Time-Driven Computations in P Systems

Matteo Cavaliere[1], Claudio Zandron[2]

[1] Microsoft Research - University of Trento
Centre for Computational and Systems Biology, Trento, Italy
`matteo.cavaliere@msr-unitn.unitn.it`

[2] Università degli Studi di Milano - Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
`zandron@disco.unimib.it`

Summary. It is a well-known fact that the time of execution of a (biochemical) reaction depends on many factors, and, in particular, on the current situation of the whole system. With this motivation in mind, we propose a model of computation based on membrane systems where the various rewriting rules have different times of execution and, moreover, the time of execution of each rule can vary during the computation, depending on the configuration of the whole system (in this sense, the computation is "time-driven"). We show that such systems are universal in a very simple framework: a regular time-mapping suffices to obtain universality for systems with minimal cooperation (one catalyst).

1 Time-Driven Computations

In [3] a new class of membrane systems, called *timed membrane systems* have been introduced. The main idea of timed membrane systems (called also timed P systems) is to associate, via a mapping, to each rule of the system a certain time of execution. The motivations for this model come from the fact that, in a living cell, different chemical reactions may take different times to be executed.

Actually, the time of execution of a chemical reaction may not be a constant but can change according to the state of the system where the reaction takes place. For instance, a reaction can be executed in a faster or slower way if certain substances are present. Motivated by this fact, it is natural to consider a model of timed P systems where the time of execution of the rules changes dynamically, according to the state of the system. In fact, it is quite common in biochemical experiments to control the whole reaction process by modifying the conditions in which the system operates, in order to speed-up certain reactions and to delay other reactions. In this way, one can safely assume that some reactions will take place only when some chemicals have been produced or consumed by other processes.

We formalize this notions by introducing contents-timed P system, that is, P systems where the time of execution of the rules is defined by a mapping depending on the contents of the regions of the system; in other words, the time of execution depends on the current configuration of the system.

We prove that time can indeed be a powerful resource for computation, when it can be "programmed" and it is used together with maximal parallelism: universality can be obtained with a quite simple system, using a regular time-mapping and a single catalyst.

The rest of the paper is organized as follows. In Section 2 we give all formal language notions we will use in the paper. In Section 3 we formally define contents-timed P systems and we gives some preliminary results. In Section 4 we show how a time-driven P system works, by means of an example. Then, in Section 5 we give a universality results for time-driven P systems with regular time mappings. We conclude the paper with Section 6, where some open problems and directions for future research are given.

2 Formal Languages Preliminaries

We will shortly recall the main elements of formal language theory used in the paper. For more detailed information the reader can consult the standard books and monographs in the area (as, for instance, [4], [8], and [7]).

As usual, V^* denotes the set of all strings over the alphabet V. For $a \in V$ and $x \in V^*$ we denote by $|x|_a$ the number of occurrences of a in x. Then, for $V = \{a_1, \dots, a_n\}$, the *Parikh mapping* associated with V is the mapping on V^* defined by $\Psi_V(x) = (|x|_{a_1}, \dots, |x|_{a_n})$ for each $x \in V^*$. The family of recursively enumerable languages is denoted by RE, and the family of Parikh images of RE languages is denoted by $PsRE$ (this is the family of all recursively enumerable sets of vectors of natural numbers). The family of all recursively enumerable sets of natural numbers is denoted by NRE.

The multisets over a given finite support (alphabet) are represented here by strings of symbols. The order of symbols does not matter, because the number of copies of an object in a multiset is given by the number of occurrences of the corresponding symbol in the string.

A context-free *programmed grammar with appearance checking* is a construct $G = (N, T, S, P)$, where N (T, resp.) is a finite set of nonterminals (terminals, resp.), $S \in N$ is the start symbol, and P is a finite set of productions of the form $(b : A \to u, E_b, F_b)$, where b is a label, $A \to u$ with $A \in N$ and $u \in (N \cup T)^*$ is a context-free production, and E_b, F_b are two sets of labels of productions of G (E_b is called the *success field* and F_b the *failure field* of the production). A production $(b : A \to u, E_b, F_b)$ is applied as follows: if A is present in the sentential form, then the production is applied and the next production to be applied is chosen from those with the labels in E_b, otherwise, the sentential form remains unchanged and we choose the next production from the rules labeled by some element of F_b. A derivation step is denoted by $\Rightarrow$ while $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

If no failure field is given for any of the productions, then we obtain a programmed grammar *without appearance checking.*

We denote the set of labels as $Lab(P) = \{b \mid (b : A \rightarrow u, E_b, F_b) \in P\}$. Also, for $A \in N$, we define $l(A) = \{b \mid (b : A \rightarrow u, E_b, F_b) \in P\}$.

In this paper we use the following normal form for programmed grammars with appearance checking.

Theorem 1. *For any programmed grammar G [with appearance checking] there exists a programmed grammar G' [with appearance checking, respectively] such that there is a unique initial production applied (with label l_0), the last production applied in G' is an erasing production $Z \rightarrow \lambda$ (with label l_h) and $L(G') = L(G)$. The unsuccessful derivations in G' halts in sentential forms from $Z(N \cup T)^*$ if G' is without appearance checking, or from $(N \cup T)^* N (N \cup T)^*$ if G' is with appearance checking.*

By PR we denote the family of languages generated by programmed grammars without appearance checking, and by PR_{ac} we denote the family of languages generated by programmed grammars with appearance checking. Proofs of the following results can be found in [4].

Theorem 2. $CF \subset PR \subset PR_{ac} = RE$.

3 Contents-Timed P Systems

As mentioned in the Introduction, a contents-timed P system is essentially a P system where the time of execution of the rules depends on the configuration in which they are applied.

We start from the standard definition of P system with symbol-objects and catalytic, non-cooperative evolution rules (we suppose the reader familiar with the notions of P systems, as, for instance, presented in [6]; many further information concerning P systems (including an up–to–date bibliography) can be found in [9]).

Definition 1. *A P system (of degree $m \geq 1$) with symbol-objects is a construct*

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

where:

- *O is the alphabet of Π; its elements are called objects;*
- *$C \subseteq V$ is the set of catalysts;*
- *μ is a membrane structure consisting of m membranes labeled $1, 2, \ldots, m$;*
- *w_i, $1 \leq i \leq m$, specifies the multiset 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 O associated with regions $1, 2, \ldots, m$ of μ; the rules can be non-cooperative, that is, of the form $a \rightarrow v$, where a is an object from $O - C$ and v is a string over $\{a_{here}, a_{out}, a_{in_j} \mid a \in O - C, 1 \leq j \leq m\}$, or they can be catalytic rules, of the form, $ca \rightarrow cv$, where a is an object from $O - C$, v is a string over $\{a_{here}, a_{out}, a_{in_j} \mid a \in O - C, 1 \leq j \leq m\}$, and $c \in C_b$; if the target indication is not specified, then it is intended to be here;*
- *$i_0 \in \{0, 1, \ldots, m\}$ is the output region; if $i_0 = 0$, then it indicates the environment.*

A *configuration* of Π can be *represented* by a string of parentheses (the structure μ) and strings over O (contents of the regions). For instance, a possible configuration of the system Π with alphabet $O = \{a, b\}$ and structure $\mu = [_1[_2\ ]_2]_1$ is represented by the string: $[_1aab[_2a]_2]_1$. Given a string w representing a configuration, then all the strings obtained from w by taking any permutation of the strings representing the contents of the regions or exchanging (with their associated contents) two regions present at the same level, still represent the same configuration.

When we say that a configuration $I \in L$, where L is a set of strings, we mean that I is one of the configurations represented by the strings present in L.

Given the system Π we denote by $I(\Pi)$ the set of strings representing *all* the possible configurations of the system (not only the *reachable* ones, i.e., the ones reached during a possible evolution of the system Π).

We consider a *contents time-mapping* e that fixes the time of execution of a rule given a certain configuration.

Formally,

$$e : (R_1 \cup R_2 \cup \cdots \cup R_m) \times I(\Pi) \longrightarrow \mathbb{N}.$$

We suppose the mapping complete, that is, defined for each pair (r, I), $r \in R_1 \cup R_2 \cup \cdots \cup R_m, I \in I(\Pi)$.

Now, joining the system Π and the mapping e, it is possible to construct a *contents-timed P system* $\Pi(e)$ as $(O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0, e_{id})$ working in the following way.

We suppose the existence of an external global clock that ticks at uniform intervals, taken as time unit, starting at time 0.

With each region of the system a finite number of objects and a finite number of evolution rules are associated.

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 started in each region have to be applied.

If a rule $r \in R_i, 1 \leq i \leq m$, 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 as standard in P systems area).

When a rule r is started at step j, and the current configuration of the system is I, then the execution of r terminates at step $j + e_{id}(r, I)$.

If two rules start at the same time, then possible conflicts for using the occurrences of symbol-objects are solved by assigning the objects in a non-deterministic way (again, in the way usually defined in P systems area).

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

The computation halts when no rule can be applied in any region and there are no rules in execution (that is, the system reaches a *halting configuration*).

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 all possible halting computations, we get the set $Ps(\Pi(e))$ of vectors of natural numbers generated by the system $\Pi(e)$.

We say that the time-mapping e is *regular* if it can be calculated by using a *monadic transducer*, that is, a finite state automaton with a label associated to each state, as defined in [2]. As in [2], we will indicate only the time-mapping and not the monadic transducer that implements it. The reader can find more details in [2], but as general idea, the monadic transducer that implements a time-mapping reads, like a finite state automaton, the configuration I of the system followed by the label of the rule r, and produces (outputs) the label associated to the state where it halts. This label represents the time of execution of the rule r, when the system is in the configuration I, that means $e(I, r)$. In what follows we will use only regular time-mappings, even if the idea can be easily extended to more general time-mappings.

Therefore, we use the following notation

$$\mathbb{T}PsP_m(\alpha, reg), \alpha \in \{ncoo\} \cup \{cat_k \mid k \geq 0\},$$

to denote the family of sets of vectors of natural numbers generated by contents-timed P systems with at most m membranes, evolution rules that can be non-cooperative ($ncoo$) or catalytic (cat_k), using at most k catalysts (as usual $*$ is used if the corresponding number of membranes or catalysts/bi-stable catalysts is unknown) and a regular time-mapping.

In [3] a special class of P systems is defined. A P system is called *time-free* if it generates always the same set of vectors, independently of the time-mapping used.

The following Theorem can be found in ([1]).

Theorem 3. *Every P system with non–cooperative rules is time-free.*

In fact, each computation of such a system can be described by a tree, in which the result of a computation is stored in the yield, and such that the length of an application of a rule to an object is represented by the length of the corresponding edge. Thus, it is clear that the result of the computation, obtained on the leaves of the tree, does not depend on the length of the edges in the tree itself. This is also true even in the case when the same rule on the same object could require, in different moment, a different number of steps to be executed.

Therefore, since P systems with symbol-objects and non-cooperative evolution rules generate exactly $PsCF$ (see, e.g., [6]), it follows that

Theorem 4. $\mathbb{T}PsP_m(ncoo, reg) = PsCF,\ m \geq 1.$

4 Time-Driven Computations: An Example

While time does not "help" in case of systems using only non-cooperative evolution rules (Theorem 4), the situation is clearly different when a minimal amount of cooperation is introduced.

In fact, the following example shows that a contents-timed P system using one occurrence of one catalyst and a regular time-mapping can generate the Parikh image of a non-semilinear language.

$\Pi = (O, \mu = [_1\ [_2\ \ ]_2\ ]_1, w_1 = \lambda, w_2 = Acb, R_1, R_2, i_0 = 1)$, where:
$O = \{A, a, A', A'', b, b', b'', b''', c, b'_h, b''_h, b'''_h, \#\}$,
$R_1 = \emptyset$,
$R_2 = \{r_1 : A \to A'A',\ r_2 : A \to a_{out},\ r_3 : A' \to A'',\ r_4 : A'' \to A\}$
$\cup \{r_5 : cb \to cb'b'',\ r_6 : b' \to b''',\ r_7 : cb'' \to c,\ r_8 : cb''' \to cb,\ r_9 : b''' \to \#\}$
$\cup \{r_{10} : cb \to cb'_h b''_h,\ r_{11} : cb''_h \to c,\ r_{12} : b'_h \to b'''_h,\ r_{13} : b'''_h \to \#,\ r_{14} : cb'''_h \to c\}$
$\cup \{r_{15} : \# \to \#\}$.

The time mapping e is defined as follows.
$e(r_j, I) = 1$, for each $r_j, j \in \{1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 13, 14\}$ and any $I \in I(\Pi)$.

$$e(r_7, I) = \begin{cases} 1 \text{ if } & I \in \{[_1\lambda[_2cb'b''w]_2]_1, w \in (A')^+\} \\ 2\ else & \end{cases}$$

$$e(r_{11}, I) = \begin{cases} 1 \text{ if } & I \in \{[_1w[_2cb'_h b''_h]_2]_1, w \in (a)^+\} \\ 2\ else & \end{cases}$$

The contents-timed P system $\Pi(e)$ generates the Parikh image of the language $\{a^{2^n} \mid n \geq 1\}$.

In fact, the computation starts with region 1 empty (it is the output region) and region 2 containing the multiset Acb.

The rule r_1 takes care of the duplication of the objects A. In case the duplication is not done in the correct way, then the entire computation is trashed by using the change in run-time of the execution time of rule r_7, according to the defined time-mapping e. The change in run-time of the execution time of rule r_{11} is used to check that the system halts in a correct way. Notice that all the rules, except r_7 and r_{11} have an execution time always fixed to be one step.

Suppose that in region 2, at certain step $k-1$, there is the multiset A^i, for some i that is a power of two, together with objects c and b. Suppose now that, at step $k-1$, all the occurrences of A are changed by applying the rule r_1 while c and b are rewritten by using the rule r_5. The next configuration obtained (at step k; the rule applied last one step) will contain the multiset $(A')^{i+1}$, and the objects c, b'' and b' in region 2, while region 1 remains empty. Now at step k the rule r_7 is applied, together with the rule r_3 and r_6. Because, at step k, the system contains only A' and c, b'', b', then, for definition of the used time-mapping, the rule r_7 lasts exactly one step. Therefore, at step $k+1$, the system will contain the multiset $(A'')^{i+1}$ and the objects c and b''' in region 2, while region 1 is still empty. In the next step all the objects A'' are changed to A by using the rule r_4 while, at the same time, the objects c and b''' are rewritten to c and b, by using the rule r_8 (both rules take one step to be executed). The initial configuration is obtained again, the number of A has been duplicated, and the process can be iterated.

In case the duplication of objects A has not been done in the correct way, then the entire computation is trashed. In fact, suppose that at step $k-1$ some of the occurrences of A are changed to A' by rule r_1, while some of them are changed to a and sent out to region 1 by rule r_2. Then, in the same step, objects c and b are rewritten to objects c, b'' and b' by rule r_5. At step k (rules r_1, r_2, r_5 take one step to be executed) the system contains some A' and objects c, b'' and b' in region 2 and some a in region 1. Now the rules r_3 and r_7, r_6 are applied, but because of the time-mapping used, and because of the configuration of the system at step k, the rule r_7 takes two steps to be executed. Therefore at step $k+1$, object b''' is produced, but because the catalysts c is still busy by rule r_7, the rule r_9 is used, object # is produced and the computation is non-halting.

The correct halting of the system is guaranteed from the presence of rule r_{10} and rules r_{11}, r_{12}. In fact, suppose that in region 2, at step $k-1$, there is the multiset A^i, for some i that is a power of two, together with objects c and b. To halt the system it is necessary to apply the rule r_{10}. At the same time all the objects A need to be changed in a and sent to region 1, by using rule r_2. In the next step k, the system contains the multiset a^i in region 1, and objects c, b''_h, b'_h in region 2. The computation will halt in two more steps by using rules r_{12} and r_{11} that will last exactly one step and finally rule r_{14}.

If not all the objects A are changed to a in the step $k-1$, then some of them are changed to A' by using rule r_1, and therefore, at step k, the system will contain some a in region 1, some A' in region 2, together with objects c, b''_h and b'_h. At step k, the rule r_{11} and r_{12} are started, but because of the current configuration, rule r_{11} lasts two steps, while r_{12} lasts only one step. Then, objects b'''_h is produced at step $k+1$, while catalyst c is still busy by rule r_{11}. Therefore rule r_{13} is used, object # produced and the computation cannot halt.

This guarantees that the system halts in a correct way and with a number of objects a in the output region that is a power of 2.

5 Time-Driven Universality

It is possible to show even more than in the example presented in Section 4: a P system using a regular time-mapping and only one occurrence of one catalyst is universal, in the sense that it can generate the family of Parikh images of recursively enumerable languages.

As in the previous example, the non-cooperative rules implement the rewriting of the objects, while the catalytic rules check that the rewriting is done in the correct way. If this does not happen, the time of execution of the catalytic rules is set in such a way that the computation does not halt.

The proof is based on the simulation of programmed grammars with appearance checking.

Theorem 5. *$\mathbb{T}PsP(cat_1, reg) = PsRE$.*

Proof. We show that, for every programmed grammar $G = (N, T, S, P)$ with appearance checking is possible to construct a P system Π, using one catalyst, and a regular time-mapping e, such that $Ps(\Pi(e))$ is exactly the Parikh image of $L(G)$. We use the morphism h defined as $h(a) = a', a \in N$, and $h(a) = a_{out}, a \in T$. We suppose that G is in the normal form presented in Theorem 1.

We construct the following P system.

$$\begin{aligned}
\Pi &= (O, \mu = [_1\, [_2\ \ ]_2\,]_1, w_1 = \lambda, w_2 = l_0 S X_1 c X_2 h, R_1, R_2, i_0 = 1), \text{ where:}\\
O &= N \cup T \cup N' \cup T' \cup Lab(P) \cup Lab'(P) \cup \overline{Lab(P)}\\
&\quad \cup \{\#, c, h, h', X_1, X_1', X_2, X_2', X_1'', X_2'', \overline{h'}\},\\
R_1 &= \emptyset,\\
R_2 &= \{r_1 : \overline{l_i} \to \lambda, r_2 : l_i' \to l_i, l_i \to \overline{l_i} \mid l_i \in Lab(P)\}\\
&\quad \cup \{A \to h(u) l_j' \mid (l_i : A \to u, E_{l_i}, F_{l_i}) \in P, l_j \in E_{l_i}\}\\
&\quad \cup \{r_3 : A' \to A \mid A \in N\} \cup \{A \to A \mid A \in N\}\\
&\quad \cup \{h \to h', h \to \overline{h}, r_4 : h' \to h, r_5 : \overline{h} \to h\}\\
&\quad \cup \{X_1 \to X_1', X_1 \to X_1'', cX_1'' \to cX_1, cX_2 \to cX_2', r_6 : cX_2' \to cX_2'', X_2'' \to X_2\}\\
&\quad \cup \{X_1'' \to \#, \# \to \#\},
\end{aligned}$$

where N' and T' are obtained by priming the symbols in N and T, respectively, and $Lab'(P)$ and $\overline{Lab(P)}$ are obtained from $Lab(P)$ by priming and overlining the symbols in $Lab(P)$.

The regular time-mapping e is defined as follows.

Let R' be the set $\{r_1, r_2, r_3, r_4, r_5\}$.

Then $e(r, I) = 2$, for $r \in R'$, and $e(r, I) = 1$ for $r \in R_2 - R'$, $I \in I(\Pi)$.

The time-mapping for r_6 is then fixed in the following way.

$$e(r_6, I) = \begin{cases} 1 & \text{if } I \in \{[_1 w[_2 w'\overline{l_i} l'_j h' X'_1 c X'_2]_2]_1 \\ & \quad \mid l_i, l_j \in Lab(P), l_j \in E_{l_i}, w' \in (N \cup N')^*, w \in T^*\} \\ & \quad \cup\{[_1 w[_2 w'\overline{l_i} l'_j \overline{h} X'_1 c X'_2]_2]_1 \\ & \quad \mid l_i, l_j \in Lab(P), l_j \in F_{l_i}, w' \in ((N \cup N') - A))^*, w \in T^*, \\ & \quad (l_i : A \to u, E_{l_i}, F_{l_i}) \in P\} \\ & \quad \cup\{[_1 w[_2 Z\overline{l_i} l'_h h' X'_1 c X'_2]_2]_1 \mid w \in T^*\} \\ 2 & \text{otherwise.} \end{cases}$$

The idea of the proof is simple: non-cooperative evolution rules of Π simulate the productions of G, the catalytic evolution rules, together with the appropriate regular time-mapping e can be used to check that the grammar productions are applied in the correct order, and this is done very much in the spirit of the example shown in Section 4 (one catalyst is then enough). Notice that some of the rules are labeled, some of them with the same label (when it is not necessary to distinguish them in the time-mapping). The "critical" rule whose time is modified in an appropriate way is r_6: the time of execution is fixed to be 1 if a correct configuration is reached (grammar productions applied correctly), otherwise is fixed to be 2, and this brings to a non-halting computation (# is produced).

The rules $A \to h(u)l'_j$ simulates the productions of G. Each rule, when applied, produces the symbol l'_j that indicates the next production to simulate. The role of r_1 is to guarantee that the productions are applied or skipped in the correct way. Therefore, the time of execution of r_1 is fixed in an appropriate way by e.

In particular:

The configurations of Π represented by the strings in the set

$$\{P[_1 w[_2 w'\overline{l_i} l'_j h' X'_1 c X'_2]_2]_1 \mid l_i, l_j \in Lab(P), l_j \in E_{l_i}, w' \in (N \cup N')^*, w \in T^*\}$$

correspond to all (and only) the configurations obtained when a production of G (the one with label l_i) has been correctly simulated (l_j is the label of the next production to apply).

The configurations represented by the strings in the set

$$\{[_1 w[_2 w'\overline{l_i} l'_j \overline{h} X'_1 c X'_2]_2]_1 \mid l_i, l_j \in Lab(P), l_j \in F_{l_i}, w' \in ((N \cup N') - A))^*, w \in T^*,$$
$$(l_i : A \to u, E_{l_i}, F_{l_i}) \in P\}$$

correspond to all (and only) the configurations obtained when the production of G with label l_i has been correctly skipped (appearance checking) and the next production to apply is the one with label l_j.

Finally the strings in the set

$$\{[_1 w[_2 Z\overline{l_i} l'_h h' X'_1 c X'_2]_2]_1 \mid w \in T^*\}$$

correspond to the configurations where the next production to simulate is the one with label l_h (that, recalling the normal form of G, there should not be nonterminals).

If the configuration where r_1 is started is not among those presented, then the time of execution of r_1 is set to be 2 and this brings Π to produce # (by using the rule $X_1'' \rightarrow \#$). Therefore, the system Π can simulate all and only the correct derivations of G; from this the result follows.

6 Conclusions

We have proposed a model of computation based on membrane systems, called contents-timed P systems, where rewriting rules have different times of execution which can vary during the whole computation, depending on the configuration of the system. This is motivated by the fact that the time required by different reactions in a biochemical process may be different and, moreover, the time required by the same reaction can change dynamically according to the state of the system where that reaction takes place.

We have shown that such systems are universal in a very simple framework: a regular time-mapping suffices to obtain universality for systems with a single catalyst.

An interesting aspect which remains to investigate concerns systems with the time mapping of more restricted forms.

For instance, we could consider systems in which the time required by the various rewriting rules depends only upon specific regions or only upon the regions which are close (i.e., immediately outside or immediately inside) the region where the rule is applied.

Another possibility is to consider time mapping only depending upon some specific objects, representing specific conditions in which the systems is (e.g., an object representing the temperature).

Another fruitful research direction could be to compare this kind of systems with timed Petri nets, to point out similarities and differences with those systems.

Finally, we think that the application of the same idea of time mapping to some variants of P systems, such as symport/antiport P systems, could be useful to clarify the role of dynamically changing execution time with respect to various aspects of P systems.

Acknowledgements

The authors would like to thank Artiom Alhazov for useful discussions.

The work of Claudio Zandron has been partially supported by the Italian Ministry of University (MIUR), under project PRIN-04 "Systems Biology: modellazione, linguaggi e analisi (SYBILLA)".

References

1. M. Cavaliere, V. Deufemia: Further results on time-sndependent P Systems. *Int. Journal Foundation Computer Science*, 17, 1 (2006), 69–89.

2. M. Cavaliere, P. Leupold: Evolution and observation: A non-standard way to accept formal languages. In *Machines, Computations, and Universality*, Saint Petersburg, Russia, 2004, (M. Margenstern, ed.), LNCS 3354, Springer-Verlag, 2005, 153–163.
3. M. Cavaliere, D. Sburlan: Time-independent P systems. In *Membrane Computing. International Workshop* WMC 2004, Milan, Italy, 2004, Revised Selected and Invited Papers, (G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds.) LNCS 3365, Springer-Verlag, 2005, 239–258.
4. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.
5. M.L. Minsky: *Computation. Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, 1967.
6. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin (Natural Computing Series), 2002.
7. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1997.
8. J.E. Hopcroft, J.D. Ullman: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.
9. The P systems web page: `http://psystems.disco.unimib.it`