P Systems with Symport/Antiport and Time

Hitesh Nagda[1], Andrei Păun[1,2], and Alfonso Rodríguez-Patón[2]

[1] Department of Computer Science/IfM, Louisiana Tech University
P.O. Box 10348, Ruston, LA 71272, USA
{hhn002, apaun}@latech.edu
[2] Universidad Politécnica de Madrid - UPM, Facultad de Informática
Campus de Montegancedo S/N, Boadilla del Monte, 28660 Madrid, Spain
arpaton@fi.upm.es

Abstract. We consider symport/antiport P systems using time as the output of a computation. We describe and study the properties of "timed symport/antiport systems", showing that considering the time as support for reporting the result of computation yields more power/flexibility to these systems. We were able to improve or match the best results concerning the symport/antiport systems which consider the output as originally defined as the number of molecules found in a pre-defined elementary membrane in the halting configuration of the system.

1 Introduction

We continue the work on symport/antiport P systems [5], [6], [11], [1], [2] using the paradigm of time as the output of a computation as previously introduced in [4] and [8]. The idea originates in [12] as Problem W; the novelty is that instead of the "standard" way to output, like the multiplicities of objects found at the end at the computation in a distinguished membrane as it was defined in the model from [11], it seems more "natural" to consider certain *events* (i.e., configurations) that may occur during a computation and to relate the output of such a computation with the time interval between such distinguished configurations. Our system will compute a set of numbers like in the case of "normal" symport/antiport [11], but the benefit of the current setting is that the computation and the observance of the output are now close to the biology and to the tools used for cell biology (fluorescence microscopy, FACS). The model of the "timed" P system that we investigate here is the symport/antiport P system. We note that such a "timed" approach could be applied also to other types of P systems. Actually the spiking neural P systems ([9], [14]) use a similar idea: the output of such a system is the time elapsed between two spikes of a pre-defined "output" neuron. Going back to the symport/antiport model, we are studying another way of viewing the output of such a system; the motivation comes from the fact that cells can become fluorescent if, for example, some types of proteins with fluorescence properties are present in the cells. Such a fluorescent "configuration" of a cell will be the configuration that starts the clock used for the output. Even more interesting (making our definition a very natural way of viewing the

output of a system) is the fact that there are tools currently used by researchers in cell biology that can detect the fluorescence of each cell individually. Such an automated technique for viewing the output of a computation using cells is highly desirable since it holds the promise of fast readouts of the computations.

2 Timed Symport/Antiport Systems

We will use a modified definition than the one in [11]; instead of specifying the output region where the result of the computation will be "stored" in a halting computation, we specify two relations C_{start} and C_{stop} (which are described by regular languages) that need to be satisfied by the multisets of objects in the membrane structure at two different times during the computation. We restrict the description of C_{start} and C_{stop} to regular languages, each word representing a possible configuration of the system that satisfies the respective relation due to two main reasons: first we want to use the idea of restriction in the *start/stop* configurations. In this way we make sure that some "artificial" constructs (for example encoding the whole RE set into the configurations) are not possible, and on the other hand, the regular languages express enough for the *start/stop* configurations to help in obtaining similar universality results as for "regular" systems with symport/antiport.

An important observation is the fact that we will not require the cell to "stop working" when reaching the result; i.e., we will not require the strong restriction that the system reaches a halting configuration for a computation to have a result. It is worth noting that a similar idea of "configuration-based output" was considered also in [7] where the authors pre-defined in the system two membranes: *fin* and *ack*; initially *ack* is empty, but once it receives an element, the number of objects hat are present in that moment in *fin* is the output of the computation. In this way one can consider the output of non-halting computations in that case as well. Of course, one can immediately see that in [7] the result is still encoded as multiplicities of different molecules, in our framework, we use the time between configurations to encode the result of the computation.

Before progressing further we give some basic notions used in the remainder of the paper; the language theory notions used but not described are standard, and can be found in any of the many monographs available, for instance, in [15].

A membrane structure is pictorially represented by a Venn diagram, and it will be represented here by a string of matching parentheses.

A multiset over a set X is a mapping $M : X \longrightarrow \mathbf{N}$. Here we always use multisets over finite sets X (that is, X will be an alphabet). A multiset with a finite support can be represented by a string over X; the number of occurrences of a symbol $a \in X$ in a string $x \in X^*$, denoted by $|x|_a$, represents the multiplicity of a in the multiset represented by x. Clearly, all permutations of a string represent the same multiset, and the empty multiset is represented by the empty string, λ.

We will use symport rules of the form (ab, in) and (ab, out), associated with a membrane and stating that the objects a, b can enter, respectively, exit the

membrane together, and antiport rules of the form $(a, out; b, in)$, stating that a exits and at the same time b enters the membrane.

Based on rules of these types, we modify the definition from [11] to introduce the model of a *timed symport/antiport P system* as a construct,

$$\Pi = (O, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, C_{start}, C_{stop}),$$

where:

- $O = \{a_1, \ldots, a_k\}$ is an alphabet (its elements are called *objects*);
- μ is a membrane structure consisting of m membranes, with the membranes (and hence the regions) bijectively labeled with $1, 2, \ldots, m$; m is called the *degree* of Π;
- $w_i, 1 \leq i \leq m$, are strings over O representing multisets of objects associated with the regions $1, 2, \ldots, m$ of μ, present in the system at the beginning of a computation;
- $E \subseteq O$ is the set of objects that are continuously available in the environment in arbitrarily many copies;
- $R_1, \ldots, R_m$ are finite sets of symport and antiport rules over the alphabet V associated with the membranes $1, 2, \ldots, m$ of μ;
- C_{start} and C_{stop} are regular subsets of $(O^*)^m$, describing configurations of Π. We will use a regular language over $O \cup \{\$\}$ to describe them, the special symbol $\$ \notin O$ being used as a marker between the configurations in the different regions of the system. More details will be given in the following.

For a symport rule (x, in) or (x, out), we say that $|x|$ is the *weight* of the rule. The *weight* of an antiport rule $(x, out; y, in)$ is $\max\{|x|, |y|\}$. The rules from a set R_i are used with respect to membrane i as explained above. In the case of (x, in), the multiset of objects x enters the region defined by the membrane, from the surrounding region, which is the environment when the rule is associated with the skin membrane. In the case of (x, out), the objects specified by x are sent out of membrane i, into the surrounding region; in the case of the skin membrane, this is the environment. The use of a rule $(x, out; y, in)$ means expelling the objects specified by x from membrane i at the same time with bringing the objects specified by y into membrane i. The objects from E appear in arbitrarily many copies in the environment. The rules are used in the non-deterministic maximally parallel manner specific to P systems with symbol objects: in each step, a maximally parallel multiset of rules is used.

In this way, we obtain transitions between the configurations of the system. A configuration is described by the m-tuple of multisets of objects present in the m regions of the system, as well as the multiset of objects from $O - E$ which were sent out of the system during the computation. It is important to note that such objects appear only in a finite number of copies in the initial configuration and can enter the system again (knowing the initial configuration and the current configuration in the membrane system, one can know precisely what "extra" objects are present in the environment). On the other hand, it is not necessary to take care of the objects from E which leave the system because

they appear in arbitrarily many copies in the environment as defined before (the environment is supposed to be inexhaustible, irrespective how many copies of an object from E are introduced into the system, still arbitrarily many remain in the environment). The initial configuration is $\alpha_0 = (w_1, \ldots, w_m)$.

Let us now describe the way this systems "outputs" the result of its computation: when the system enters some configuration α from C_{start} (we also say that C_{start} is satisfied), we start a counter t that is incremented each time the symport/antiport rules are applied in the nondeterministic parallel manner. At some point, when the system enters some configuration β from C_{stop} (hence C_{stop} is satisfied), we stop incrementing t, and the value of t represents the output of the computation[3]. If the system never reaches a configuration in C_{start} or in C_{stop}, then we consider the computation unsuccessful, no output is associated with it. The set of all such t's (computed as described) is denoted by $N(\Pi)$. The family of all sets $N(\Pi)$ computed by systems Π of degree at most $m \geq 1$, using symport rules of weight at most p and antiport rules of weight at most q, is denoted by $NTP_m(sym_p, anti_q)$ (we use here similar notations as the ones from [11] and [2]).

We emphasize the fact that in the definition of Π we assume that C_{start} and C_{stop} are regular. Other, more restrictive, cases can be of interest but we do not discuss them here.

Details about P systems with symport/antiport rules can be found in [11]; a complete formalization of the syntax and the semantics of these systems is provided in [13] where reachability of symport/antiport configurations was discussed.

3 Register Machines and Counter Automata

In the proofs from the next sections we will use register machines and counter automata as devices characterizing NRE, hence the Turing computability.

Informally speaking, a register machine consists of a specified number of registers (counters) which can hold any natural number, and which are handled according to a program consisting of labeled instructions; the registers can be increased or decreased by 1 – the decreasing being possible only if a register 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) *register 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 labeled (hence uniquely identified) by elements from B (R is also called the *program* of the machine). The labeled instructions are of the following forms:

- $l_1 : (\texttt{ADD}(r), l_2, l_3)$, $1 \leq r \leq m$ (increment the value of the register r and then jump non-deterministically to one of the instructions with labels l_2, l_3),

[3] By convention, in the case when a configuration α is reached that satisfies both C_{start} and C_{stop}, then we consider that the system has computed the value 0.

- $l_1 : (\texttt{SUB}(r), l_2, l_3)$, $1 \leq r \leq m$ (if register 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 register machine generates a natural number in the following manner: we start computing with all m registers being empty, with the instruction labeled by l_0; if the computation reaches the instruction $l_h : \texttt{HALT}$ (we say that it halts), then the values of register 1 is the number generated by the computation. The set of numbers computed by M in this way is denoted by $N(M)$.

We recall also the definition of the counter automaton; for more details we refer the interested to the literature [10]. Such a device is a construct $M = (m, Q, q_0, q_f, P)$, where d is the number of counters, $Q = \{q_0, \ldots, q_f\}$ is the set of states of the machine, q_0 is the start state while q_f is the final state and P is the set of instructions, of three types:

- $(p \to q, c+)$ with $p, q \in Q$ and c a counter. This instruction will increment the value of the register c and move from state p in state q.
- $(p \to q, c-)$ with $p, q \in Q$ and c a counter. The instruction tries to decrement the value of the counter c; if it was originally greater than zero, then it is decremented and M moves to the state q, otherwise (if the value stored in c is zero) the computation is stopped and the machine does not produce output.
- $(p \to q, c = 0)$ with $p, q \in Q$ and c a counter. The instruction tests the value of the counter c; if it is zero, then M moves to state q, otherwise the computation stops and the machine does not produce an output.

It is known (see [10]) that non-deterministic register machines and counter automata generate exactly the family NRE, of Turing computable sets of numbers.

4 Universality Results for Timed P Systems Having Only one Membrane

The first result given here is related to the results obtained in [2] and [6] where it is proved that systems using only one membrane and symport rules of size 3 are universal. We need to mention that in our setup (by using the time as the output of the computation) we are able to generate all the subsets of NRE including the ones containing the values from 0 to 7, which is not the case of the aforementioned results where there are some "garbage" symbols left in the output region. Another remark that should be made is that rather than using the more complicated notion of conflicting counters in a register machine, we use here a proof which is easier to understand and, implicitly, easier to implement.

Theorem 1. *P systems with time are universal for one membrane and symport of length 3 even when no antiport is used:* $NRE = NTP_1(sym_3, anti_0)$

Proof. We consider a register machine $M = (m, B, l_0, l_h, R)$ and we construct the system

$$\Pi = (O, [_1\ ]_1, w_1, E, R_1, C_{start}, C_{stop})$$

with the following components

$$\begin{aligned}
O &= \{a_r \mid 1 \leq r \leq m\} \cup \{P_l, P'_l, P''_l, Q_l, Q'_l, X_l, X'_l, X''_l, l \mid l \in B\} \cup \{b, t\},\\
w_1 &= l_0 P_1 \dots P_l Q_1 \dots Q_l X_1 \dots X_l X''_1 \dots X''_l b^2,\\
E &= \{a_r \mid 1 \leq r \leq m\} \cup \{P'_l, P''_l, Q'_l, X'_l, l \mid l \in B\} \cup \{t\},\\
C_{start} &= \{b^2 t^2 w_1 \mid w_1 \in (O - \{b, t\})^*\},\ \text{in other words, } b \text{ and } t \text{ appear exactly}\\
&\quad \text{two times, and the rest of the symbols can appear in any numbers.}\\
C_{stop} &= \{t^i w_2 \mid i \geq 1, w_2 \in (O - \{a_1\})^*\},\ \text{in this case we have that } t \text{ appears at}\\
&\quad \text{least once, while } a_1 \text{ does not appear in the region.}
\end{aligned}$$

and the following rules in R_1:

1. For an ADD instruction $l_1 : (\texttt{ADD}(r), l_2, l_3) \in R$, we consider the rules

   $(P_{l_1} l_1, out),\ (P_{l_1} l_2 a_r, in),\ (P_{l_1} l_3 a_r, in).$

   We simulate the work of the ADD instruction in two steps. First we send out the current instruction label together with the marker P_{l_1} that will come back in the membrane with two other objects, a copy of a_r so that the register r is incremented and also the new instruction label. To simulate the non-deterministic behavior of these machines we have two symport rules that do the same job, the only difference being the next instruction label being brought back in the system. It is clear that the simulation of the ADD instruction is performed correctly.
2. For a SUB instruction $l_1 : (\texttt{SUB}(r), l_2, l_3) \in R$ we consider the following rules:

   $(P_{l_1} l_1, out),\ (P_{l_1} P'_{l_1} P''_{l_1}, in),\ (P'_{l_1} Q_{l_1}, out),\ (P''_{l_1} X_{l_1} a_r, out),\ (Q_{l_1} Q'_{l_1}, in),$
   $(X_{l_1} X'_{l_1} l_2, in),\ (Q'_{l_1} P''_{l_1} X''_{l_1}, out),\ (Q'_{l_1} X'_{l_1}, out),\ (X''_{l_1} l_3, in).$

   We simulate the work of the SUB instruction in several steps (5 if the register is not empty and 6 if it is empty). We first send out the current label with its corresponding P marker by the rule $(P_{l_1} l_1, out)$. At the next step the symbol P brings in two more symbols that keep track of the instruction being simulated with theirs indices: $(P_{l_1} P'_{l_1} P''_{l_1}, in)$, P' is working as a timer while P'' is checking whether the register is empty or not. If the register is not empty, then P'' will exit decreasing the register and taking at the same time another marker to the outside to help identify the correct case later: $(P''_{l_1} X_{l_1} a_r, out)$. At the next stage X will return with yet another marker and the next instruction label to be brought in (in this case l_2 as the register was not empty), $(X_{l_1} X'_{l_1} l_2, in)$. The work is finished in this case by the rule $(Q'_{l_1} X'_{l_1}, out)$.
   If the register is empty, we perform the same initial steps, sending P and the current instruction label out, P returns with P' and P''; this time P'' cannot exit the membrane at the next step since the register is empty, but P'

is exiting together with Q, then Q returns with Q'. At the next step we have the "branching point": rather than exiting with X' (which will be present in the membrane in the case when the register was not empty), Q' exits with P'' and X''. If X'' exits, that means that the register was empty, thus when X'' returns in the system, it returns with the label of the next instruction to be simulated as l_3.

3. The terminating/counting work is done by the rules:

   $(l_h b^2, out),\ (bt, in),\ (bta_1, out)$.

   It is clear that at the end of the simulation, if the register machine has reached the final state, we will also have the halting instruction symbol in the system membrane. At that time we will have the computed value encoded as the multiplicity of the object a_1 that is associated with the output register. We will also have in the system the label of the halting instruction, l_h, thus the rule $(l_h b^2, out)$ can be applied only when the simulation was performed correctly. At the next step, the two b-s return with two copies of t, satisfying the C_{start} configuration. One can note that if there are no copies of a_1 in the membrane, then also the configuration C_{stop} is satisfied at the same time, thus our system would compute the value 0 in that case. For any even value encoded in the multiplicity of a_1 it will take exactly half that number of steps for the two copies of the pair bt to push the a_1-s out of the membrane and again the same amount to return to the membrane.
   Let us give a small example: for the value 4, the first step 2 copies of a_1 are pushed out, and at the next step the symbols b^2t^2 return in the membrane; in two more steps we will have 0 copies of a_1 and at least one copy of t, so the whole process took the correct 4 steps to complete.
   For an odd number we perform the same work, with the exception of the last step, when there is only one copy of a_1 in the membrane. At that moment, only one bt can exit, leaving the second one in the membrane, satisfying the C_{stop} condition at the correct time. □

In the following we recall a proof from [8] due to its relevance to the current paper. The best known result for "standard" symport/antiport P systems in this setting is $N_1RE = N_1OP_1(sym_0, anti_2)$, thus the following result improves the best known result for symport/antiport systems by being able to generate sets of numbers containing also the values 0 and 1. Another observation is that the $C_{start},\ C_{stop}$ configuration are in this case quite simple. We call them having minimal restrictions on multiplicities; we mean by this the fact that for each object and each membrane, the $C_{start},\ C_{stop}$ rules will impose either a fixed multiplicity or not impose any restrictions for the object.

Theorem 2. *Using minimal restrictions on the multiplicities of the objects for the $C_{start},\ C_{stop}$ rules we have $NRE = NTP_1(sym_0, anti_2)$.*

Proof. To prove the theorem we follow the constructions from the literature for the "standard" symport/antiport P systems, this time using counter automata. In the initial configuration, the unique membrane contains the start state as

its only object. The work of the counter automaton can be simulated using the antiport rules in the following way.

For a rule $(p \to q, \lambda) \in R$ we will have in our timed P system the rule $(q, in; p, out)$; for an increment instruction $(p \to q, i+)$ on the counter c_i we will add the antiport rule $(qc_i, in; p, out)$ to R_1. The decrement instruction can only be applied if the counter is non zero: $(p \to q, i-)$ is simulated by $(q, in; pc_i, out)$. Finally, $(p \to q, i = 0)$ is simulated by the rules $(q'\overline{i}, in; p, out)$; $(\infty, in; \overline{i}c_i, out)$, $(q'', in; q', out)$, and $(q, in; q''\overline{i}, out)$ in three steps: first we replace p by q' and $\overline{i}$, then $\overline{i}$ checks whether the register i is empty or not; if nonempty, the special marker ∞ is brought in and the computation cannot continue; in the case when the register was empty the computation can continue by expelling the two symbols q'' and $\overline{i}$ together to bring in the next state q.

It is clear now that the register machine is simulated in this way only by using antiport rules of weight[4] 2. When the final state appears as the current state of the simulation it is time to start "counting" the result. We define $C_{start} = \{wf \mid w \in (O - \{f, \infty\})^*\}$. The rule $(f, in; fc_0, out)$ will expel one symbol c_0 at a time, thus if we define $C_{stop} = \{fw' \mid w' \in (O - \{f, c_0\})^*\}$, we will have exactly i steps between C_{start} and C_{stop}, where i is the multiplicity of the symbol c_0 (i.e., the contents of the output register) in the system. Following the previous discussion the equality $NRE = NTP_1(sym_0, anti_2)$ was shown, which completes the proof. □

5 Universality Results for Timed P Systems Having Two Membranes

In this section we will provide two dual results with the ones presented in [1], [2] when considering systems with two membranes. It is worth noting that in [1] the authors use and intersection with a finite alphabet when defining the result of a computation. The best result from [2] is $N_3RE = N_3OP_2(sym_1, anti_1)$. Here we improve this result in the sense that we generate also the sets of numbers containing the values 0 through 3. We will give in the following theorem the C_{start}/C_{stop} configurations in the form <multiset for membrane 1>$<multiset for membrane 2> described by regular languages as defined in the second section of the paper; in the first two proofs we showed universality of a single region, thus the symbol $ was not used.

Theorem 3. $NRE = NTP_2(sym_1, anti_1)$.

Proof. We will follow the construction from [1] and note the changes made. For a detailed explanation of the work of the system we refer the interested reader to [1].

[4] The result can be strengthened in the following way: the construction works even if we only use antiport rules of dimensions $(1, 2)$ or $(2, 1)$ by adding to the only two rules of dimension $(1, 1)$ some padding symbols. For example the rule $(q'', in; q', out)$ can be padded with the extra symbol P in this way $(q''P, in; q', out)$.

Let us consider a counter automaton $M = (m, Q, q_0, q_f, P)$ which starts with empty counters and has n instructions in the set P.

We construct the P system $\Pi = (O, [_1[_2\,]_2]_1, w_1, w_2, E, R_1, R_2, C_{start}, C_{stop})$ with the following components

$$\begin{aligned}
O &= E \cup \{b_j, b'_j \mid 1 \le j \le n\} \cup \{\#, F, I\},\\
w_1 &= q_0 IF\#\#,\\
w_2 &= b_1 b_2 \dots b_n b'_1 b'_2 \dots b'_n dd,\\
E &= Q \cup \{c_r \mid 1 \le r \le m\} \cup \{a_j, a'_j, a''_j \mid 1 \le j \le n\} \cup \{z\},\\
C_{start} &= \{d\#^2 w_1 \$ w_2 \mid w_1 \in (O - \{d, \#\})^*, w_2 \in (O - \{d\})^*\},\\
C_{stop} &= \{z w_3 \$ w_4 \mid w_3 \in O^*, w_4 \in (O - \{c_1\})^*\}.
\end{aligned}$$

The definition of C_{start} means that for starting to count we need to reach a state when exactly one copy of d and two copies of $\#$ are present in membrane 1, at the same time as no copy of d is present in membrane 2. At the same time, C_{stop} is only satisfied when at least one copy of the symbol z is present in membrane 1 and no copies of c_1 appears in membrane 2.

Let us now define the rules from R_1 and R_2:

$$\begin{aligned}
R_1 &= R_1^{ini} \cup R_1^{sim} \cup R_1^{timer}, \text{ and}\\
R_2 &= R_2^{ini} \cup R_2^{sim} \cup R_2^{timer}, \text{ where}\\
R_1^{ini} &= \{(I, out; c_k, in) \mid 1 \le k \le m\} \cup \{(I, in)\},\\
R_2^{ini} &= \emptyset,\\
R_1^{sim} &= \{(q_i, out; a_j, in), (a''_j, out; q_l, in) \mid (j : q_i \to q_l, .) \in P\}\\
&\quad \cup \{(b_j, out; a'_j, in), (a_j, out; b_j, in), (\#, out; b_j, in) \mid 1 \le j \le n\}\\
&\quad \cup \{(a'_j, out; a''_j, in) \mid \text{ where } j \text{ is the label of an increment or decrement}\\
&\qquad \text{instruction}\} \cup \{(\#, out; \#, in)\}\\
&\quad \cup \{(b'_j, out; a''_j, in), (a'_j, out; b'_j, in) \mid (j : q_i \to q_l, k = 0) \in P\}\\
&\quad \cup \{(\#, out; b'_j, in) \mid (j : q_i \to q_l, k = 0) \in P\}\\
R_2^{sim} &= \{(b_j, out; a_j, in) \mid 1 \le j \le n\}\\
&\quad \cup \{(a_j, out; c_k, in), (a'_j, in) \mid (j : q_i \to q_l, k+) \in P\}\\
&\quad \cup \{(a'_j, out; b_j, in) \mid j \text{ labels an increment or decrement instruction}\}\\
&\quad \cup \{(a_j, out) \mid j \text{ labels a decrement or test with 0 instruction}\}\\
&\quad \cup \{(c_k, out; a'_j, in) \mid (j : q_i \to q_l, .) \in P\}\\
&\quad \cup \{(b'_j, out; b_j, in), (b'_j, in) \mid (j : q_i \to q_l, k = 0) \in P\},\\
R_1^{timer} &= \{(d, out; z, in)\},\\
R_2^{timer} &= \{(d, out; q_f, in), (q_f, out; F, in), (c_1, out; z, in), (z, out)\}.
\end{aligned}$$

The simulation of the counter automaton is done in phases. The rules from the *ini* phase bring in membrane 1 an arbitrary number of objects c_i for any register i. This helps with the simulation of the increment instruction in the

automaton. The rules in the *sim* group perform the actual simulation of the different instructions in the automaton (increment, decrement, test for 0). We refer the interested reader to [1] for the details of the construction and the proof of correctness. We change the construction in the finishing part of the simulation to match the output based on the time that passes between two configurations. For this we note that at the start of each simulation step we have a symbol codifying the current state in the automaton q_i in membrane 1, whereas membrane 2 holds the current values of the counters codified by multiplicities of the symbols c_j.

After a successful simulation, we will reach a state when q_f is in membrane 1 and some number of c_1 objects that are present in membrane 2 codify the output of the system. At this moment we can use the rules from the set *timer*. We will give step-by-step explanations for this phase.

At step 1 q_f is in membrane 1 and dd in membrane 2. By using the rule $(d, out; q_f, in)$ we will now have d in membrane 1 and dq_f in membrane 2. At the next step d from membrane 1 is replaced by a z using the rule $(d, out; z, in)$, and at the same time q_f returns in membrane 1 by the rule $(q_f, out; F, in)$. At the next step q_f will remove the second d from membrane 2, thus satisfying the C_{start} condition. There are two cases: a) the output register was empty and b) the output register is not empty.

In the case a) we have the following configuration: in membrane 1 we have zd and in membrane 2 we have $q_f F$ and no copies of c_1; this configuration that satisfied C_{start} will satisfy also C_{stop}, thus the value computed is correctly reported as 0.

Case b) if there was at least on copy of c_1 in membrane 2, then the rule $(c_1, out; z, in)$ is applicable, so now membrane 1 has d and membrane 2 has $zq_f F$, and this satisfies C_{start}. At the next moment the symbol d will be replaced by a second z in membrane 1, while the first z returns to membrane 1 by the rule (z, out). If the value of the output counter was 1, then there are no more copies of c_1 in membrane 2, thus the configuration satisfies at this step C_{stop}, thus correctly computing the value 1. If the register stored a value more than 2, then the computation continues in a homogenous fashion from now on: the two copies of z will expel each a copy of the c_1 marker and at the next step return to membrane 1. If the amount of objects c_1 was odd, then the computation finishes with the two symbols z in membrane 1 and no symbols c_1 in membrane 2, thus reaching the C_{stop} in a correct amount of time. On the other hand, if the output of the computation was an even value, then one of the z symbols will be swapped with the last c_1, while the second z will remain in membrane 1, making the C_{stop} satisfiable, thus computing also in this case the correct number of steps. This concludes the proof. □

We will now proceed to prove the last result of the paper which still deals with universality of timed symport/antiport P systems. We prove that two membranes and symport of size 2 are enough for generating all the NRE sets. The best result for the case of symport/antiport systems with output in an elementary membrane is given in [2] where the authors show that such systems with two

membranes and symport of size 2 are universal, but cannot generate sets of numbers containing the values 0 through 6. In the case of systems based on time, we match the universality result of systems with two membranes and symport of size two, but we are also able to generate the sets of numbers containing the values 0 through 6.

Theorem 4. $NRE = NTP_2(sym_2, anti_0)$.

Proof. We will follow again the construction from [1] and note the changes made to it.

Let us consider as in the proof of Theorem 3 a counter automaton $M = (m, Q, q_0, q_f, P)$ which starts with empty counters and has n instructions. We construct the P system $\Pi = (O, [\,_1[\,_2\,]_2]_1, w_1, w_2, E, R_1, R_2, C_{start}, C_{stop})$ with the following components:

$$
\begin{aligned}
O &= E \cup Q \cup \{b_j, g_j \mid 1 \le j \le n\} \cup \{g'_j \mid 1 \le j \le n-1\} \cup \{\#, \$, F, y\},\\
w_1 &= q_0 a_1 F \$ b_1 b_2 \dots b_n,\\
w_2 &= \# q_1 q_2 \dots q_f g_1 g_2 \dots g_n g'_1 g'_2 \dots g'_{n-1} y y,\\
E &= \{c_r \mid 1 \le r \le m\} \cup \{a_j, a'_j, d_j, d'_j \mid 1 \le j \le n\},\\
C_{start} &= \{y^2 w_1 \$ \# w_2 \mid w_1 \in (O - \{y\})^*, w_2 \in (O - \{\#\})^*\},\\
C_{stop} &= \{y w_3 \$ w_4 \mid w_3 \in (O - \{c_1\})^*, w_4 \in O^*\}.
\end{aligned}
$$

The rules from R_1 and R_2 are as follows:

$$
\begin{aligned}
R_1 &= R_1^{sim}, \text{ and}\\
R_2 &= R_2^{sim} \cup R_2^{timer}, \text{ where}\\
R_1^{sim} &= \{(q_i a_j, in) \mid (j : q_i \to q_l, .) \in P\}\\
&\quad \cup \{(b_j g_j, out) \mid j \text{ is the label of a increment or zero check}\\
&\quad \cup \{(c_k b_j, in) \mid (j : q_i \to q_l, k+) \in P,\ q_i, q_l \in Q,\ 1 \le k \le m\}\\
&\quad \cup \{(c_k g_j, out) \mid (j : q_i \to q_l, k-) \in P,\ q_i, q_l \in Q,\ 1 \le k \le m\}\\
&\quad \cup \{(a'_j g_j, in) \mid 1 \le j \le n-1\} \cup \{(\#, out), (\#, in)\}\\
&\quad \cup \{(d_j b_j, in), (d_j c_k, out), (a'_j g'_j, out) \mid (j : p \to q, k = 0) \in P\}\\
&\quad \cup \{(d'_j g'_j, in), (d'_j, out) \mid (j : p \to q, k = 0) \in P\}\\
&\quad \cup \{(a'_j q_l, out) \mid (j : q_i \to q_l, k+) \in P \text{ or } (j : q_i \to q_l, k-) \in P\}\\
&\quad \cup \{(d_j q_l, out) \mid (j : q_i \to q_l, k = 0) \in P,\ 1 \le k \le m\},\\
R_2^{sim} &= \{(a_j b_j, in), (b_j g_j, out), (a'_j g_j, in) \mid 1 \le j \le n-1,\ (j : q_i \to q_l, .) \in P\}\\
&\quad \cup \{(q_i, in) \mid q_i \in Q\} \cup \{(a'_j \$, in) \mid 1 \le j \le n\} \cup \{(\#\$, out)\}\\
&\quad \cup \{(a'_j q_l, out) \mid (j : q_i \to q_l, k+) \in P \text{ or } (j : q_i \to q_l, k-) \in P\}\\
&\quad \cup \{(a'_j g'_j, out), (d'_j g'_j, in), (d_j q_l, out) \mid\\
&\qquad (j : q_i \to q_l, k = 0) \in P,\ 1 \le k \le m\}\\
R_2^{timer} &= \{(q_f F, in), (q_f y, out), (F y, out), (y c_1, in)\}.
\end{aligned}
$$

The simulation of the register machine is done by the rules in the *sim* groups. We refer the interested reader to [1] for the details of the construction and the proof of correctness. We note that the construction from [1] was changed for our purposes, thus the rules from the group *start* as they were defined in the Theorem 2 in [1] are no longer needed since we do not need the "safety net" of the computation running forever (done by using the symbol $\#_2$) if the simulation is blocked. This is due to the flexibility of the C_{start}/C_{stop} configurations. We note that the construction is simulating the work of the counter automaton in membrane 1 and use membrane 2 as a filtering mechanism. Membrane 1 contains both the values of the counters (codified as multiplicities of objects c_i), the current state of the automaton (codified as an object q_j) and the next instruction to be executed (codified as a symbol a_k). At the end of a successful simulation we will have q_f in membrane 1, some number of objects c_1 codifying the output of the simulation and # in membrane 2 (if # has reached membrane 1, then the simulation was not correct). The rules in the *timer* group can only be applied after the simulation was completed successfully, thus the symbol q_f appears in membrane 1. At that time q_f together with F enter membrane 2 by rule $(q_f F, in) \in R_2^{timer}$, where they are able to "pair" with a copy of y each and exit to membrane 1, by using the rules $(q_f y, out), (Fy, out) \in R_2^{timer}$. At this moment we will have again q_f and F in membrane 1, and also y^2 in the same membrane. It is easy to see that the C_{start} is satisfied now since the two carrier symbols y are in membrane 1. If the value in register 1 is zero, then the C_{stop} is also valid producing the correct output for this case. If the value computed by the automaton is non-zero, then the objects y will start moving copies of c_1 in membrane 2, and return to membrane 1 using the symbols q_f and F. This is done up until all the symbols c_1 have been moved in region 2. We have two cases for this process: a) the number computed by the automaton is even or b) the number computed is odd.

In the case a) one can notice that it takes two steps for a symbol y to return to membrane 1, but since there are exactly two such symbols in the system, for every two steps of the system, two c_1-s are moved to region 2 and the two y symbols return to membrane 1. Thus in the same amount of steps as the number computed, the configuration C_{stop} becomes satisfied – to this aim, it needs at least one copy of y present in membrane 1, so we need in this case to wait until both y-s return to membrane 1 after the c_1-s are depleted. In the case b) one can note that in some even number of steps, $2s$ for example, exactly $2s$ copies of c_1 are moved to region 2 and the y-s return to membrane 1, thus without loss of generality we can assume that in $2s$ steps we only have one more copy of c_1 in membrane 1. In that moment, one of the y-s will move the c_1 to region 2, while the other copy of y cannot move, thus at the next step we reach a configuration from C_{stop}, and the system outputs correctly the value $2s + 1$. □

In this way we matched/improved all the four best results known for the symport/aniport P systems. This shows that considering the time as the support for outputting the result for these systems is both powerful and, as described in the introduction, motivated from the bio-molecular tools.

6 Final Remarks

For the newly introduced timed P systems we improved or matched the four best known results for "regular" symport/antiport P systems. It is worth noting that the new feature of outputting the result using time is more flexible than the previously considered methods, thus the previous results could be even improved by using completely different techniques that take advantage of the flexibility of the time as a framework of outputting the result of a computation. For example in the new framework we no longer have the (rather strong) requirement that the computation should halt, only to reach a configuration from C_{start} and then one from C_{stop}. We conjecture that this new definition could prove useful also in conjunction with classes of symport/antiport systems that using the original definition could only generate finite sets (e.g., generate some non-finite family of numbers, etc.).

Acknowledgments

A. Păun gratefully acknowledges the support in part by LA BoR RSC grant LEQSF (2004-07)-RD-A-23 and NSF Grants IMR-0414903 and CCF-0523572.

References

1. A. Alhazov, R. Freund, Yu. Rogozhin, Some Optimal Results on Symport/Antiport P Systems with Minimal Cooperation, M.A. Gutiérrez-Naranjo et al. (eds.), Cellular Computing (Complexity Aspects), ESF PESC Exploratory Workshop, Fénix Editora, Sevilla (2005), 23–36.
2. A. Alhazov, R. Freund, Yu. Rogozhin, Computational Power of Symport / Antiport: History, Advances and Open Problems, R. Freund et al. (eds.), Membrane Computing, International Workshop, WMC 2005, Vienna (2005), revised papers, *Lecture Notes in Computer Science* 3850, Springer (2006), 1–30.
3. F. Bernardini, A. Păun, Universality of Minimal Symport/Antiport: Five Membranes Suffice, WMC03 revised papers in *Lecture Notes in Computer Science* 2933, Springer (2004), 43–54.
4. M. Cavaliere, R. Freund, Gh. Păun, Event–Related Outputs of Computations in P Systems, M.A. Gutiérrez-Naranjo et al. (eds.), Cellular Computing (Complexity Aspects), ESF PESC Exploratory Workshop, Fénix Editora, Sevilla (2005), 107–122.
5. R. Freund, A. Păun, Membrane Systems with Symport/Antiport: Universality Results, in *Membrane Computing. Intern. Workshop WMC-CdeA2002, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), *Lecture Notes in Computer Science*, 2597, Springer-Verlag, Berlin (2003), 270–287.
6. P. Frisco, J.H. Hogeboom, P systems with Symport/Antiport Simulating Counter Automata, *Acta Informatica*, 41 (2004), 145–170.
7. P. Frisco, S. Ji, Towards a hierarchy of conformon-P systems, *Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 2002, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), Springer, Berlin, 2003, 302–318,

8. O.H. Ibarra, A. Păun, Counting Time in Computing with Cells, Proceedings of DNA11 conference, June 6-9, 2005, London Ontario, Canada, (14 pages).
9. M. Ionescu, Gh. Păun, T. Yokomori, Spiking Neural P Systems, *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
10. M.L. Minsky, Recursive Unsolvability of Post's Problem of "Tag" and Other Topics in Theory of Turing Machines, *Annals of Mathematics*, 74 (1961), 437–455.
11. A. Păun, Gh. Păun, The Power of Communication: P Systems with Symport/Antiport, *New Generation Computing*, 20, 3 (2002) 295–306.
12. Gh. Păun, Further Twenty-six Open Problems in Membrane Computing, the Third Brainstorming Meeting on Membrane Computing, Sevilla, Spain, February 2005.
13. Gh. Păun, M.J. Pérez-Jiménez, F. Sancho-Caparrini, On the Reachability Problem for P Systems with Symport/Antiport, *Proc. Automata and Formal Languages Conf.*, Debrecen, Hungary, 2002.
14. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, Spike Trains in Spiking Neural P Systems, *International Journal of Foundations of Computer Science*, in press.
15. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.