Fundamenta Informaticae XX (2007) 1–14
IOS Press

Time and Synchronization in Membrane Systems

Matteo Cavaliere*
Department of Computer Science and Artificial Intelligence
University of Sevilla,
Av. Reina Mercedes, 41012, Sevilla, Spain

Dragoş Sburlan †
Department of Computer Science and Artificial Intelligence
University of Sevilla,
Av. Reina Mercedes, 41012, Sevilla, Spain,
Department of Informatics and Numerical Methods
Ovidius University of Constantza
124 Mamaia Bv., Constantza, Romania
dsburlan@univ-ovidius.ro

Abstract. Membrane systems are parallel computational devices inspired from the cell functioning. Since the original definition, a standard feature of membrane systems is the fact that each rule of the system is executed in exactly one time-unit. However, this hypothesis seems not to have a counterpart in real world. In fact, in cells, chemical reactions might take different times to be executed. Therefore, a natural step is to associate to each rule of a membrane system a certain time of execution. We are interested in membrane systems, called time-free, working independently from the assigned execution time of the rules. A basic and interesting problem in time-free membrane systems consists in the synchronization of different rules, running in parallel, and having unknown execution times. Here, we present different ways to approach this problem within the framework of membrane systems. Several research proposals are also suggested.

Address for correspondence:
*This author acknowledges the doctoral FPU-MEC fellowship.
†This author acknowledges the doctoral MAE-AECI fellowship.

Keywords: Membrane Systems, Synchronization, Time, Time-Free, Universality.

1. Introduction: The Time-Freeness Property

Standard features of membrane systems (known also as P systems) are the presence of a global clock, common to all the regions, and the hypothesis that each rule is executed in exactly one time-unit. These mathematical features, in general, do not have a corresponding counterpart in cell biology. Even more, in the real world, a common global clock cannot be found in living beings: we can discover many different systems that evolve independently, with different times, and that use particular methods to synchronize their evolutions. In particular this is true when we consider our society as a collection of independent and still interactive and "synchronized" human beings, living in a "parallel" way. Mathematical modeling of human time has been afforded in the comprehensive surveys that can be found in [7, 8]. On the other hand, this perfectly applies to systems where different communicating components, evolving in parallel, can be identified.

An example of this can be found even inside nano-systems, like living cells, where different chemical reactions are executed in parallel, with different execution times, and such that various processes are synchronized via biological signals that move across different areas of the cell ([12]).

Starting from all these considerations, a model of membrane systems based on signals has been introduced in [1]. Later, in [3], a model of membrane systems has been introduced where to each evolution rule r is associated a certain integer value $e(r)$ that indicates its execution time (in literature they are known as *timed P systems*).

On the other hand, the execution time of a certain chemical reaction might be difficult to know precisely and usually such parameter is very sensitive to environmental factors that might be hard to control. Therefore it is interesting to construct systems that work in the way we expect as much as possible independently from the values associated to the execution times of rules.

For this reason, in [3], time-free P systems have been introduced. Informally, a time-free P system is a P system that produces always the same result, independently from the execution times of its rules.

One possible mathematical formalization of this idea is to take into account any possible value associated to the parameter $e(r)$, for any rule r present in the system. A P system that generates (or accepts) the same family of vectors of natural numbers, independently of the values assigned to the execution time of its rules, is called *time-free*.

In this way, a time-free P system can be considered stable against "environmental" factors that might influence the execution times of the rules.

As in real world where synchronization of independent events is fundamental, also in the case of time-free P systems a similar problem must be considered: in order to get time-free systems that are computationally powerful enough it is important to synchronize rules with different execution times and that run in parallel. In this paper we present several approaches to this problem. Initially, we show how signals can be useful to tackle this problem (several results on time-free P systems with signal-promoters are recalled).

On other hand, synchronization can also be obtained by using a restricted kind of cooperative evolution rules like bi-stable catalytic rules. Finally, we show how universal computational devices can be obtained by making use of synchronization based only on communication (symport/antiport rules).

We conclude the paper with several open problems and research proposals.

2. Time-Free P Systems: Definition

We recall the definition of time-free P systems as it was originally introduced in [3]. The systems defined here make use of signal-promoters (introduced in [1]) and of (bi-stable) catalysts (recalled in [11]).

Definition 2.1. A P system Π of degree $m \geq 1$, with signal-promoters and bi-stable catalysts is a construct

$$\Pi = (V, C, D, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, R'_1, \ldots, R'_m, i_0),$$

where:

- V is the alphabet of Π; its elements are called *objects*;
- $C \subseteq V$ is the set of bi-stable catalysts; each bi-stable catalyst c is an object that can be in two states, c and $\overline{c}$ (if for a bi-stable catalyst c the two states coincides, then c is a catalyst);
- $D \subseteq V$ is the set of signal-promoters;
- μ is a membrane structure consisting of m membranes labeled $1, 2, \cdots, m$, where m is called the *degree* of the system Π;
- w_i, $1 \leq i \leq m$, specifies the multiset of objects present in the corresponding region i at the beginning of the computation;
- R_i, $1 \leq i \leq m$, are finite sets of evolution rules over V associated with regions $1, 2, \ldots, m$ of μ; there are rules of two types, non-cooperative, that are of the form $a \to v$ where a is an object from $V \setminus (C \cup D)$ and v is a string over $\{a_{here}, a_{out} \mid a \in V \setminus (C \cup D)\} \cup \{a_{in_j} \mid a \in V \setminus (C \cup D), 1 \leq j \leq m\}$, and catalytic rules (using bi-stable catalysts) of the forms $ca \to cv, ca \to \overline{c}v, \overline{c}a \to \overline{c}v$, and $\overline{c}a \to cv$, where a is an object from $V \setminus (C \cup D)$, v is a string over $\{a_{here}, a_{out} \mid a \in V \setminus (C \cup D)\} \cup \{a_{in_j} \mid a \in V \setminus (C \cup D), 1 \leq j \leq m\}$, and $c \in C$;
- R'_i, $1 \leq i \leq m$, are finite sets of signaling-rules over D associated with regions $1, 2, \ldots, m$ of μ; the signaling-rules are of the form $a \to v|_z$ or $ca \to cv|_z$, where a is an object from $V \setminus (C \cup D)$, v is a string over $V \setminus (C \cup D)$, z is a string representing a subset of $\{(p, here), (p, out) \mid p \in D\} \cup \{(p, in_j) \mid p \in D, 1 \leq j \leq m\}$, and $c \in C$; if there is no ambiguity on the target, then the target in_j is simply written as in;
- $i_0 \in \{0, 1, \cdots, m\}$ is the label of the *output region* (0 represents the environment).

Given a computable mapping

$$e : R_1 \cup \cdots \cup R_m \cup R'_1 \cup \cdots \cup R'_m \longrightarrow \mathbb{N}$$

and system Π as defined above, it is possible to construct a *timed P system* as

$$\Pi(e) = (V, C, D, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, R'_1, \ldots, R'_m, i_0, e)$$

working in the following way.

We suppose to have an external clock (that does not have any influence on the system) that marks time units of equal length, starting from a time 0.

To each region of the system is associated a finite number of objects (among them, signal-promoters and catalysts) and a finite number of evolution rules and of signaling-rules.

At each time, in the regions of the system we have together rules in execution and rules not in execution. At each time, all the rules that can be applied (started) in each region, must be applied.

If a rule $r \in R_i, R'_i, 1 \leq i \leq m$, is applied, then all objects that can be processed by the rule have to evolve by this rule.

To apply an evolution rule $u \rightarrow v$ or $u \rightarrow v|_z$ in a region i means to remove the multiset of objects identified by u from region i, and to add the objects specified by the multiset v, in the regions specified by the target indications associated to each object in v.

Signaling rules are evolution rules, promoted by the signal-promoters specified in the string z (signal-promoters work like standard promoters but they can only be moved and not created, see [1]).

In the case of a signaling-rule $u \rightarrow v|_z$, also the signal-promoters specified by z are moved to the regions according to their target indications. In every region the signal-promoters are present in the *set sense*, i.e., in each region cannot be more than one copy of the same signal-promoter.

When a rule r (either evolution or signaling) is started at time j, then its execution terminates at time $j + e(r)$ (the objects produced as well as the signal-promoters moved by the rule can be used starting from the time $j + e(r) + 1$).

If two rules start at the same time, then possible conflicts for using the occurrences of objects are solved assigning the occurrences in a non-deterministic way. The rules are applied in the maximally parallel way, as usually defined in the P systems framework.

Notice that, when the execution of a rule r is started, the occurrences of objects used by this rule are not available for other rules during the entire execution of r.

The computation stops when no rule can be applied in any region and there is no rule in execution.

The output of a halting computation is the vector of numbers representing the multiplicities of objects present in the output region in the halting configuration.

Collecting all the vectors obtained, for any possible halting computation, we get the set $Ps(\Pi(e))$ of vectors of natural numbers generated by the system $\Pi(e)$.

Definition 2.2. A P system $\Pi = (V, C, D, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, R'_1, \ldots, R'_m, i_0)$ is *time-free* if and only if every timed system in the set

$$\{\Pi(e) \mid e : R \longrightarrow \mathbb{N}, e \text{ computable}\},$$

where $R = R_1 \cup \cdots \cup R_m \cup R'_1 \cup \cdots \cup R'_m$, produces the same set of vectors of natural numbers.

We use the notation

$$PsP_m(\alpha, j, free), \alpha \in \{ncoo, coo\} \cup \{2cat_k, cat_k \mid k \geq 0\},$$

to denote the family of sets of vectors of natural numbers generated by *time-free* P systems with at most m membranes, at most j signal-promoters, evolution rules and signaling-rules that can be non-cooperative $(ncoo)$, or catalytic $(cat_k/2cat_k)$, using at most k catalysts/k bi-stable catalysts (as usual, $*$ is used if the corresponding number of membranes, signal-promoters or catalysts/bi-stable catalysts is not known).

3. Time-Free and Non Time-Free P Systems: Two Examples

The reader can easily see that every P systems using non-cooperative rules (so in the Definition 2.1 the sets C and D are empty) is time-free (a formal proof can be found in [2]).

This result does not hold for P systems using catalytic rules (even with only one catalyst) as it is shown by the following example.

Example 3.1. Consider the following P system with one catalyst:

$$\Pi = (V, C, D, \mu, w_1, R_1, R_1', i_0 = 0),$$

where:

$$\begin{aligned}
V &= \{B', B'', c, X, D, b, a\};\\
C &= \{c\};\\
D &= \emptyset;\\
\mu &= [_1\,]_1;\\
w_1 &= B'B''c;\\
R_1 &= \{r_1 : B' \to b_{out}X,\ r_2 : cB'' \to c,\ r_3 : X \to D,\ r_4 : cX \to c,\ r_5 : D \to a_{out}\};\\
R_1' &= \emptyset.
\end{aligned}$$

The system Π is not time-free. To prove this assertion, we show that it generates two different set of vectors in case two different time-mappings (e' and e'') are considered.

These mappings are defined as follows:

$$\begin{aligned}
e'(r_1) &= 1,\\
e'(r_2) &= 2,\\
e'(r_i) &= k', \quad \text{for some } k' \in \mathbb{N},\ \text{for } i = 3, 4, 5,
\end{aligned}$$

$$\begin{aligned}
e''(r_1) &= 2,\\
e''(r_2) &= 1,\\
e''(r_i) &= k'', \quad \text{for some } k'' \in \mathbb{N},\ \text{for } i = 3, 4, 5.
\end{aligned}$$

If the execution times of the rules in Π are given by the mapping e', then one can see that $Ps(\Pi(e')) = \{(1, 1)\}$. In fact, both rules r_1 and r_2 are started in parallel. When r_1 terminates, the objects b and X are produced. Then, at step 2, the rule r_3 is applied and the object D is produced. Notice that, in step 2, the rule r_3 is the only one that can be applied, and this is because rule r_2 is still in execution (therefore, the catalyst c is busy). In step 3, the object a is produced by $D \to a_{out}$ and sent to the environment. Therefore this is the only possible computation and the output of the system is the set $\{(1, 1)\}$.

If the execution times of the rules in Π are associated by using the mapping e'', then $Ps(\Pi(e'')) = \{(1,0),(1,1)\}$.

In fact, rules r_1 and r_2 are started in parallel as in the previous case but, because of the time-mapping e'', rule r_2 ends after rule r_1. Then, after two steps, object X is produced and b is sent to the environment; in the next step, X can be rewritten in two possible ways. In the first way the catalyst c can be used to apply rule r_4 and then the computation halts producing as output the vector $(1,0)$. In the second way rules r_3 and then r_5 are used. Rule r_5 produces and sends the object a to the environment. Therefore, in this case, we have as the output of the system the set $\{(1,1),(1,0)\}$. Notice that for any computable time-mapping e, $Ps(\Pi(e)) \neq \emptyset$.

On the other hand, it is easy to construct (computationally powerful) time-free P systems using signal-promoters, as shown in the following example.

Example 3.2. We consider the system

$$\Pi = (V, C, D, \mu, w_1, w_2, R_1, R_2, R_1', R_2', i_0),$$

where:

$$\begin{array}{ll} V = \{a,b,p\}; & R_1 = \emptyset; \\ C = \emptyset; & R_2 = \emptyset; \\ D = \{p\}; & R_1' = \{b \to b|_{(p,in)},\ b \to b|_{(p,out)}\}; \\ \mu = [_1\,[_2\,]_2\,]_1; & R_2' = \{a \to aa|_{(p,out)}\}; \\ w_1 = bp; & i_0 = 2. \\ w_2 = a; & \end{array}$$

The rule $a \to aa$ is activated by the signal-promoter p which is present at the beginning of the computation in region 1. The rule is applied an arbitrary number of times in the maximally parallel manner. Every time that the signal-promoter p is sent to region 1, one of the rules present in that region is applied. If rule $b \to b|_{(p,in)}$ is applied, then the process can be iterated. If rule $b \to b|_{(p,out)}$ is applied, then the signal-promoter is sent to the environment and the computation halts with a number of objects in region 2 that is a power of 2. It is trivial to see that the system generates the Parikh image of $\{a^{2^n} \mid n \geq 0\}$ independently from the execution times of the rules and, therefore, the system Π is time-free.

4. Synchronization by Signals: Computational Results

Using signals, as presented in Example 3.2, it is possible to synchronize in an easy way the rules of the system and to get time-free systems with enough computational power. In what follows we recall from [3] a series of results for time-free P systems using signal-promoters. Initially, we consider time-free P systems using only non-cooperative rules and signal-promoters. As soon as we increase the power of the rules to be bi-stable catalytic ones, then we obtain universality even without using signals (in this case the synchronization is obtained by making use of the short term memory of the bi-stable catalysts). On the other hand, the short memory of bi-stable catalysts can be simulated by signal-promoters and, in this way, we get an universality proof for time-free P systems using catalysts and signal-promoters.

4.1. Time-Free P Systems using Non-Cooperative Rules and Signals

Using only non-cooperative rules, signal-promoters, and two regions it is possible to generate at least the family of Parikh images of languages generated by Indian parallel grammars (denoted by $PsIPG$). Indian parallel grammars can generate non semilinear languages, then whose Parikh set is not in $PsCF$ (for details on these grammars we refer to [4]).

Theorem 4.1. $PsIPG \subseteq PsP_2(ncoo, *, free)$.

The proof of this result can be found in [3].

4.2. Time-Free P Systems Using Catalysts and Signal-Promoters: Universality

In time-free P systems synchronization can be obtained by using the short term memory of bi-stable catalysts. In [2] we can find the following universality result ($PsRE$ denotes the family of Turing computable sets of vectors of natural numbers).

Theorem 4.2. $PsRE = PsP_1(2cat_*, 0, free)$.

Concerning this last result, the following considerations need to be done. The class of time-free P systems seems to have similitudes with the class of *non-synchronized P systems* recalled in [11] (non-synchronized P systems are systems with symbol-objects where in each region a rule $A \to A$ is present, for each symbol A in the alphabet). In fact, in both models, there is a certain degree of "asynchronism" that must be tackled during the computation. But the results obtained are strongly different for the two models: while time-free P systems using bi-stable catalysts are universal (as recalled above), non-synchronized P systems can generate only the length sets of context-free languages when using bi-stable catalysts (the proof can be found in [11]). Such big difference of computational power is due to the fact that in time-free P systems is still possible to make use of the maximal parallelism as a tool for synchronizing different computational branches.

On the other hand, as we can see in the next theorem, a time-free P system without signals, with bi-stable catalysts and a membrane structure of the form $\mu = [_1\ [_2 \cdots [_m\ ]_m\ \cdots]_2\ ]_1$ can be simulated by a time-free P system using only catalysts, (at most) $m + 1$ membranes, and signal-promoters (the following result improves the result presented in [3] where also a priority relation on the set of rules is used). The idea of the proof follows the one given in [3].

Theorem 4.3. $PsRE = PsP_2(cat_*, *, free)$.

Proof:
To prove the theorem we show how a time-free P system

$$\begin{aligned}\Pi_1 \quad = \quad & (V_1, C_1, \emptyset, [_1\ [_2 \cdots\ [_m\ ]_m\ \cdots]_2\ ]_1, w_{1,1}, w_{1,2}, \cdots, w_{1,m}, \\ & R_{1,1}, R_{1,2}, \cdots, R_{1,m}, \emptyset, \emptyset, \cdots, \emptyset, i_0),\end{aligned}$$

using bi-stable catalysts, m membranes can be simulated by a time-free P system

$$\Pi_2 = (V_2, C_2, D_2, [_0\ [_1\ [_2 \cdots\ [_m\ ]_m\ \cdots]_2\ ]_1\ ]_0, w_{2,0}, w_{2,1}, w_{2,2}, \cdots, w_{2,m}, R_{2,0}, R_{2,1}, R_{2,2}, \cdots, R_{2,m}, R'_{2,0}, R'_{2,1}, R'_{2,2}, \cdots, R'_{2,m}, i_0),$$

using catalysts, $m+1$ membranes and signal-promoters (both systems Π_1, Π_2 are written according to the notation given in Definition 2.1). Without loss of generality, we suppose that each occurrence of the bi-stable catalysts present in the regions of Π_1 is named differently. We suppose that the set of bi-stable catalysts used in Π_1 is $C_1 = \{c_1, c_2, \cdots, c_h\}$.

We construct the system Π_2 in the following way.

Suppose $j \in \{1, 2, \cdots, h\}$. For each rule $r^{1,1}_{i\langle a\to w\rangle} : \overline{c_j}a \to c_j w$ that is present in $R_{1,i}$ we add to $R'_{2,i}$ the signaling-rule $r^{2,1}_{i\langle a\to w\rangle} : c_j a \to c_j w|_{(\overline{p_j},out)}$, and to $R'_{2,i-1}$ the signaling-rules $X_j \to X''_j|_{(\overline{p_j},here)}$ and $X''_j \to X'''_j|_{(p_j,in)}$.

In a similar way, for each rule $r^{1,2}_{i\langle a\to w\rangle} : c_j a \to \overline{c_j}w$ that is present in $R_{1,i}$ we add the signaling-rule $r^{2,2}_{i\langle a\to w\rangle} : c_j a \to c_j w|_{(p_j,out)}$ to $R'_{2,i}$, and to $R'_{2,i-1}$ the signaling-rules $X'''_j \to X'_j|_{(p_j,here)}$ and $X'_j \to X_j|_{(\overline{p_j},in)}$.

For each rule $r^{1,3}_{i\langle a\to w\rangle} : \overline{c_j}a \to \overline{c_j}w$ that is present in $R_{1,i}$ we add the signaling-rule $r^{2,3}_{i\langle a\to w\rangle} : c_j a \to c_j w|_{(\overline{p_j},here)}$ to $R'_{2,i}$; for each rule $r^{1,4}_{i\langle a\to w\rangle} : c_j a \to c_j w$ presents in $R_{1,i}$ we add the rule $r^{2,4}_{i\langle a\to w\rangle} : c_j a \to c_j w|_{(p_j,here)}$ to $R'_{2,i}$.

For each non-cooperative rule $r^{1,0}_{i\langle X\to w\rangle} : X \to w$ that is present in $R_{1,i}$ we add the non-cooperative rule $r^{2,0}_{i\langle X\to w\rangle} : X \to w$ to the set $R_{2,i}$.

We take the alphabet $V_2 = V_1 \cup \{X_i, X''_i, X'_i, X'''_i \mid 1 \le i \le h\}$ and the set of signal-promoters as $D_2 = \{\overline{p_i}, p_i \mid 1 \le i \le h\}$. The set of catalysts C_2 is exactly the set C_1 except the fact that objects in C_2 are used as catalysts and not as bi-stable catalysts.

Finally we construct $w_{2,l}$, for $0 \le l \le m$, in the following way. Add to $w_{2,l}$ all the objects $x \in V_1 \backslash C_1$ present in $w_{1,l}$ (by definition, $w_{2,0} = \lambda$). If $w_{1,l}$ contains $c_{j'}$, for some $j' \in \{1, 2, \cdots, h\}$, then add $p_{j'}$ and $c_{j'}$ to $w_{2,l}$; if $w_{1,l}$ contains $\overline{c_{j'}}$ for some $j' \in \{1, 2, \cdots, h\}$, then add $\overline{p_{j'}}$ and $c_{j'}$ to $w_{2,l}$; if $w_{1,l+1}$ contains $\overline{c_{j'}}$, for some $j' \in \{1, 2, \cdots, h\}$, then add $X_{j'}$ to $w_{2,l}$; if $w_{1,l+1}$ contains $c_{j'}$, for some $j' \in \{1, 2, \cdots, h\}$, then add $X'''_{j'}$ to $w_{2,l}$.

The main idea of the proof is that the "change of state" of a bi-stable catalyst present in region i of Π_1 is simulated by an exchange of signal-promoters between region i and the surrounding region $i-1$ in Π_2.

For instance (in all other cases the situation is similar), the execution of the rule $r^{1,1}_{i\langle a\to w\rangle} : \overline{c_j}a \to c_j w$ present in region i of Π_1, for some $j \in \{1, 2, \cdots, h\}$, is simulated in Π_2 in the following way. First, rule $c_j a \to c_j w|_{(\overline{p_j},out)}$ is executed in region i; at the end of its execution the signal-promoter $\overline{p_j}$ is sent out to the surrounding region $i-1$. There, the two rules $X_j \to X''_j|_{(\overline{p_j},here)}$ and $X''_j \to X'''_j|_{(p_j,in)}$ are executed sequentially and they send inside region i the signal-promoter p_j. In region i of Π_2 the presence of signal-promoter p_j activates now all (and only) the rules catalyzed by c_j; in this way the simulation of the execution of rule $r^{1,1}_{i\langle a\to w\rangle} : \overline{c_j}a \to c_j w$ in Π_1 has been completely simulated (the obtained object X'''_j in region $i-1$ stores the information that c_j is in state c_j).

The execution of rule $r^{1,2}_{i\langle a\to w\rangle} : c_j a \to \overline{c_j}w$ present in region i of Π_1 is simulated in Π_2 in the

following way. First the rule $r^{2,2}_{i\langle a\to w\rangle} : c_j a \to c_j w|_{(p_j,out)}$ in $R'_{2,i}$ of Π_2 is executed; at the end of its execution the signal-promoter p_j is sent out to the surrounding region $i-1$. There, both rules $X'''_j \to X'_j|_{(p_j,here)}$ and $X'_j \to X_j|_{(\overline{p_j},in)}$ are executed and the signal-promoter $\overline{p_j}$ is sent to region i. In region i of Π_2 the presence of signal-promoter $\overline{p_j}$ activates now all (and only) the rules catalyzed by $\overline{c_j}$; in this way the simulation of the execution of rule $r^{1,2}_{i\langle a\to w\rangle} : c_j a \to \overline{c_j} w$ has been completely simulated (the object X_j obtained in region $i-1$ stores the information that c_j is in state $\overline{c_j}$).

From the way we construct Π_2 and because Π_1 is time-free, then also Π_2 is time-free; moreover, they generate the same set of vectors of natural numbers. Therefore, because of Theorem 4.2, the statement is true. □

As a consequence of Theorem 4.3, a natural question concerning time-free P systems using only catalysts arises: are they universal? In standard P systems (where the time of each rule is fixed to be one clock step) the answer is known to be positive ([5]). In case of time-free P systems we conjecture that the answer is negative, because catalysts do not have memory and, therofore, it seems that they cannot be used to synchronize rules with different (and unknown) times of execution.

5. Synchronization by Transport Mechanisms: the Case of Symport/Antiport Rules

As mentioned in the Introduction, synchronization in time-free P systems can be also obtained by using transport mechanisms. We consider interesting to investigate systems where the synchronization during the computation is obtained by only moving objects across the regions of the system.

In particular, in this section, we start the investigation of time-free P systems with symport/antiport rules. These kinds of rules have been originally introduced in [10] and have been motivated by the biological mechanisms of moving chemicals through the cell regions.

We recall the definition of P sytems with symport/antiport rules in the accepting variant (see [11] for further details). A P system with symport/antiport is a construct

$$\Pi = (V, \mu, w_1, \cdots, w_m, E_{inf}, E_{fin}, R_1, \cdots, R_m, i_{inp}),$$

where:

- V is the alphabet of Π; its elements are called *objects*;
- μ is a membrane structure consisting of m membranes injectively labelled with $1, 2, \cdots, m$, where m is called the *degree* of the system Π;
- w_i, $1 \leq i \leq m$, specifies the multisets of objects present in the corresponding region i at the beginning of the computation;
- $E_{inf} \subseteq V$ is the set of objects which are supposed to appear in arbitrary many copies in the environment;
- E_{fin} specifies the multiset of objects present, at the beginning of the computation, in a finite number of copies, in the environment;

- $R_1, \cdots, R_m$ are finite sets of *symport/antiport rules* over V associated with the membranes $1, 2 \cdots, m$ of μ; a *symport rule* is of the kind (x, in) or (x, out), while an *antiport rule* is of the kind $(x, out; y, in)$, where x, y are strings representing multisets of objects of V; for a symport rule (x, in) or (x, out) we say that $|x|$ is the *weight* of the rule; in case of an antiport rule $(x, out; y, in)$, the *weight* is defined as *max*$\{|x|, |y|\}$;

- $i_{inp} \in \{1, 2, \cdots, m\}$ is the label of the *input region*.

In a P system with symport/antiport, a configuration is described by the m-tuples of multisets of objects present in the m regions of the system and by the multiset describing the objects present in the environment in a finite number of copies.

The objects of E_{inf} are supposed to be present in the environment in an arbitrarily number of copies, their numbers remains arbitrarily irrespective of how many copies have been introduced in the system during the computation.

The application of the *symport rule* (x, in) to membrane i means to bring in membrane i the objects represented by x, from the surrounding region.

If the symport (x, out) is applied to membrane i, then the objects represented by the multiset x move out from this membrane to the surrounding region.

Finally, if the *antiport rule* $(x, out; y, in)$ is applied to membrane i, then the objects represented by multiset x are sent out of membrane i to the surrounding region while, at the same time, the objects represented by the multiset y move in the opposite direction.

In the accepting mode, a system Π as above starts computing in the configuration obtained by adding a multiset x to w_{inp}. The vector of multiplicities of objects in x is accepted if and only if the computation halts.

The definition of time-free P systems with symport/antiport rules follows the one given previously in Definition 2.2.

Given a computable mapping

$$e : R_1 \cup \cdots \cup R_m \longrightarrow \mathbb{N}$$

and system Π as defined above, it is possible to construct a *timed P system with symport/antiport rules* $\Pi(e)$ as $(V, \mu, w_1, \cdots, w_m, E_{inf}, E_{fin} R_1, \cdots, R_m, i_{inp}, e)$ working in the following way.

As earlier described, we suppose to have an external clock (that does not have any influence on the system) that starts at time 0 and ticks from each time j to the next one $j + 1$.

The computation (and the accepting of a vector of natural numbers) in a timed P system with symport/antiport rules is defined like for P systems with symport/antiport except that *when a rule r (either symport or antiport) is started at time j, then its execution terminates at time $j + e(r)$ (the objects moved can be used starting from the time $j + e(r) + 1$).* If two rules start at the same time, then possible conflicts for using the occurrences of objects are solved assigning the occurrences in a non-deterministic way.

The computation stops when no rule can be applied in any region and no rule is in execution.

Definition 5.1. A P system with symport/antiport

$$\Pi = (V, \mu, w_1, \cdots, w_m, E_{inf}, E_{fin}, R_1, \cdots, R_m, i_{inp}),$$

is *time-free* if and only if every timed system in the set

$$\{\Pi(e) \mid e : R \longrightarrow \mathbb{N}, e \text{ computable}\},$$

for $R = R_1 \cup \cdots \cup R_m$, accepts the same set of vectors of natural numbers.

We use the notation

$$PsPP_m(i, k, free)$$

to denote the family of sets of vectors of natural numbers accepted by *time-free* P systems with at most m membranes, symport rules of weight at most i, and antiport rules of weight at most k.

In what follows we present a brief description of register machines and of their computational power. More details can be found in [9].

An n-register machine is a construct $M = (n, \mathcal{P}, l_0, l_h)$ where n is the number of registers (each register stores an arbitrary natural number); $\mathcal{P}$ (the *program* of the machine) is a finite set of labeled instructions of the form $(l_1 : op(r), l_2, l_3)$ where $op(r)$ is an operation on register r of M, $l_1, l_2, l_3 \in Lab(\mathcal{P})$ (where $Lab(\mathcal{P})$ denotes the set of labels of the instructions from $\mathcal{P}$); l_0 is the initial label; l_h is the final label.

The operations allowed by an n-register machine are:
• $(l_1 : \text{ADD}(r), l_2, l_3)$ – increment the value stored into register r and proceed, in a non-deterministic way, to the instruction labeled l_2 or to the instruction labeled l_3 ($l_2 = l_3$ for the deterministic variant and then the instruction is written in the form $(l_1 : \text{ADD}(r), l_2)$);
• $(l_1 : \text{SUB}(r), l_2, l_3)$ – jump to instruction labeled l_3 if the register r is empty; otherwise subtract one from the value stored into register r and jump to instruction labeled l_2;
• $(l_h : \text{HALT})$ – halts the computation (there is an unique halting instruction).

A register machine $M = (n, \mathcal{P}, l_0, l_h)$ accepts a vector $(r_1, \cdots, r_{n-2}) \in \mathbb{N}^{n-2}$ iff, starting from the instruction labeled l_0, with register j having value r_j for $1 \leq j \leq n-2$, and the contents of registers $n-1$ and n being empty, the machine halts.

It is known (see [9]) that deterministic register machines accept exactly the family of Turing computable sets of vectors of natural numbers.

Theorem 5.1. $PsRE = PsPP_1(1, 2, free)$.

Proof:
Consider a register machine $M = (n, \mathcal{P}, l_0, l_h)$. We will simulate its accepting computation with a P system constructed in the following way. The value stored in register j of M will be represented by the multiplicity of an associated object a_j. Let $Lab(\mathcal{P}) = \{l_1, l_2, \cdots, l_q\}$.

Formally, we construct the P system $\Pi = (V, \mu, w_1, E_{inf}, E_{fin}, R_1, 1)$ where:

$V = Lab(\mathcal{P}) \cup \{S_l, T_l \mid l \in Lab(\mathcal{P})\} \cup E_{inf} \cup \{k, k_1\}$;

$E_{inf} = \{a_r \mid 1 \leq r \leq n\}$;

$E_{fin} = l_1 l_2 \cdots l_q S_1 S_2 \cdots S_q T_1 T_2 \cdots T_q k k_1$;

$\mu = [_1\ ]_1$;

$w_1 = l_0$; in addition, for a vector $(r_1, \cdots, r_{n-2})$ to be accepted by M, we introduce in the input region 1 the multiset $a_1^{r_1} \cdots a_{n-2}^{r_{n-2}}$;

R_1 is defined as follows:

• for each addition instruction $(l_1 : \text{ADD}(r), l_2) \in \mathcal{P}$, $l_1, l_2 \in Lab(\mathcal{P})$, $1 \leq r \leq n$, we add to R_1 the rule $(l_1, out; a_r l_2, in)$;

• for each subtraction instruction $(l_1 : \text{SUB}(r), l_2, l_3) \in \mathcal{P}$, $l_1, l_2, l_3 \in Lab(\mathcal{P})$, $1 \leq r \leq n$, we add to R_1 the rules:

$(l_1, out; S_{l_1} k, in)$,

$(S_{l_1} a_r, out; T_{l_1}, in)$,

$(k, out; k_1, in)$,

$(S_{l_1} k_1, out; l_3, in)$,

$(T_{l_1} k_1, out; l_2, in)$;

• for instruction $(l_h : \text{HALT}) \in \mathcal{P}$, $l_h \in Lab(\mathcal{P})$, we add to R_1 the rule (l_h, out).

The system Π simulates the work of the register machine program as follows. Suppose that the content of region 1 is described by the multiset $a_1^{i_1} \cdots a_n^{i_n} l_1$.

The simulation of an instruction $(l_1 : \text{ADD}(r), l_2) \in P$ is straightforward: the object l_1 (which corresponds to register machine instruction label l_1) is sent out of the membrane, while, in the same time, objects a_r and l_2 are brought inside. In this way, the new instruction label is generated and the number of objects a_r is incremented.

If an instruction $(l_1 : \text{SUB}(r), l_2, l_3)$ has to be executed by M, then object l_1 is sent out, as in the previous case, and at the same time, objects S_{l_1} and k are brought inside the region. Object S_{l_1} represents the "command" to remove one object a_r from the current multiset in case this is possible; object k represents a "checker" – it will check whether or not the subtraction command was successfully accomplished.

If in the region there exists at least one copy of object a_r, then rule $(S_{l_1} a_r, out; T_{l_1}, in)$ starts in the same time with rule $(k, out; k_1, in)$. Next, since object S_{l_1} is not anymore in region 1, only the rule $(T_{l_1} k_1, out; l_2, in)$ can start. Therefore, the label l_2 of the next register machine instruction is generated and the simulation can continue.

In case in region 1 there is no object a_r, only rule $(k, out; k_1, in)$ can be started (in fact, rule $(S_{l_1} a_r, out; T_{l_1}, in)$ cannot be started). Next, since in region 1 there is no object T_{l_1}, only the rule $(S_{l_1} k_1, out; l_3, in)$ can be started. Therefore, the label l_3 of the next register machine instruction is generated and simulation can continue.

When object l_h appears in region 1, indicating that the register machine program has to terminate, then the rule (l_h, out) is executed and the simulation terminates as well.

One can notice that the simulation of register machine instructions is deterministic. In addition, the result of the simulations does not depend on the execution times of rules. Therefore, the constructed system is time-free. Consequently, the system Π accepts the same set of vectors as accepted by M, irrespective of the duration of rules. □

6. Open Problems and Concluding Remarks

Several open problems and research proposals can be easily distinguished for time-free P systems. As a general suggestion, it would be interesting to see which are the results known for standard P systems that remain true also for time-free P systems. For instance, as already mentioned, it would be interesting to know if time-free catalytic P systems are universal. In this respect we have conjectured that the answer to this question is negative. Such an answer would imply that for the class of catalytic P systems *time is important* to get universality.

Another interesting way to investigate time-free P systems is to impose certain conditions on the time of execution of the rules (this class of systems has been refereed as *partially time-free P systems* in [3]). How to formalize and to use such conditions? Which are the conditions that should be used (that means, "realistic")? These are only two of the many questions concerning this class.

As we have seen in Example 3.1, even using only one catalyst is possible to construct non time-free P systems. A rather natural question is the following one: given an arbitrary system is it possible to decide if it is time-free? This problem has been addressed in [2] where it has been shown that it is undecidable if a P system using bi-stable catalysts is time-free. It would be useful to find a class C of P systems with enough computational power, with two more features: given any system in C there exists an equivalent time-free system in C, producing the same output or accepting the same input; given an arbitrary system Π in C it is possible to decide (in an easy way) if Π is time-free. The class of catalytic P systems seems to be a good candidate.

Moreover, something that can be further investigated concerns new methods to synchronize the computation in time-free P systems. We have mainly used signal-promoters; are there other synchronizing devices (maybe inspired from cell functioning)? This work might also have impact on the more general field of asynchronous parallel computing.

It is also possible to add other parameters to construct a "more realistic" P system; for instance, it would be very interesting to associate to each rule a time of delay that indicates the time to wait before a rule is started and then to study a class of delay-free systems. This might model the fact that, sometimes, chemical rules are not started immediately, even in the presence of the necessary chemicals.

We conclude with the belief that these lines of research deserve further investigations.

References

[1] Ardelean, I.I., Cavaliere, M., Sburlan, D.: Computing Using Signals: From Cells to P Systems, *Technical Report 01/2004 of RGNC, Brainstorming Week on Membrane Computing*, University of Sevilla, Sevilla, 2004, and *Soft Computing*, to appear.

[2] Cavaliere, M., Deufemia, V.: On Time-Free P systems, manuscript, 2004.

[3] Cavaliere, M., Sburlan, D.: Time-Independent P Systems, *Proceedings of WMC5, Fifth Workshop on Membrane Computing*, Milano, 2004, *Lecture Notes in Computer Science*, to appear.

[4] Dassow, J., Păun, Gh.: *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[5] Freund, R., Kari, L., Oswald, M., Sosik, P.: Computationally Universal P systems Without Priorities: Two Catalysts are Sufficient, submitted, 2004.

[6] Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[7] Marcus, S.: The Mathematical Modeling of Human Time, *Noesis*, 10, 1984, 129–135.

[8] Marcus, S.: *The Time* (in Romanian), Ed. Albatros, Bucuresti, 1985.

[9] Minsky, M.L.: *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, 1967.

[10] Păun, A., Păun, Gh.: The Power of Communication: P Systems with Symport/Antiport, *New Generation Computing*, 20, 2002, 295–305.

[11] Păun, Gh.: *Membrane Computing – An Introduction*, Springer-Verlag, Berlin, 2002.

[12] Ray, L.B., Adler, E.M., Gough, N.G., Chong, L.D.: Focus Issue: Computational Approaches in Signal Transduction, *Science's STKE*, eg3 2004, *http://www.stke.org*.