

# Towards Asynchronous P Systems

Matteo CAVALIERE

Department of Computer Science and Artificial Intelligence  
University of Sevilla  
Sevilla, Spain  
E-mail: martew@inwind.it

## Abstract

We introduce a class of P systems where with each rule is associated an integer that represents the time needed by the rule (reaction) to be entirely executed. This new parameter brings a certain degree of asynchronism with respect to standard P systems where the time of execution of each rule is fixed in one step. In this work we are interested in P systems, called time-free, working always the same way (i.e., producing always the same result) independently from the values associated to the time of execution of their rules; in biological terms, this means that we are interested in systems that are stable with respect to (possible and, maybe, unpredictable) changes of the time that the reactions need to be executed.

Several results are presented together with some open problems and research proposals.

## 1 Introduction: Motivations

A standard feature of membrane computing is the fact that each rule is executed in exactly one step: this mathematical feature, in general, does not have a corresponding counterpart in cell biology. In general, different chemical reactions take different times to be executed. In many cases, different reactions with different times of reaction are synchronized via biological signals that move across different areas present in the cell; some considerations on this topic can be found in [1] where a class of P systems that uses signals to compute has been introduced. On the other hand, it is true that, generally, in cell, when a reaction is applied, then all the chemicals that can be subject of the reaction are transformed by the reaction.

Starting from all these considerations we introduce and investigate here a model of P systems where to each rule  $r$  is associated a certain value  $e(r)$  that indicates its time of execution.

In cell biology the time of execution of a 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, we believe that it is extremely interesting to construct systems that work in the way we expect as much as possible independently from the values associated to the time of execution.

For this reason we introduce here the concept of time-free P system. In informal words, a time-free P system is a P system that produces always the same result independently from the time of executions of its rules.

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

In this way a *time – free* P system can be considered stable against “environmental” factors that might influence the time of execution of the rules. In case the time of execution associated to some of the rules have to satisfied certain conditions then the system is called *partially time-free* (the mathematical formalization of this is not treated in this paper).

In this paper we start only a preliminary investigation of this model of computation. Several results are presented considering the use of cooperative/non-cooperative rules, the absence of signal-promoters (then considering a basic model), the use of priority. Several open problems are also presented.

## 2 Asynchronous P(e) Systems: Definition

**Definition 2.1** *An asynchronous P(e) system (in short, an  $P^a(e)$  system), of degree  $m \geq 1$ , with symbol-objects and with time-mapping  $e$  is a construct*

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

where:

- $V$  is the alphabet of  $\Pi$ ; its elements are called objects;
- $C \subseteq V$  is the set of catalysts;
- $P \subseteq V$  is the set of signal-promoters;
- $\mu$  is a membrane structure consisting of  $m$  membranes labeled  $1, 2, \dots, m$ ;
- $w_i, 1 \leq i \leq m$ , specify the multisets of objects present in the corresponding region  $i$  at the beginning of a computation;
- $R_i, 1 \leq i \leq m$ , are finite sets of evolution rules over  $V$  associated with regions  $1, 2, \dots, m$  of  $\mu$ ; these evolution rules are of the form  $a \rightarrow v$  or  $ca \rightarrow cv$ , where  $a$  is an object from  $V \setminus \{C \cup P\}$  and  $v$  is a string over  $\{a_{here}, a_{out}, a_{in_j} \mid a \in V \setminus \{C \cup P\}, 1 \leq j \leq m\}, c \in C$ ;
- $R'_i, 1 \leq i \leq m$ , are finite sets of signaling-rules over  $P$  associated with regions  $1, 2, \dots, m$  of  $\mu$ ; the signaling-rules are of the form  $a \rightarrow v|_p$  or  $ca \rightarrow cv|_p$ , where  $a$  is an object from  $V \setminus \{C \cup P\}$ ,  $v$  is a string over  $V \setminus \{C \cup P\}$ , and  $p$  is a string representing a set  $S \subseteq \{p'_{here}, p'_{out}, p'_{in_j} \mid p' \in P, j \in \{1, \dots, m\}\}$ ;
- $e$  is a mapping that associates to each rule  $r \in R_i \cup R'_i, 1 \leq i \leq m$ , an integer greater than 0 that indicates the time of execution of the rule;

- $i_0$  is a number between 0 and  $m$  and specifies the output region of  $\Pi$  (0 represents the environment).

To describe the working of the system we suppose to have an external clock (that does not have any influence on the system) that passes from a certain time-unit  $j$  to the next one  $j + 1$ . We suppose that the external clock starts at 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 unit in the regions of the system we have together rules in execution and rules not in execution. At each time unit all the rules that can be applied (started) in each region are applied in the maximally parallel way. 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 (a rule is applied in a maximally parallel manner).

The application of an evolution rule  $u \rightarrow v$  or  $u \rightarrow v|_p$  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  $p$  (see more details in [1]). In case of signaling-rule  $u \rightarrow v|_p$  also the signal-promoters specified by  $p$  are moved to the regions according to their target indications. In every region the signal-promoters are present in the *set sense*, i.e., we cannot have more than one copy of the same signal-promoter in one region.

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

If two rules are started in the same time unit, then possible conflicts for using the occurrences of objects are solved assigning the objects in a non-deterministic way.

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

When using signals, there are two situations that can cause conflicts.

If (at least one of) the signal-promoters that activates  $r$  is moved out from the region where  $r$  is present before rule  $r$  is terminated, then the execution of  $r$  cannot continue and then the entire computation is trashed.

On the other hand, if two or more signaling-rules, promoted by the same signal-promoters, but with different targets terminate their execution in the same time unit, then, also in this case, the computation is trashed (a conflict over the destination of promoters appears).

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

If the system halts (the computation is *successful*), we consider the number of objects contained in the output region  $i_0$  as the result of the computation of  $\Pi$ .

The set of vectors of natural numbers computed in this way by the system  $\Pi$  is denoted by  $N(\Pi)$ .

We also investigate  $P^a(e)$  systems using a priority relation in the *strong* sense (as described in [5], Section 3.4.2).

In a  $P^a(e)$  system, a rule  $r_1$  in a region  $R_i$ ,  $1 \leq i \leq m$ , is started if there is no rule  $r_2$  which can also be started in the same time unit or that is already in execution and  $r_2 > r_1$ . If a rule  $r_2$  with a higher priority with respect to  $r_1$  is applied or is already in execution and it terminates at time-unit  $j$ , then  $r_1$  can be started only at time-unit  $j + 1$ .

Notice that when we say that a rule can be applied this means that there are the necessary occurrences of symbol-objects (without considering the presence/absence of signal-promoters to activate the rule).

If a  $P^a(e)$  system does not use signal-promoters (i.e., the set  $P$  is empty), then the system is called *basic*.

A  $P^a(e)$  system  $\Pi$  is called *time-free* when, for any possible mapping  $e$ , it generates always the same set of vectors of natural numbers. In case of time-free  $P^a(e)$  systems the function  $e$  is left unspecified and the system is simply indicated as time-free  $P^a$  system.

We use the following notation

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

to denote the family of sets of vectors of natural numbers generated by *free*  $P^a$  systems with at most  $m$  membranes,  $j$  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 is not known) and priority (the parameter  $j$  is 0 if the system is basic).

### 3 Time-Free $P^a$ Systems: An Example

We present a simple example of a time-free  $P^a$  system using two membranes, one signal-promoter and non cooperative rules that generates the Parikh image of the language  $\{a^{2^n} \mid n \geq 0\}$ . From this example it is clear how signals can be very useful to synchronize rules with different times of execution.

We take the system

$$\Pi = (V, C, P, \mu, w_1, w_2, R_1, R'_1, R_2, R'_2, i_o, e),$$

where:

- $V = \{a, z, p\}$ ;
- $C = \emptyset$ ;
- $P = \{p\}$ ;
- $\mu = [1[2]2]_1$ ;
- $w_1 = ap$ ;

- $w_2 = z$ ;
- $R_1 = \emptyset$ ;
- $R'_1 = \{a \rightarrow aa|_{p,out}\}$ ;
- $R_2 = \emptyset$ ;
- $R'_2 = \{z \rightarrow z|_{p,in_1}, z \rightarrow z|_{p,out}\}$ ;
- $i_o = 1$ .

The rule  $a \rightarrow aa$  is activated by the signal-promoter  $p$  which is presents at the beginning in region 1. The rule is applied an arbitrary number of times in the maximally parallel way. Every time the signal-promoter  $p$  is sent in region 2, one of the rules present in that region is applied. If the rule  $z \rightarrow z|_{p,in_1}$  is applied, then the process can be iterated. If the rule  $z \rightarrow z|_{p,out}$  is applied, then the signal-promoter is lost 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 always the Parikh image of  $\{a^{2^n} \mid n \geq 0\}$  independently from the mapping  $e$  used and therefore, the system  $\Pi$  is time-free.

## 4 Time-Free $P^a$ Systems Using Non Cooperative Rules

Before presenting the result of this section, we recall the definition of an *Indian parallel grammar* (for more details we refer the reader to [2] and [4]).

An Indian parallel grammar is a construct  $G = (N, T, S, P)$ , where at each step of the derivation every occurrence of one letter is rewritten using the same production.

This means that the derivations are defined in the following way: for every  $x \in (N \cup T)^+$ , for every  $y \in (N \cup T)^*$  :  $x \Rightarrow y$  if and only if:  $x = x_1 A x_2 A \cdots x_n A x_{n+1}$ ,  $A \in N$ ,  $x_i \in ((N \cup T) - A^*)$ ,  $1 \leq i \leq n + 1$ ,  $y = x_1 w x_2 w \cdots x_n w x_{n+1}$ ,  $A \rightarrow w \in P$ .

The language generated by  $G$  is  $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$ , where  $\Rightarrow^*$  denotes the reflexive and transitive closure of  $\Rightarrow$ .

The family of languages generated by Indian parallel grammars is denoted by  $IPG$ .

Using non cooperative rules, signal-promoters and two regions is possible to generate at least the Parikh set of languages in the family of  $IPG$ , as shown in the following theorem.

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

*Proof.* Given an Indian parallel grammar  $G = (N, T, S, P)$  we suppose that to each rule  $r \in P$  an unique label  $l(r)$  has been associated. We denote  $L(P) = \{l(r) \mid r \in P\}$ .

We construct the following  $P^a$  system simulating  $G$ :

$$\Pi = (V, C, P, \mu, w_1, w_2, R_1, R_2, R'_1, R'_2, i_o, e),$$

where:

- $V = N \cup T \cup P \cup \{Q', Q'', T, T', Z, \#\}$ ;
- $C = \emptyset$ ;
- $P = L(P) \cup \{s', s\}$ ;
- $\mu = [1[2]2]_1$ ;
- $w_1 = ZTr_1r_2 \cdots r_k, r_i \in L(P)$ ;
- $w_2 = Ss'Q$ ;
- $R_1 = \{r_4 : Z \rightarrow Z\}$ ;
- $R_2 = \{\# \rightarrow \#\}$ ;
- $R'_1 = \{r_3 : Z \rightarrow \lambda|_{s',here}, r_1 : T \rightarrow T'|_{s,in_2}\} \cup \{r_2 : T \rightarrow T'|_{r,in_2}, T' \rightarrow T|_{r,here} \mid r \in L(P)\}$ ;
- $R'_2 = \{X \rightarrow w|_{r,out} \mid X \rightarrow w \in P \text{ and } l(X \rightarrow w) = r\} \cup \{X \rightarrow \#|_{s,here} \mid X \in N\} \cup \{Q' \rightarrow \lambda|_{s',out}, Q \rightarrow Q'|_{s,here}\}$ ;
- $i_o = 1$ .

The system works in the following way. In region 2 (the output region) the rules of the grammar  $G$  are simulated. In region 1 the application of one of the rules in the set  $\{T \rightarrow T'|_{r,in_2} \mid r \in L(P)\}$  selects the rule of the grammar  $G$  to be activated (and, if possible, applied) in region 2, by sending the associate signal-promoter  $r \in L(P)$ . When an activated rule  $X \rightarrow w|_{r,out}$  is applied in region 2, the signal-promoter  $r$  is sent back to region 1. Therefore the simulation of another rule can be made.

To stop the computation it is necessary to halt the rule  $Z \rightarrow Z$  presents in region 1. To do this, the rule  $T \rightarrow T'|_{s,in_2}$  must be used in region 1. When  $T \rightarrow T'|_{s,in_2}$  is executed, the signal-promoter  $s$  will arrive in region 2 checking that all symbol-objects present are terminals (by activating the rules  $X \rightarrow \#|_{s,here} \mid X \in N$ ). Moreover, the rules  $Q' \rightarrow \lambda|_{s',out}, Q \rightarrow Q'|_{s,here}$  will send to region 1 the signal-promoter  $s'$  necessary to delete  $Z$  and then stop the computation. Therefore the computation halts when  $r_3$  is applied if and only if all terminals have been obtained in the output region and this means that the system  $\Pi$  generates exactly the Parikh set of the grammar  $G$ . It is clear that the system is time-free because it generates always the Parikh set of the grammar  $G$  for any possible time-mapping  $e$ .  $\square$

## 5 Basic $P^a(e)$ Systems (Using Priority)

If we do not use signal-promoters (i.e., in Definition 2.1 the set of signal-promoters  $P$  is empty), then it seems harder to synchronize the application of the rules in the regions of the system.

In this section we (shortly) investigate basic time-free  $P^a$  and “partially” time-free  $P^a(e)$  systems using a priority among evolution rules.

It is easy to see that for basic time-free  $P^a$  system the following result is true:

$$PsCF \subseteq PsP_2^a(ncoo, 0, free).$$

Given a context-free grammar  $G = (N, T, S, P)$  we construct the following  $P^a$  system simulating  $G$ :

$$\Pi = (V, C, P, \mu, w_1, R_1, R'_1, i_o, e),$$

where:

- $V = N \cup T$ ;
- $C = \emptyset$ ;
- $P = \emptyset$ ;
- $\mu = [1]_1$ ;
- $w_1 = S$ ;
- $R_1 = P \cup \{A \rightarrow A \mid A \in N\}$ ;
- $R'_1 = \emptyset$ ;
- $i_o = 1$ .

Clearly, the system  $\Pi$  generates exactly the Parikh image of the  $L(G)$  and this is true independently from the mapping  $e$  chosen.

Then, it seems natural to ask what we gain if we use a priority in the strong sense as introduced before. This problem seems particularly interesting because standard P systems (i.e., where the time of execution of each rule is fixed as 1 time-unit) using symbol objects, non cooperative rules and priority in the strong sense can generate at least the Parikh set of ETOL systems (see Theorem 3.4.4 in [5]).

The following example shows that basic  $P^a(e)$  systems, using priority and non-cooperative rules can generate the Parikh image of non semilinear languages. Notice that in this case the  $P^a(e)$  system is not time-free but only “partially” time-free: this means that, because the system generates always the same result, the time-mapping  $e$  cannot be arbitrary but it must fulfill some conditions. Anyway, the exact mathematical formalization of partially time-free is left as open problem.

**Example 5.1** *We take the following basic partially time-free  $P^a(e)$  systems  $\Pi$ :*

$$\Pi = (V, C, P, \mu, w_1, R_1, R'_1, i_o, e),$$

where:

- $V = \{A, T, T_1, T_2, a, T_3\}$ ;
- $C = \emptyset$ ;
- $P = \emptyset$ ;

- $\mu = [1]_1$ ;
- $w_1 = TA$ ;
- $R_1 = \{r_1 : T \rightarrow T_1, r'_1 : T \rightarrow T_2, r_2 : A \rightarrow A'A', r_3 : A \rightarrow a, r_4 : T_1 \rightarrow T'_1, r_5 : T_2 \rightarrow T'_2, r_6 : T'_1 \rightarrow T, r_7 : T'_2 \rightarrow T_3\}$ ;
- $R'_1 = \emptyset$ ;
- $i_o = 1$ ;
- $r_1 > \{r_2, r_3\}, r'_1 > \{r_2, r_3\}, r_4 > r_3, r_5 > r_2, r_3 > r_7, r_6 < r_2$ ;
- $e(r_6) = e(r_8), e(r_2) > e(r_4), e(r_3) > e(r_5)$ .

The system works in the following way. At the beginning only objects  $A$  and  $T$  are present in region 1.

Because of the priority relation the only rules that can be applied are  $r_1$  and  $r'_1$ . The applications of one of the two rules corresponds to the choice of the rule  $r_2$  or  $r_3$ . In fact, if  $T$  is changed to  $T_1$  (the other choice is similar), then, because of priority  $r_4 > r_3$ , the rule  $r_2 : A \rightarrow A'A'$  is applied while the rule  $A \rightarrow a$  is blocked. Rules  $r_4$  and rules  $r_2$  are executed in parallel. Because of the condition  $e(r_2) > e(r_4)$ , the rule  $r_4$  terminates before rule  $r_2$ . When rule  $r_4$  ends, the rule  $r_6$  might be applied, but, because of the priority  $r_6 < r_2$ , the rule  $r_6$  have to wait the end of  $r_2$  (still in execution) to be applied.

When  $r_2$  ends, rules  $r_6$  and  $r_8$  are applied in parallel and they terminate in the same time unit (because of the condition  $e(r_6) = e(r_8)$ ). Therefore,  $T$  is obtained again and all the objects  $A'$  are changed in  $A$  and the process can be iterated.

Hence, at the end of any halting computation, region 1 contains a number of objects  $a$  that is a power of 2. Then the set of numbers generated by  $\Pi$ , using an arbitrary function  $e$  that fulfills the conditions indicated, is  $\{2^n \mid n \in \mathbb{N}\}$ .

It seems difficult to delete the conditions imposed on the time-mapping  $e$  and then to make the  $P^a$  system time-free. This fact suggest us the following open problem:

**Open Problem** Is it possible to generate non semilinear sets using a basic time-free  $P^a$  system (i.e., using only non cooperative rules and a priority relation) ?

## 6 Time-Free $P^a$ Systems Using Catalysts

When we use one catalyst and signal-promoters then it is possible to synchronize the execution of the evolution rules in a way to simulate sequential grammar devices, even if the  $P^a$  system considered is time-free.

In particular, in this section we show how to simulate programmed grammars; we recall that a context-free *programmed grammar* with appearance checking (a.c.) is a construct  $G = (N, T, P, S)$ , where  $N, T, S$  are the set of non-terminals, the

set of terminals, and the start symbol, and  $P$  is a finite set of rules of the form  $(b : A \rightarrow x, E, F)$ , where  $b$  is a label,  $A \rightarrow x$  is a context-free rule over  $N \cup T$ , and  $E, F$  are two sets of labels of rules of  $G$  ( $E$  is called the *success field* and  $F$  the *failure field* of the rule). A rule  $(b : A \rightarrow x, E, F)$  is applied as follows: if  $A$  is present in the sentential form, then the rule is used and the next rule to be applied is chosen from those with label in  $E$ , otherwise, the sentential form remains unchanged and we choose the next rule from the rules labelled by some element of  $F$ , and try to apply it. If no failure field is given for any of the rules, then we obtain a programmed grammar without appearance checking.

We denote  $Lab(P) = \{b \mid (b : A \rightarrow x, E, F) \in P\}$ .

A context-free programmed grammar with a.c. can be also written in the form  $G = (N, T, S, R, \sigma, \varphi)$ , where  $N, T, S$  are defined as before,  $R$  is a set of context-free rules and  $\sigma$  and  $\varphi$  are mappings from  $R$  to the power set of  $R$ ;  $\sigma(p)$  is the success field of the rule  $p$  (this means that a rule in  $\sigma(p)$  must be used after successfully applying the rule  $p$ ), and  $\varphi(p)$  is the failure field (this means that a rule from  $\varphi(p)$  must be considered when  $p$  cannot be applied).

Sometimes, when no confusions arises, the two definitions are, somehow, “joined”: we consider a context-free programmed grammar with a.c. as a construct  $G = (N, T, P, S)$ , as defined before, where each production in  $P$  is of the kind  $(b : A \rightarrow x, \sigma(b), \varphi(b))$  ( $\sigma(b)$  is the success field of the rule (with label)  $b$  and  $\varphi(b)$  is the failure field of the rule (with label)  $b$ ).

In the following theorem we show how a time-free  $P^a$  system can simulate a programmed grammar without appearance checking using one catalyst, two membranes and signal-promoters. We know from [2] that such grammars generate non semilinear languages, hence whose Parikh image is not in  $PsCF$ . The family of languages generated by programmed grammars with  $\lambda$  rules and without appearance checking is denoted by  $PR$ .

**Theorem 6.1**  $PsPR \subseteq PsP_2^a(cat_1, *, free)$ .

*Proof.* Consider the programmed grammar  $G = (N, T, P, S)$  without appearance checking. We denote by  $l(S)$  the set of labels of rules of the form  $(k : S \rightarrow x, \sigma(k)) \in P$ . We add to  $P$  the triple  $(0 : U \rightarrow S, \sigma(0))$  with  $\sigma(0) = l(S)$ , where  $U$  is a new non-terminal. We denote by  $G'$  the obtained grammar  $(N', T, P', U)$ , where  $N' = N \cup \{U\}$ . Clearly,  $L(G) = L(G')$  and each derivation in  $G'$  starts with the rule with label 0.

We construct the  $P^a$  system

$$\Pi = (V, C, P, \mu, w_1, w_2, R_1, R_2, R'_1, R'_2, i_0),$$

where:

- $V = N' \cup T \cup P \cup \{A^i, \underline{A}^i \mid i \in lab(P')\} \cup \{A^{-1}, \#, Z, B, B_2, c\}$ ;
- $C = \{c\}$ ;
- $P = \{s, s'\} \cup \{i \mid i \in lab(P')\}$ ;

- $\mu = [1[2]2]_1$  ;
- $w_1 = ZsA^{-1}$ ;
- $w_2 = Ucs'$ ;
- $R_1 = \{Z \rightarrow Z\}$ ;
- $R'_1 = \{A^j \rightarrow \lambda|_{s,in} \mid j \in lab(P')\}$   
 $\cup \{A^j \rightarrow \underline{A}^i|_{i,in} \mid i \in \sigma(j)\}$   
 $\cup \{A^{-1} \rightarrow \underline{A}^0|_{0,in}\}$   
 $\cup \{\underline{A}^i \rightarrow A^j|_{i,here} \mid j \in \sigma(i)\}$   
 $\cup \{Z \rightarrow \lambda|_{s'}\}$ ;
- $R_2 = \emptyset$ ;
- $R'_2 = \{cX \rightarrow cw|_{i,out} \mid (i : X \rightarrow w, \sigma(i)) \in P'\}$   
 $\cup \{B \rightarrow B_2|_{s,here}, B_2 \rightarrow \lambda|_{s',out}\}$   
 $\cup \{X \rightarrow \#|_{s,here} \mid X \in N\}$ ;
- $i_0 = 2$ .

The constructed system simulates the programmed grammar  $G'$  in the following way. The rules  $\{cX \rightarrow cw|_{i,out} \mid (i : X \rightarrow w, \sigma(i)) \in P'\}$  present in region 2 simulate the context-free rules of  $G'$ . The rules present in region 1 are used to select the rules in region 2 following the order defined by the labels in the success field of  $G'$ . In the initial configuration, in region 2 is present the object  $U$  that is the axiom of  $G'$ . In region 1 is present the symbol-object  $A^j$  indicating that the last rule of  $G'$  simulated in region 2 has been the one with label  $j$  (at the beginning of the computation the object  $A^{-1}$  is present in region 1). The object  $A^j$  is changed to  $\underline{A}^i$  using one of the rules in  $\{A^j \rightarrow \underline{A}^i|_{i,in} \mid i \in \sigma(j)\}$ . When this rule is terminated the signal-promoter  $i$  is sent to region 2. This signal-promoter activates the rule  $cX \rightarrow cw|_{i,out} \mid (i : X \rightarrow w, \sigma(i)) \in P'$ ; this means that the rule of the grammar  $G'$  with label  $i$  can be simulated in region 2. Suppose the activated rule  $cX \rightarrow cw|_{i,out}$  can be applied (i.e., object  $X$  is present in region 2; the case when the rule cannot be applied will be discussed later).

When the execution of the rule  $cX \rightarrow cw|_{i,out}$  is terminated, the signal-promoter  $i$  is sent back to region 1. The presence of the signal-promoter in region 1 activates the rule  $\{\underline{A}^i \rightarrow A^j|_{i,here} \mid j \in \sigma(i)\}$ . One of the rules present in this set will be executed and then a new object  $A^j$  is obtained and the process can be iterated.

To halt the computation it is necessary to stop the evolution rule  $Z \rightarrow Z$  that runs in region 1. The only way to stop this rule is to delete the object  $Z$  by using the rule  $Z \rightarrow \lambda|_{s'}$  present in region 1. Therefore, it is necessary to introduce in region 1 the signal-promoter  $s'$ . This can be done only by using one the rules present in  $\{A^j \rightarrow \lambda|_{s,in} \mid j \in lab(P')\}$  and then sending the signal-promoter to region 2. This signal-promoter activates the rules to send  $s'$  to region 1 (and then to delete the  $Z$ ) and activates the rules  $\{X \rightarrow \#|_{s,here} \mid X \in N\}$  present in region 2. Therefore, the computation will halt if and only if only terminals are present in region 2. That means that the system  $\Pi$  generates exactly the Parikh set of the language generated

by grammar  $G'$ . Because in each time-unit only one rule is started or is in execution (except for the rules used to delete the  $Z$  in the final phase) then it is clear that, for any possible time-mapping  $e$ , the system  $\Pi$  generates always exactly the Parikh set of the language  $L(G')$ .  $\square$

We recall that a *bi-stable* catalyst  $c$  is a catalyst that has two states,  $c$  and  $\bar{c}$ , and the rules involving catalysts may switch between these states. That is, the allowed rules are of the forms  $ca \rightarrow cv$ ,  $ca \rightarrow \bar{c}v$ ,  $\bar{c}a \rightarrow \bar{c}v$  and  $\bar{c}a \rightarrow cv$ .

In Theorem 3.4.6 from [5] it is shown that by using bi-stable catalysts, targets and strong priority, the family of sets of numbers computed by *non-synchronized P systems* is exactly the family of recursively enumerable sets of natural numbers (non-synchronized P systems are introduced in [5], see section 3.4.5, and they are P systems with symbol-objects where in each region is present a rule  $a \rightarrow a$  for each symbol  $a$  in the alphabet).

Looking to the proof of Theorem 3.4.6 in [5] we can notice that the evolution rules of the constructed system are executed sequentially, that is, never more than one evolution rule is under execution in a certain step (this is true because of the use of the bi-stable catalysts and of the priority). In this way, it is clear that the same proof works also for basic  $P^a$  systems independently from the duration of the single evolution rules and then the same proof works also for basic time-free  $P^a$  systems.

In a formal way this means that

**Theorem 6.2**  $PsRE = PsP_2^a(2cat_*, 0, free, pri)$ .

On the other hand, as we can see in the next theorem, a basic time-free  $P^a$  system with bi-stable catalysts, strong priority and a structure of the form  $\mu = [1[2 \cdots [m]_m \cdots ]_2]_1$  can be simulated by a time-free  $P^a$  system using catalysts, signal-promoters, priority and at most  $m + 1$  membranes.

**Theorem 6.3**  $PsRE = PsP_3^a(cat_*, *, free, pri)$ .

*Proof.* To prove the theorem we show how a basic time-free  $P^a$  system

$$\Pi = (V, C, \emptyset, [1[2 \cdots [m]_m \cdots ]_2]_1, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m, \emptyset, \emptyset, \dots, \emptyset, i_o, e),$$

using bi-stable catalysts,  $m$  membranes and strong priority can be simulated by a time-free  $P^a$  system

$$\begin{aligned} \Pi' = & (V', C', P', [0[1[2 \cdots [m]_m \cdots ]_2]_1]_0, w_1, w_2, \dots, w_m, \\ & R_1, R_2, \dots, R_m, R'_1, R'_2, \dots, R'_m, i_o, e'), \end{aligned}$$

using catalysts,  $m + 1$  membranes, signal-promoters and strong priority. We suppose that the set of bi-stable catalysts used in  $\Pi$  is  $C = \{c_1, c_2, \dots, c_h\}$ .

The main point of the proof is to show how, using signal-promoters in  $\Pi'$ , is possible to simulate the bi-stable catalysts present in  $\Pi$ .

Without loss of generality we suppose that bi-stable catalysts present in different regions of  $\Pi$  are named differently.

For each rule  $r_{i<a\rightarrow w}^1 : \bar{c}_i a \rightarrow c_i w$  presents in region  $i$  of  $\Pi$  we add to  $R'_i$  of  $\Pi'$  the signaling-rule  $r_{i<a\rightarrow w}^1 : c_i a \rightarrow c_i w | \bar{p}_i, out$  and we add to  $R'_{i-1}$  of  $\Pi'$  the signaling-rules  $X_i \rightarrow X_i'' | \bar{p}_i, here$  and  $X_i'' \rightarrow X_i | p_i, in$ .

In a similar way, for each rule  $r_{i<a\rightarrow w}^2 : c_i a \rightarrow \bar{c}_i w$  presents in region  $i$  of  $\Pi$  we add the signaling-rule  $r_{i<a\rightarrow w}^2 : c_i a \rightarrow c_i w | p_i, out$  in  $R'_i$  of  $\Pi'$  and we add to  $R'_{i-1}$  of  $\Pi'$  the signaling-rules  $X_i \rightarrow X_i' | p_i, here$  and  $X_i' \rightarrow X_i | \bar{p}_i, in$ .

For each rule  $r_{i<a\rightarrow w}^3 : \bar{c}_i a \rightarrow \bar{c}_i w$  present in  $R_i$  of  $\Pi$  we add the signaling-rule  $r_{i<a\rightarrow w}^3 : c_i a \rightarrow c_i w | \bar{p}_i, here$  in  $R'_i$  of  $\Pi'$ ; for each rule  $r_{i<a\rightarrow w}^4 : c_i a \rightarrow c_i w$  presents in region  $i$  of  $\Pi$  we add the rule  $r_{i<a\rightarrow w}^4 : c_i a \rightarrow c_i w | p_i, here$  in  $R'_i$  of  $\Pi'$ .

We take the alphabet  $V' = V - C \cup \{X_i, X_i'', X_i' \mid 1 \leq i \leq h\}$ , the set of signal-promoters as  $P' = \{\bar{p}_i, p_i \mid 1 \leq i \leq h\}$  and the set of catalysts as  $C' = \{c_i \mid 1 \leq i \leq h\}$ .

For each non-cooperative rule  $r_{i<X\rightarrow w}^0 : X \rightarrow w$  presents in  $R_i$  of  $\Pi$  we add the non-cooperative rule  $r_{i<X\rightarrow w}^0 : X \rightarrow w$  to the  $R_i$  of  $\Pi'$ .

If in the system  $\Pi$  the rule  $r_{i<u\rightarrow w}^k$  has priority over rule  $r_{j<u\rightarrow w}^k$ ,  $k = 0, 1, 2, 3, 4$ , then in  $\Pi'$ , the rule  $r_{i<u\rightarrow w}^k$  has priority over the rule  $r_{j<u\rightarrow w}^k$ , for  $1 \leq j \leq m$ ,  $1 \leq i \leq m$ .

The idea is that the “change of state” of a bi-stable catalyst presents in region  $i$  of  $\Pi$  is simulated by an exchange of signal-promoters between region  $i$  and the surrounding region  $i - 1$  in  $\Pi'$ .

For instance (in all other cases the situation is similar) the execution of the rule  $r_{i<a\rightarrow w}^1 = \bar{c}_i a \rightarrow c_i w$  presents in region  $i$  of  $\Pi$  is simulated in  $\Pi'$  in the following way: first the rule  $c_i a \rightarrow c_i w | \bar{p}_i, out$  is executed in region  $i$ ; at the end of its execution the signal-promoter  $\bar{p}_i$  is sent out to the surrounding region  $i - 1$ . There, the two rules  $X_i \rightarrow X_i'' | \bar{p}_i, here$  and  $X_i'' \rightarrow X_i | p_i, in$  are executed sequentially and they send inside region  $i$  the signal-promoter  $p_i$ . In region  $i$  of  $\Pi'$  the presence of signal-promoter  $p_i$  activates now all (and only) the rules catalyzed by  $c_i$ ; in this way the simulation of the execution of rule  $r_{i<a\rightarrow w}^1 = \bar{c}_i a \rightarrow c_i w$  in  $\Pi$  has been completely simulated.

It is easy to see that, because  $\Pi$  is time-free then also  $\Pi'$  is time-free and they generate the same family of set of vectors of natural numbers. Therefore, because of Theorem 6.2, the statement is true. □

## 7 Concluding Remarks

We have introduced a class of P system called asynchronous P systems ( $P^a(e)$ ) which use signal-promoters and where to each rule is associated a time of execution according to a time-mapping  $e$ . Further, we have introduced time-free  $P^a$  systems that are  $P^a(e)$  systems producing always the same result independently of the time-mapping  $e$  used. We have shown that time-free  $P^a$  systems can generate the Parikh image of the languages generated by Indian parallel grammar using an unbounded number of signal-promoters. When we add the ability to use catalysts and priority (in the strong meaning) then time-free  $P^a$  systems become universal (the result has

been obtained simulating non-synchronized P systems that are known to be universal when using bi-stable catalysts and priority, [5]).

Not much has been found regarding basic time-free  $P^a$  systems that are time-free  $P^a$  systems not using signal-promoters. In this case we have shown that basic (partially) time-free  $P^a(e)$  systems, using non cooperative rules and priority, can generate non-semilinear set of vectors of natural numbers. The result has been obtained introducing conditions over the time of executions of the rules (from here the name partially time-free). Is it possible to get the same result for a basic time-free  $P^a$  system? It remains to formalize the concept of partially time-free  $P^a(e)$  system and to see which are the classes of partially time-free  $P^a$  that could be of interests (both for practical and mathematical reasons).

Other suggestion is to add other parameters to make a  $P^a$  system “more asynchronous” (and then more realistic); for instance, it would be very interesting to associate to each rule a time of delay that indicates the number of time-units to wait before a rule is started.

We believe that many interesting problems and result can be found in these lines of research.

**Acknowledgments.** The author is very thankful to Dragoş Sburlan for very interesting and useful discussion on the draft of this work.

## References

- [1] I.I. Ardelean, M. Cavaliere, S. Dragoş, Computing Using Signals: From Cells To P Systems, *Technical Report 01/2004 of RGNC, Brainstorming Week on Membrane Computing 2004*, University of Sevilla, Sevilla (2004).
- [2] J. Dassow, G. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin (1989).
- [3] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
- [4] H. Fernau, Parallel Grammars: A Phenomenology, *Grammars*, 6, 1, 2003, 25–87.
- [5] Gh. Păun, *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.