Clock-free P systems

Dragoş SBURLAN

Faculty of Mathematics and Informatics
Ovidius University of Constanţa, România
E-mail:dsburlan@univ-ovidius.ro

Abstract

We present a model of P system that computes without the help of the universal clock. Deterministic computational universality is obtain even when "low-sensitive" components are used (non-cooperative rules with promoters and one catalyst).

1 Introduction

Reflected on the rapid advances in microelectronics and computer technology, biology and medical technology, as well as in many other domains, the field of digital signal processing is one of the emerging fields in computer science.

Contemporary signal processing is more likely to involve discrete-time signals than continuous-time signals. In general, for a classical computational device, instead of time continuum, a discrete clock globally regulates the computation. Signals have values at the discrete clock ticks, but not in between. In literature is known that multirate systems involve multiple clocks, but with clear relationships among them, so that ticks in the multiple clocks can be unambiguously associated.

However, signal processing has its intellectual roots in circuit theory. We know also that algorithms tend to be modeled as compositions of components that conceptually operate concurrently and communicate via signals that are functions of time. The components are typically filters, which transform the input signals to construct output signals.

Components can resides on the same computational device or may exists a distribution of components on a graph structure for instance (in particular a tree structure); these components exchange signals based on certain events that occur. Traditionally, the signals are continuous functions of the time continuum.

Independently from above considerations, and having as objective a new, bio-inspired, computational paradigm, Gheorghe Păun has proposed the field of membrane computing. The formal model of a cell with its underlying tree structure of embedded vesicles (components in a sense) was intended to be regulated by an universal discrete clock - in this respect, running processes (which model bio-chemical reactions) in the same or in different regions can be synchronized in an intuitive manner.

The limitation of the knowledge about the biology of living cells made the advances in the bio-mathematics/bio-informatics to be rather slow. The formal models created lately were chosen to be as much as possible close to the biological reality. However, the trend in the underlying formal theory is to use simplified techniques (for instance, the usage of context-free object rewriting rules and fewer and fewer "context-sensitive" components, or the usage of a global clock). The advantages of simplified models are quite obvious. At very last, the simulations will be much easier; at best the analysis of the behavior of the living cell becomes possible.

Nevertheless, some questions arise in this framework. After all, the final goal of a formal theory is to explain the reality. Moreover, some important details that are avoided may induce different behavior and, therefore, the simplified model will have nothing in common with the "natural" one. Also, the analysis of the simplified model does not offer always a good overview of the whole biological system. Therefore, the best way to avoid the traps inherited from simplified models is to maintain strong bonds with the biology and its feasible experiments.

It is then natural from a biological point of view to try to generalize the original definition of P systems and to consider reactions that have different time of execution. A variant could be that each rule has as reaction time a multiple of a global unit. One can see that such a model can easily simulate the original P system model.

A natural question arises: what would be the computational power of systems where there is no internal clock at all? ...and this brings us to the scope of this paper.

Here we propose a type of P systems where the time to execute a reaction is not anymore a natural number but a real number. Moreover, we may further consider that the time to execute a reaction is a real, finite and positive, variable. Just to have a simple argument in the support of such a hypothesis, we can imagine that a certain reaction whose duration depends on the temperature will behave differently even in the same cell region (recall that the propagation of heat is not uniform).

In general, we may assume that from the cell/bio-system perspective it is not so important whether the time can be expressed in integer or in real numbers. However, this is especially important from the observer point of view (the human, for example, who tries to explain and to use the system features). If we consider, for instance, that the duration of a bio-reaction is an irrational number, then, without approximations, a human observer will never be able to interpret/use a number with an infinity of decimals. Here, we arrive to another computational dilemma: what is "better" from the observer point of view? to approximate at each step/locally or to approximate at the end/globally. Of course, the answer depends on the model we use: for example, in silicon computers is better to approximate the voltage of current that enters in a circuit, while in quantum computing any measure will destroy the experiment, therefore we have to measure at the end. The same problem can be formulated to the reaction time of a certain bio-reaction. In biology, when we measure the reaction time of a certain chemical reaction, we depend on the sensitivity of the instruments we use, therefore natural/rational numbers seems to be more fitted to create formal models (because they is easier to approximate/use).

We will prove that these kind of P systems is computational universal when we use non-cooperative rules, promoters at the level of rules and only one catalyst.

In fact, we will see that the reaction time of a rule does not have a significant importance and can be avoided; the things that we have to demand are that the execution time for a rule is not infinite (a reasonable request) and that there exist rules that act at the same time (for example, the rules that are applied at the beginning of computation) even if they are spatially separated (a questionable claim; recall that the simultaneity on spatially separated events is not a primitive concept, also because we cannot be simultaneously present at both events and so we cannot make the same judgment we can make for local events).

However, knowing that a bio-system is a dynamical system which obeys in a certain sense to the principle of causality (cause$\Rightarrow$ effect $\stackrel{instant}{=}$ new cause$\Rightarrow$new effect$\Rightarrow \cdots$), we will consider that objects that are the result of the same reaction will further evolve synchronously if they can (in a sense they are local events; the time for all events at the same place is defined by one single clock at that place).

From this point of view one can use for reaction time, under the same mentioned constraints, an arbitrary finite value from $\mathbb{N}$, Q_+ or $\mathbb{R}_+$, but the result we prove remains valid. For the sake of generality, we will consider in this paper that reactions times have real number values.

A final remark which links digital signal processing to the P systems (and hence with cell biology as well) is that in the model presented, promoters play the role of signals, indicating when certain reactions have to run.

2 Preliminaries

In this section we will give some notions regarding register machines and their computational power.

An n-register machine is a construct $M = (n, P, i, h)$ where:

- n is the number of registers;
- P is a set of labeled instructions of the form $(j : op(r), k, l)$ where $op(r)$ is an operation on register r of M; symbols j, k, l belong to the set of labels associated in a one-to-one manner with instructions of P;
- i is the initial label;
- h is the final label.

The instructions allowed by an n-register machine are:

- $(e : inc(r), f, z)$ – add one to the contents of register r and proceed to instruction f or to instruction z ($f = z$ for the deterministic variant);
- $(e : dec(r), f, z)$ – jump to register z if the register r is null; otherwise subtract one from register r and jump to instruction labeled f;
- $(h : halt)$ – finish the computation. This is a unique instruction with label h.

If a register machine $M = (n, P, i, h)$, starting from the instruction labeled i with all registers being empty, stops by halting with value n_j in every register j, $1 \leq j \leq k$, and the contents of registers $k+1, \cdots, n$ being empty, then it generates the vector $(n_1, \cdots, n_k) \in \mathbb{N}^k$. Any recursively enumerable numeric vector set can be generated by a register machine.

A register machine $M = (n, P, i, h)$ accepts a vector $(n_1, \cdots, n_k) \in \mathbb{N}^k$ iff, starting from the instruction labeled i, with register j having value n_j for $1 \leq j \leq k$, and the contents of registers $k+1, \cdots, n$ being empty, the machine stops by the *halt* instruction with all registers being empty. *Deterministic* register machines can accept the family of all recursively enumerable sets of numeric vectors.

3 Clock-free P Systems

Definition 3.1 *A clock-free P system (of degree $m \geq 1$) with symbol objects and rewriting evolution rules is a construct*

$$\Pi = (V, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where

- *V is an alphabet; its elements are called objects;*
- *$C \subseteq V$ is a distinguished subset of the alphabet, called the set of catalysts;*
- *μ is a membrane structure consisting of m membranes usually labeled $1, 2, \ldots, m$;*
- *w_i, $1 \leq i \leq m$, specify the multisets of objects present in the corresponding regions $i, 1 \leq i \leq m$, at the beginning of a computation;*
- *$R_i = \{(r_{(i,1)}, t_{(i,1)}), (r_{(i,2)}, t_{(i,2)}), \cdots, (r_{(i,k_i)}, t_{(i,k_i)})\}$, $1 \leq i \leq m$, are finite sets of pairs (evolution rules over V, associated reaction time) corresponding to regions $1, 2, \ldots, m$ of μ; these evolution rules are of the non-cooperative form $a \rightarrow v$ or catalytic rules $ca \rightarrow cv$, where $c \in C$, a is an object from $V \setminus C$ and v is a string over*

  $$(V \setminus C) \times (\{here, out, in\});^{1}$$

  All rules can be promoted by certain symbols (the corresponding rules can be executed only in their presence). An object a is a promoter for a rule $u \rightarrow v$, and we denote this by $u \rightarrow v|_a$, if the rule is active only in the presence of object a. In particular, promoters themselves can evolve according to some rules.

  For a rule $r_{(i,j)}$ the ***finite, arbitrary real number*** *$t_{(i,j)} \in \mathbb{R}$ represents the reaction time. In a given configuration, at each application of rule $r_{(i,j)}$ on the present objects, the reaction time may be different.*

[1] For simplicity we will use subscripts associated with objects in order to specify the destinations of objects.

- *i_0 is a number between 0 and m and specifies the output membrane of Π (in case of $i_0 = 0$, the environment is used for the output).*

The result of using a rule $u \to v$ is determined by v:

- *if an object a appears in v in a pair $(a, here)$, then it will remain in the same region;*
- *if an object a appear in v in a pair (a, out), then it will exit the region to become an element of the immediately upper region;*
- *if an object a appear in v in a pair (a, in_q) then it will be added to the multiset corresponding to region q, providing that a is adjacent to the membrane q.*

We do not know what is the execution time of a certain rule $r_{(i,j)} : u_i \to v_i$; for generality, we have considered that the reaction time of a certain rule is a real arbitrary, finite variable. Moreover, we have considered that each application of rule $r_{(i,j)}$ on the objects from a certain configuration may have a different execution time. All objects from v_i will react in the same time if they can (because they appear in the same time) and as soon as possible (if there are proper existing conditions the reactions will not be delayed).

Starting from an initial configuration, *the system evolves according to the rules and objects present in the membranes, in a non-deterministic maximally parallel manner. The m-tuple of multisets of objects present at any moment in the m regions of Π constitutes the* configuration *of the system at that moment. The m-tuple $(w_1, w_2, \cdots, w_m)$ constitutes the initial configuration of Π.*

Recall that because there is no discrete clock, starting from a given configuration, we will consider the next configuration the instant description of the system when the output of a previous initiated reaction appears.

The objects that can evolve from a given configuration as well as the rules by which they evolve are chosen in a non-deterministic manner and also, in case of rules, in a maximal parallel manner. This means that we assign objects to rules, non-deterministically choosing the rules and the objects assigned to each rule, but in such a way that after this assignation, no further rule can be applied to the remaining objects. The objects which remain unassigned are left where they are, and they are passed unchanged to the next configuration.

The system will make a successful computation if and only if it halts: there is no rule applicable to the objects present in the halting configuration. The result of a successful computation is the number of objects present in the output membrane in a halting configuration of Π, providing that in the system there are not running reactions (recall that the reaction time for a rule is a real variable). If the computation never halts, then we will have no output.

We will use the following notation:

$$N\bar{c}IP_m(\alpha, \beta), \alpha \in \{ncoo\} \cup \{cat_k \mid k \geq 0\}, \beta \in \{proR\}$$

to denote the family of sets of natural numbers generated by clock-free P systems with input, having at most m membranes, evolution rules that can be non-cooperative ($ncoo$), or catalytic (cat_k), using at most k catalysts, and promoters ($proR$) at the level of rules.

Also, we may consider as the result of a halting computation the vector $\Psi(w)$ (the vector of multiplicities of objects) where w is the multiset present in the region i_0 in the halting configuration. In this case, the set of all vectors constructed in this way by a system Π is denoted by $Ps(\Pi)$.

Just to have an idea of how such a system works we may consider that in a region of a P system we have the multiset a^n and following rules:

$a \rightarrow x_1 y_1,$
$a \rightarrow x_1 q_1,$
$y_1 \rightarrow y_2,$
$q_1 \rightarrow q_2.$

Initially, both rules $a \rightarrow x_1 y_1$ and $a \rightarrow x_1 q_1$ can be applied. Let us suppose that actually both rules were applied. After a while objects x_1 and y_1 (or x_1 and q_1) will appear simultaneously. It is not necessary for objects y_1 and q_1 to appear simultaneously. In conclusion, we cannot control when actually objects y_2 and q_2 will appear.

Another relevant fact concerning the synchronization of the execution of two rules is exemplified by the following rules:

$a \rightarrow \alpha|_b$
$b \rightarrow \beta|_a$

If in the region there are just objects a (and no objects b) then no rule can be applied. However, if some objects b appear then both rules can and will be applied.

For a more elaborate example which has relevance from the classical definition of P systems point of view, in Figure 1 we have a particular clock-free P system which uses non-cooperative promoted rules to generate in the output region $a^{2^n}, n \geq 1$. The interest for this system comes from the following fact: instead of considering that each application of rule R_i on the objects from a certain configuration has a different execution time, we have considered that the execution time is the same (i.e., supposing that we have the multiset $\{(a,n)\}$ and the rule $a \rightarrow b$, then after a while we will have the multiset $\{(b,n)\}$, with all objects b appearing in the same moment).

Formally, we define the following clock-free P system

$$\Pi_{A^{2n}} = (V, C, \mu, w_1, w_2, R_1, R_2, 2),$$

where:

$$\begin{aligned} V &= \{A, a, p, r\}, \\ C &= \emptyset, \end{aligned}$$

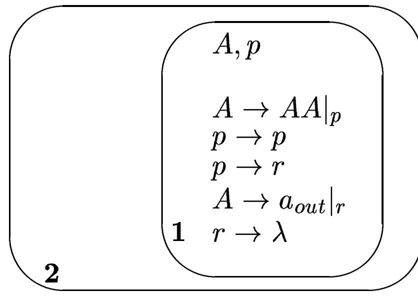


Figure 1: Generating $a^{2^n}, n \geq 1$

$$
\begin{aligned}
\mu &= [_2[_1\ ]_1]_2, \\
w_1 &= \{A, p\}, \quad w_2 = \emptyset, \\
R_1 &= \{(A \to AA|_p, t_{(1,1)}),\ (p \to p, t_{(1,2)}),\ (p \to r, t_{(1,3)}),\ (A \to a_{out}|_r, t_{(1,4)})\} \\
&\cup\ \{(r \to \lambda, t_{(1,5)})\},\ \text{with } t_{(1,i)} \in I\!R, 1 \leq i \leq 5,\ \text{finite arbitrary variable}, \\
R_2 &= \emptyset.
\end{aligned}
$$

Here, while object p is in region 1 the rule $A \to AA|_p$ is executed and because of the maximal parallel manner the number of objects A is doubled at each iteration. If the rule $p \to r$ is executed, then, after a while, we will have in region 2 the object r; therefore, the rule $A \to a_{out}|_r$ can be started and $2^n, n \geq 1$, copies of object a are expelled into the output region.

The second example reflects more accurately the capabilities of clock-free P systems. The system generates in the output region $a^{2^n} + 1, n \geq 1$, but considering that each application of rule r_i on the objects from a certain configuration has a different execution time. The example anticipates the main result of the paper. Recall that in the previous example the massive parallelism has played a significant role because of the way we have considered the application of a rule. In this more general approach, this is not anymore possible since we have to known when a certain rule finishes its execution (recall the above example where we have known that all objects A were transformed in the same time and moreover, the results have appeared in the same time because the reaction time was the same). The trick will be to sequentially transform each A into BB, then, again sequentially, transform B to A, and iterate the process up to a non-deterministic chosen moment when we brake the computation.

Formally, we define the following clock-free P system

$$\Pi_{A^{2^n+1}} = (V, C, \mu, w_1, R_1, 1),$$

where:

$$
\begin{aligned}
V &= \{S, S_1, S_2, \overline{S_1}, \overline{S_2}, \overline{T}, \overline{F}\}, \\
C &= \{c\}, \\
\mu &= [_1\ ]_1,
\end{aligned}
$$

$$
\begin{aligned}
w_1 &= \{A, S, c\},\\
R_1 &= \{(S \to S_1, t_{(1,1)}),\ (S \to \lambda, t_{(1,2)}),\ (cA \to cBBF|_{S_1}, t_{(1,3)})\}\\
&\cup\ \{(S_1 \to S_2T|_F, t_{(1,4)}),\ (F \to \lambda|_{S_1}, t_{(1,5)}),\ (S_2 \to S_1|_A, t_{(1,6)})\}\\
&\cup\ \{(T \to T_1, t_{(1,7)}),\ (S_2 \to \overline{S_1}|_{T_1}, t_{(1,8)}),\ (T_1 \to \lambda, t_{(1,9)})\}\\
&\cup\ \{(cB \to cA\overline{F}|_{\overline{S_1}}, t_{(1,10)}),\ (\overline{S_1} \to \overline{S_2}\overline{T}|_{\overline{F}}, t_{(1,11)}),\ (\overline{F} \to \lambda|_{\overline{S_1}}, t_{(1,12)})\}\\
&\cup\ \{(\overline{S_2} \to \overline{S_1}|_B, t_{(1,13)}),\ (\overline{T} \to \overline{T_1}, t_{(1,14)}),\ (\overline{S_2} \to S|_{\overline{T_1}}, t_{(1,15)}),\\
&\quad (\overline{T_1} \to \lambda, t_{(1,16)})\},\ \text{with } t_{(1,i)} \in I\!R, 1 \le i \le 16,\ \text{finite arbitrary variable.}
\end{aligned}
$$

Rules $S \to S_1$, $S \to \lambda$ compose a selector, i.e., they decide whether the generation should stop or continue. In case generation should continue ($S \to S_1$ is applied), then, after the appearance of object S_1, the rule $cA \to cBBF|_{S_1}$ is executed. It will take an arbitrary (but finite) time up to the moment when objects B and F appear simultaneously. In that moment, rules $S_1 \to S_2T|_F$, $F \to \lambda|_{S_1}$ can be applied and will be applied, again, simultaneously. We have the following situation: object F will be eventually deleted so it will not count in further computations; objects S_2 and T will appear synchronously. After this, we know for sure that reaction $T \to T_1$ will be executed; however we do not know if it starts the reaction in the same time with reaction $S_2 \to S_1|_A$ (because we do not know if all objects A where already transformed). If there still exist objects A, then object S_1 is generated and object T_1 will be eventually deleted. Otherwise, object $\overline{S_1}$ is generated and we can start the transformation of B into A (with a quite similar construction).

If the generation should stop, then object S is deleted ($S \to \lambda$), the transformation of objects A into B will not happen anymore and the computation halts.

4 Universality

Here, we present a universality result concerning clock-free P systems with promoters at the level of rules. The proof is based on the simulation of a register machine.

Theorem 1 $Ps\overline{c}IP_2(cat_1, proR) = PsRE.$

Proof. In order to prove this assertion we will simulate an n–register machine $M = (n, P, i, h)$. At each time during the computation, the current contents of register j is represented by the multiplicity of the object a_j.

Formally, we define the P system

$$\Pi = (V, C, [_1\ [_2\ ]_2\ ]_1, w_1 = \emptyset, w_2, R_1 = \emptyset, R_2, 1),$$

where:

$$
\begin{aligned}
V &= \{a_j, A_j, D_j \mid 1 \le j \le n\} \cup \{R, T, P, P_1\} \cup \{e, e_1 \mid (e : add(j), f) \in P\}\\
&\cup\ \{e, e_1 \mid (e : sub(j), f, z) \in P\},\\
C &= \{c\},\\
w_2 &= \{c, e, a_j^{k_j}, 1 \le j \le n, k_j \in N\},
\end{aligned}
$$

and R_2 is defined as follows:

- for each instruction $(e : add(j), f) \in P$, we add to R_2 the rules:

  $e \to e_1 A_j \quad , t_{(e,1)}$
  $c \to c a_j R|_{A_j} \quad , t_{(e,2)}$
  $A_j \to \lambda \quad , t_{(e,3)}$
  $e_1 \to f|_R \quad , t_{(e,4)}$
  $R \to \lambda \quad , t_{(e,5)}$

  with $t_{(e,i)} \in I\!\!R$, $1 \leq i \leq 5$ finite arbitrary variable;

- for each instruction $(e : dec(j), f, z) \in P$, we add to R_2 the rules:

  $e \to e_1 D_j T P \quad , t_{(e,1)}$
  $c a_j \to c R|D_j \quad , t_{(e,2)}$
  $T \to \lambda|_a \quad , t_{(e,3)}$
  $T \to \lambda|_{P_2} \quad , t_{(e,4)}$
  $P \to P_1 \quad , t_{(e,5)}$
  $P_1 \to \lambda|_R \quad , t_{(e,6)}$
  $D_j \to \lambda \quad , t_{(e,7)}$
  $e_1 \to f|_R \quad , t_{(e,8)}$
  $e_1 \to z|_{P_2} \quad , t_{(e,9)}$
  $R \to \lambda|_{P_1} \quad , t_{(e,10)}$
  $P_1 \to P_2|_T \quad , t_{(e,11)}$
  $P_2 \to \lambda \quad , t_{(e,12)}$

  with $t_{(e,i)} \in I\!\!R$, $1 \leq i \leq 12$ finite arbitrary variable;

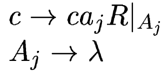
- for instruction $(h : halt)$ we add to R_2 the rules:

  $a_1 \to a_{1_{out}}|_h \quad , t_{(h,1)}$
  $\cdots$
  $a_k \to a_{k_{out}}|_h \quad , t_{(h,k)}$
  $h \to \lambda \quad , t_{(h,k+1)}$

  with $t_{(h,i)} \in I\!\!R$, $1 \leq i \leq k+1$ finite arbitrary variable.

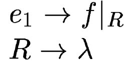
Before starting to examine the work of Π, let us mention some useful facts: objects e, f, z correspond to the register machine instruction labels e, f and z respectively; the multiplicity of object a_j represents the number stored in register j; object A_j represents the incrementation command (it corresponds to *add* in register machine definition); object D_j represents the decrementation command (it correspond to *sub* in register machine definition).

Here is how the simulation of the register machine increment instruction $(e : add(j), f) \in P$ works. Suppose that the current configuration of the P system is represented by the multiset $\{(c, 1), (a_1, n_1), \cdots, (a_j, n_j), \cdots, (a_k, n_k), (e, 1)\}$. Obviously, only the rule $e \to e_1 A_j$ can be applied. We will have as a result after a while the multiset $\{(c, 1), (a_1, n_1), \cdots (a_k, n_k), (e_1, 1), (A_j, 1)\}$; therefore the rules to be further applied are



$c \to ca_j R|_{A_j}$
$A_j \to \lambda$

Now, both rules have started the reaction run in the same time. Object A_j will be deleted so we do not have to worry when this will actually happen. After some time, objects a_j and R will appear; the configuration will be then $\{(c,1),(a_1,n_1),\cdots,(a_j,n_j+1)\cdots(a_k,n_k),(e_1,1),(A_j,1)\}$. Next, the rules to be executed simultaneously will be

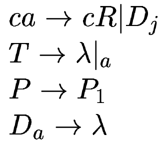


$e_1 \to f|_R$
$R \to \lambda$

Because we have obtained the next instruction label and moreover we deleted useless objects, we have correctly simulated the register machine increment instruction.

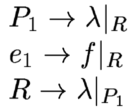
For the decrement instruction the procedure is much more complicated because we have to check when to accomplish a sensitive task – depending whether the indicated register is empty we should decrease its content and jump to the specified label.

Suppose that the current configuration of the system is represented by the multiset $\{(c,1),(a_1,n_1),\cdots,(a_j,n_j),\cdots(a_k,n_k),(e,1)\}$. As it can be seen, only the rule $e \to e_1 D_j TP$ can be applied. This means that after a while we will have the multiset $\{(c,1),(a_1,n_1),\cdots,(a_j,n_j),\cdots(a_k,n_k),(e_1,1),(D_j,1),(T,1),(P,1)\}$ (recall that objects e_1, D_j, T, and P have appeared simultaneously since they have came from the "same" object e. Next, the rules



$ca \to cR|D_j$
$T \to \lambda|_a$
$P \to P_1$
$D_a \to \lambda$

will be executed in the same time. However, we do not know if the objects R, T_1 or P_1 will appear simultaneously. But what we do know is the following: if object P_1 appears before object R, then it cannot react by the rule $P_1 \to P_2|_T$ because object T is missing (object T is/was involved in the reaction $T \to \lambda|_a$); only the rule $P_1 \to \lambda|_R$ can be further applied, but only after object R has appeared. If object R appears into region then following rules are applied:



$P_1 \to \lambda|_R$
$e_1 \to f|_R$
$R \to \lambda|_{P_1}$

In this way, the next instruction label f is generated and the simulation can continue. We do not need to know when objects P_1 and R will be removed since they cannot be involved in other reactions.

Now, suppose that the current configuration of the system is represented by multiset $\{(e,1),(c,1),(a_1,n_1),\cdots,(a_{j-1},n^{j-1}),(a_{j+1},n^{j+1}),\cdots,(a_k,n_k)\}$. As above, only the rule $e \to e_1 D_j TP$ can be applied. This means that after a while we will have the multiset $\{(c,1),(e_1,1),\ (D_j,1),\ (T,1),\ (P,1),\ (a_1,n_1),\ldots,(a_{j-1},n^{j-1}),$ $(a_{j+1},n^{j+1}),\cdots,(a_k,n_k)\}$. Next, the rules

$P \to P_1$
$D_a \to \lambda$

will start their execution simultaneously. Once the object P_1 has appeared the rule to be further applied is

$P_1 \to P_2|_T$

If object P_2 has appeared, then the system will execute the rules

$T \to \lambda|_{P_2}$
$P_2 \to \lambda$
$e_1 \to z|_{P_2}$

After some time the next instruction label will be generated and the computation can continue. Finally, if object h is generated, then all the following rules are executed:

$a_1 \to a_{1_{out}}|_h$
$\dots$
$a_k \to a_{k_{out}}|_h$
$h \to \lambda$

In conclusion, we have shown that $Ps\bar{c}IP_2(cat_1, proR) \supseteq PsRE$. By Turing-Church thesis we have the reverse inclusion, and this concludes the proof. □

5 Concluding Remarks and Open Problems

A new, more general, type of P systems has been introduced in this paper. The main idea was to consider clock-free P systems, i.e., systems where the execution time of each rule is a real variable. From this perspective, the notion of a universal clock that regulates the computations does not have anymore sense. The universality result presented here was obtained by creation/deletion of signals (promoters in our case) which activate different components (sets of rules).

Several open problems arise. For example, it is not known if the clock-freeness feature can be adapted to other types of P systems in order to obtain computationally universal devices. Also, in this paper we have considered that objects that can react will be involved in their corresponding reactions in the same time. But this, from a biological point of view, is not feasible because, for instance, when considering a cooperative rule $ca \to \alpha$ it is not guaranteed that object a will instantly meet object c and react (we may consider that they are physically far enough not to find each other).

However, we believe that this kind of clock-free P systems represents a new step toward a more bio-realistic P system model.

Acknowledgements. I gratefully acknowledge the discussions with Matteo Cavaliere about the topic of this paper.

References

[1] Bottoni, P., Martín-Vide, C., Păun, Gh., Rozenberg, G.: Membrane Systems with Promoters/Inhibitors, *Acta Informatica*, 38, 10 (2002), 695–720.

[2] Ionescu, M., Sburlan, D.: On P Systems with Promoters/Inhibitors, submitted, 2004.

[3] Marcus, S.: *Timpul*, Albatros, Bucuresti, 1985.

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

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

[6] Păun, Gh., Rozenberg G.: A Guide to Membrane Computing, *Theoretical Computer Science*, 287, 1 (2002), 73–100.

[7] Rozenberg, G., Salomaa, A.(eds.): *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.