained in this way (we leave the task to write it formally to the reader) simulates the counter machine M in the case that M is blind: the correctly halting computations in M are correctly simulated by halting computations in Π'; if M aborts, trying to subtract from an empty counter, then Π' will reach a situation when a multiset $a^{v(l_1, sub_r > 0)}$ enters membrane r but cannot exit, because there is no b present in order to enable the associated rule 11; the copies of a will enter the membrane t from membrane r, and the computation will never finish.

Consequently, we have the following result (each counter corresponds to three membranes, while the skin contains a further trap-subsystem):

Theorem 4.1 *Any blind counter machine with m counters can be simulated by a P system with symport/antiport rules using only two objects, having $3(m + 1)$ membranes.*

The technique of the trap-subsystem can be used in order to obtain a much simpler proof of Theorem 3.1, but without making possible Corollary 3.1 (the number of membranes for generating NRE will be now 12), and without having elementary membranes as output regions. We give the construction and a brief description of its working in the Appendix.

Passing now to the case of unary P systems, we have a very limited knowledge about their power.

It is clear that such a system cannot have any rule of the form (a^i, in) associated with the skin membrane (the computation will never end), and no rule of the form (a^i, out) associated with the output membrane(s) (the only results will be smaller than i, hence a finite set).

It is also easy to compute arithmetical progressions. For instance, for $A(k_0, k) = \{k_0 + kn \mid n \geq 0\}$, we can consider the system

$$
\begin{aligned}
\Pi(k_0, k) &= (\{a\}, \{1, 2, 3\}, [_1[_2 \ ]_2[_3 \ ]_3]_1, a, a^{k_0}, \lambda, \{a\}, R_1, R_2, R_3, 2), \\
R_1 &= \{(a, out; aa, in)\}, \\
R_2 &= \{(a^k, in)\}, \\
R_3 &= \{(a, in)\}.
\end{aligned}
$$

The rule from R_1 brings more and more copies of a in the system, and these copies can either enter membrane 2, in "packages" of k copies, or membrane 3; this last membrane is also used for halting the computation, by bringing all copies of a here.

It is also easy (but not entirely trivial) to compute finite sets of numbers. For instance, for a set $F = \{n_1, \ldots, n_k\}$ such that $k \geq 1$ and $1 \leq n_1 < n_2 < \ldots < n_k$, let us denote $d_i = n_i - n_1 + 1$, $2 \leq i \leq k$, and consider the system

$$
\begin{aligned}
\Pi(F) &= (\{a\}, \{1, 2, 3\}, [_1[_2 \ [_3 \ ]_3]_2]_1, a^{d_k}, a, a^{n_1 - 1}, \{a\}, R_1, R_2, R_3, 3), \\
R_1 &= \{(a, out)\}, \\
R_2 &= \{(a, out; a^{d_i}, in) \mid 2 \leq i \leq k\}, \\
R_3 &= \{(a, in)\}.
\end{aligned}
$$

Every computation lasts one or two steps. If we start by using the rule (a, in) from R_3, then at the same time all copies of a from the skin region should exit the system and we stop with n_1 copies of a in membrane 3. If we do not use this rule in the first step, then we have to use one of the rules of R_2, non-deterministically chosen, hence d_i copies of a enter membrane 2, for some $2 \leq i \leq k$, and the copy of a from membrane 2 exits to the skin membrane; simultaneously, all remaining copies of a from the skin membrane are sent out of the system by rule (a, out) from R_1. Now, all d_i copies of a from membrane 2 should enter membrane 3, thus producing the number n_i, and the copy of a from the skin membrane exits (note that $d_i \geq 2$ for all $2 \leq i \leq k$, hence no rule $(a, out; a^{d_i}, in)$ from R_2 can be used in this step).

Can also finite sets containing number 0 be computed? What else can be said about unary systems?

5 Final Remarks

The number of objects used in a P system is a descriptional complexity measure of a clear interest, hence this issue is worth investigating in more details. At least the following directions of further research are natural:

1. Is Theorem 3.1 optimal in what concerns the number of objects? What one can say about the size and properties of families of sets of numbers computed by P systems with symport/antiport rules using only one or two objects, in addition to the few results from Section 4? We conjecture that unary systems can compute only semilinear sets of numbers, and that neither systems using two objects are universal. (Can a P system using two objects be simulated by a blind counter machine?)

2. Can the previous results be extended to other classes of P systems? Which is the number of objects sufficient in each case for obtaining the universality? The first candidates to consider are tissue P systems (where the membranes are placed in the nodes of a graph, see [6]), maybe with states associated to communication channels among cells, [4]. Do the additional "programming facilities" provided by these extensions of the tree-based P systems allow the decreasing of the number of objects? (In the case of channel states we also have to take into account the number of states, not to pay a too large price for improving on the number of objects.) Other candidates of interest are communicative P systems in the sense of [11].

3. Our proof uses symport/antiport rules of an arbitrary size, hence the question remains whether the universality can be reached by using rules of a bounded size (and a small number of objects), maybe using a larger number of membranes.

4. Are there "small universal counter machines", of the type of the small universal Turing machines as discussed in [10]? Can such a counter machine be used for obtaining "small universal symport/antiport P systems"?

5. We have considered here the case of generating numbers or vectors of numbers, hence the non-determinism cannot be avoided. What about considering the recognizing case? Can we preserve the determinism of the starting counter machine so that also our system is deterministic?

Appendix: A Simpler Proof of Theorem 3.1

We start again from a counter machine $M = (m, B, l_0, l_h, R)$ with m counters. With the same notations as in the proof of Theorem 3.1, we construct the system

$$
\begin{aligned}
\Pi &= (O, H, \mu, w_0, w_{t_0}, w_{t'_0}, w_1, w_{t_1}, w_{t'_1}, \ldots, w_m, w_{t_m}, w_{t'_m}, E, \\
&\qquad R_0, R_{t_0}, R_{t'_0}, R_1, R_{t_1}, R_{t'_1}, \ldots, R_m, R_{t_m}, R_{t'_m}, i_{o,1}, i_{o,2}, \ldots, i_{o,k}), \\
O &= \{a, b, c\}, \\
H &= \{0, 1, 2, \ldots, m, t_0, t'_0, t_1, t'_1, \ldots, t_m, t'_m\}, \\
\mu &= [_0[_{t_0}[_{t'_0}\ ]_{t'_0}]_{t_0}[_1[_{t_1}[_{t'_1}\ ]_{t'_1}]_{t_1}]_1 \cdots [_m[_{t_m}[_{t'_m}\ ]_{t'_m}]_{t_m}]_m]_0, \\
w_0 &= a^{v(l_0)}, \\
w_g &= \lambda, \text{ for all } g \in H - \{0\}, \\
E &= O,
\end{aligned}
$$

13

$$i_{o,j} = j, \ 1 \le j \le k,$$

with the sets of rules constructed as follows.

1. For each instruction $l_1 : (\mathtt{ADD}(r), l_2, l_3) \in R$, the set R_0 contains the rules

   1. $(a^{v(l_1)}, out; a^{v(l_1,add_r)}bcc, in)$,
   2. $(a^{v(l_1,add_r)}, out; a^{v(l_2)}, in)$, if $l_2 \ne l_h$,
   2'. $(a^{v(l_1,add_r)}, out)$, if $l_2 = l_h$,
   3. $(a^{v(l_1,add_r)}, out; a^{v(l_3)}, in)$, if $l_3 \ne l_h$,
   3'. $(a^{v(l_1,add_r)}, out)$, if $l_3 = l_h$,

   and the set R_r (corresponding to this instruction) contains the rules:

   4. $(a^{v(l_1,add_r)}bcc, in)$,
   5. $(a^{v(l_1,add_r)}, out)$.

2. For each instruction $l_1 : (\mathtt{SUB}(r), l_2, l_3) \in R$, we introduce the following rules in R_0

   6. $(a^{v(l_1)}, out; a^{v(l_1,sub_r>0)}cc, in)$,
   7. $(a^{v(l_1,sub_r>0)}b, out; a^{v(l_2)}, in)$, if $l_2 \ne l_h$,
   7'. $(a^{v(l_1,sub_r>0)}b, out)$, if $l_2 = l_h$,
   8. $(a^{v(l_1)}, out; a^{v(l_1,sub_r=0)}c, in)$,
   9. $(a^{v(l_1,sub_r=0)}, out; a^{v(l_3)}, in)$, if $l_3 \ne l_h$,
   9'. $(a^{v(l_1,sub_r=0)}, out)$, if $l_3 = l_h$,

   and the set R_r (corresponding to this instruction) contains the rules:

   10. $(a^{v(l_1,sub_r>0)}cc, in)$,
   11. $(a^{v(l_1,sub_r>0)}b, out)$,
   12. $(a^{v(l_1,sub_r=0)}c, in)$,
   13. $(a^{v(l_1,sub_r=0)}, out)$.

3. We also introduce the following rules in R_{t_0}

   14. $(a, in), \ (b, in), \ (c, in)$,

   and the following rules in $R_{t'_0}$

   15. $(a, in), \ (b, in), \ (c, in), \ (a, out), \ (b, out), \ (c, out)$.

4. Finally, for all $1 \le r \le m$ we introduce the following rules in R_{t_r}

16. (a, in), (bc, in), (cc, in), (ccc, in).

and the following rules in $R_{t'_r}$:

17. (a, in), (b, in), (a, out), (b, out).

The membranes t_0, t'_0 make sure that no object a, b, c stays any step unused in the skin region. This ensures the fact that all blocks of copies of a are always completely used. The same is true inside all membranes r with object a. This easily ensures the correct simulation of all rules associated with ADD instructions and with the case of > 0 guess for SUB instructions. Note that the rules associated with these cases bring in the system two copies of c, which are "stored" in the membranes t_r (c can enter such a membrane, but it cannot exit).

The case of the $= 0$ guess is now rather simple: we bring $a^{v(l_1, sub_{r=0})}c$ into the system, and then into membrane r; $a^{v(l_1, sub_{r=0})}$ exits immediately both from membrane r and from the system. If membrane r contains any copy of b, then the rule (bc, in) should be used and the computation will never stop, because b can oscillate forever across membrane t'_r. If no copy of b is present, then c remains unused. If the next instruction which uses the same counter r is an ADD one, then it brings two copies of c, which have to enter into membrane t_r together with the waiting copy of c by using the rule (ccc, in) (if we use the rule (cc, in), then the remaining copy of c will move inside a copy of b – there is at least one – and the computation never ends). If the next instruction is SUB and we take the > 0 guess, then again we bring two copies of c. If all three existing copies enters membrane t'_r, we return to a situation as at the beginning, with no c in region t_r. If only two copies of c are introduced into membrane t'_r, then the remaining copy must also enter provided that at least one b is present, hence the computation never ends. If no copy of b is present, hence the subtraction has emptied the counter, then c remains unused.

What remains to examine is the case when a copy of c is present in an empty counter (after simulating SUB in the case $= 0$, or, as above, after simulating SUB in the case > 0 without introducing c in membrane t_r). We consider again the next instruction to simulate on this counter. If it is ADD, or SUB and the guess is > 0, we have the same situation as above: either all c are stored in membrane t_r, or one remains, but this is possible only if the counter is empty. The unique case not considered is that when the next instruction is SUB and the guess is $= 0$. We arrive in membrane r with one c, hence immediately we can use the rule (cc, in) and the two copies of c are introduced in membrane t_r. Although we do not check whether any b is present, this is correct, because we know already that the membrane is empty.

The simulation of instructions is correct, we can stop if and only if we simulate a halting computation of the counter machine.

References

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter, *Molecular Biology of the Cell*, 4th ed. Garland Science, New York, 2002.

[2] A. Alhazov, M. Margenstern, V. Rogozhin, Y. Rogozhin, S. Verlan, Communicative P systems with minimal cooperation. In G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds., *Membrane Computing. International Workshop WMC5, Milan, Italy, 2004. Revised Papers, Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 2005 (to appear).

[3] R. Freund, A. Păun, Membrane systems with symport/antiport rules: universality results. In Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds., *Membrane Computing. International Workshop, WMC 2002, Curtea de Argeş, Romania. Revised Papers, Lecture Notes in Computer Science*, 2597, Springer-Verlag, Berlin, 2002, 270–287.

[4] R. Freund, Gh. Păun, M.J. Pérez-Jiménez, Tissue-like P systems with channel-states. *Brainstorming Week on Membrane Computing*, Sevilla, February 2004, TR 01/04 of Research Group on Natural Computing, Sevilla University, 2004, 206–223, and *Theoretical Computer Science*, 2004, in press.

[5] S.A. Greibach, Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7 (1978), 311–324.

[6] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón, Tissue P systems. *Theoretical Computer Science*, 296, 2 (2003), 295–326.

[7] M. Minsky, *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.

[8] A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3 (2002), 295–306.

[9] Gh. Păun, *Computing with Membranes – An Introduction*. Springer-Verlag, Berlin, 2002.

[10] Y. Rogozhin, Small universal Turing machines. *Theoretical Computer Science*, 168 (1996), 215–240.

[11] P. Sosik, J. Matysek, Membrane computing: when communication is enough. In C.S. Calude, M.J. Dinneen, F. Peper, eds., *Unconventional Models of Computation 2002, Lecture Notes in Computer Science*, 2509 Springer-Verlag, Berlin, 2002, 264–275.

[12] G. Vaszil, On the size of P systems with minimal symport/antiport. *Pre-Proceedings of Workshop on Membrane Computing, WMC5, Milano, Italy*, June 2004, 422–431.