

Inhibiting/De-Inhibiting Rules in P Systems

Matteo CAVALIERE

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

Mihai IONESCU and Tseren-Onolt ISHDORJ

Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: mi@fll.urv.es,
tserenonolt.ishdorj@estudiants.urv.es

Abstract

In this paper we describe and formally specify a mechanism, inspired from chemical reactions in the cell biology, which controls computations in general P systems using inhibiting/de-inhibiting rules. We also investigate the computational universality of such rules in both generative and accepting P systems. In each case, the computational universality is obtained by using one single membrane.

1 Introduction

P systems represent a class of distributed/parallel computing devices whose functioning is inspired from the behavior of living cells. There, chemical compounds are processed in a massive parallel manner inside a compartmental structure of membranes that control the substances exchanges between regions they delimit. The reactions that take place inside such a biological structure can be formally described by cooperative rules.

In cell biology it is known that many reactions in the cell are catalyzed by the presence of associated enzyme. On the other hand, in bacteria, the enzymes (protein) can be activated/inactivated during the cellular process (in other words, an inactivated enzyme is not able to catalyze the corresponding reaction).

One particular case is that of promoted/inhibited reactions that happen in the presence/absence of certain chemicals which are not directly implied in reactions. P systems with promoters/inhibitors at the level of rules were investigated in [4] and [7]. In the case of promoters, the rules (reactions) are possible only in the presence of certain symbols. 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 . An object b is an *inhibitor* for a rule $u \rightarrow v$, and we denote this by $u \rightarrow v|_{-b}$, if the rule is active only if inhibitor b is not present in the region. In particular, promoters/inhibitors themselves can evolve according to some rules.

Another possibility is to consider biological signals processed by cells. Signals can arise from inside the cell or from the external environment and the correct answer to

certain signals is essential for bacteria to survive in a certain environment. In bacteria, the enzyme activation/inactivation, as well as other important process, are controlled by covalent modification of proteins. We refer for biological motivations to the signals-based P systems introduced in [2], which use symbol-objects (chemical substances) and chemical reactions (evolution rules) present in the regions. The symbol-objects evolve using the evolution rules but they cannot be moved, while the rules present in the regions can be activated/deactivated using signals called signal-promoters. The signal-promoters do not participate in any evolution rule but they are able to move around the regions of the system. In other words, the SB P systems are similar to the classical P systems using promoters but with two restrictions: the symbol-objects cannot be moved and signal-promoters cannot be created but only moved across the membranes. Signaling rules are simple evolution rules, promoted by some signal-promoter p . When a signaling rule $a \rightarrow v|_{p_{tar}}$ is applied, the object a is transformed into the objects specified by v and the promoter p is sent into the (adjacent) region specified by tar .

We also mention the related concept of P systems with creation of evolution rules during the computation. A set R_{pos} of all *possible rules*, injectively labeled by the elements of a set lab is given; the rules associated with a membrane i will be of the form $r : u \rightarrow v/z$, for $r \in lab, z \in lab^*$. The meaning of $r : u \rightarrow v/z$ is that after using this rule, a copy of r is consumed, and all rules indicated by z become available in membrane i . Consuming/creation of rules can be seen as a way of controlling the next rule to be used, as in programmed grammars. More details are present in [3].

In this paper we introduce the concept of *inhibiting/de-inhibiting rules* in P systems. An inhibited rules is formally written as $r : \neg(u \rightarrow v)$, and the meaning is that the rule cannot be applied, more precisely u cannot evolve to v . An evolution rule can de-inhibit an inhibited rule allowing it to apply. To this aim, we also consider rules of the form $r_i : (u \rightarrow v)\neg(r_1) \cdots \neg(r_k)$, which says that u evolves into v and the rules r_1, \dots, r_k are inhibited or de-inhibited according to the previous state of each rule. Here rules $r_j, 1 \leq j \leq k$, are any kind of developmental rules. Evolution rule itself can be a deletion rule, and then we denote it by $r_i : u \rightarrow \lambda\neg(r_1) \cdots \neg(r_k)$.

The proposing mechanism can be more general and powerful than the standard activating/deactivating computations in P systems such as promoter/inhibitor, creation rules during the computation as well as signal based systems. In other words, the mentioned classes could be simulated by inhibiting/de-inhibiting rules in P systems.

We introduce the formal definition of P Systems with inhibiting/de-inhibiting rules and we explore the computational power of the systems with context-free and inhibiting/de-inhibiting rules. Both generative and accepting cases will be studied here.

2 Preliminaries

2.1 P Systems Prerequisites

A P system (of degree $m \geq 1$) with symbol-objects and multiset rewriting rules is a construct

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

where:

- O is the alphabet of Π ; its elements are called *objects*;
- $C \subseteq O$ is the set of catalysts;

- μ 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 regions i at the beginning of a computation;
- R_i , $1 \leq i \leq m$, are finite sets of evolution rules over O associated with the regions $1, 2, \dots, m$ of μ ; these evolution rules are of the form $a \rightarrow v$ or $ca \rightarrow cv$, where a is an object from $O - C$ and v is a string over

$$(O - C) \times (\{here, out, in\})$$

(in general, the target indications *here*, *out*, *in* are written as subscripts of objects from O);

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

A P system with the mentioned features starts to evolve from an *initial configuration*, by performing all operations in a parallel way, for all applicable rules, for all occurrences of objects in the region associated with the rules, for all regions at the same time, and according to a universal clock.

In this way, one gets *transitions* between the configurations of the system. A sequence of transitions is called a *computation*. A computation is *halting*, if no rule is applicable in any region (nothing can happen anymore). A computation is *halting* if it reaches a halting configuration. We consider only halting computations. The *result* of a (halting) computation is the *number* of objects present in the region i_0 in the halting configuration. The set of all numbers constructed in this way by a system Π is denoted by $N(\Pi)$. For such kind of P systems we will use the notation

$$NOP_m(\alpha), \alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 0\}.$$

to denote the family of sets of natural numbers generated by P systems with at most m membranes, evolution rules that can be non-cooperative (*ncoo*), cooperative (*coo*), or catalytic (*cat_k*), using at most k catalysts.

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 $Ps(\Pi)$.

For the basics of formal language theory we refer the reader to [11]. In short, O^* denote the set of all strings over the alphabet O . For $a \in O$ and $x \in O^*$ we denote by $|x|_a$ the number of occurrences of a in x . Then, for $O = \{a_1, \dots, a_n\}$, the *Parikh mapping* associated with O is the mapping on O^* defined by $\Psi_{O(x)} = (|x|_{a_1}, \dots, |x|_{a_n})$ for each $x \in O^*$. The family of recursively enumerable languages is denoted by *RE*, and the 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 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. Clearly, using strings is only one of many ways to specify multisets.

2.2 Matrix Grammars

We recall here the notion of matrix grammar because we will use in the paper the characterization of recursively enumerable languages by means of *matrix grammar with appearance checking*.

Such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$ in M and the strings $w_i \in (N \cup T)^*$, $1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i , and the rule $A_i \rightarrow x_i$ appears in F . (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied – therefore we say that these rules are applied in the *appearance checking* mode.) If the set F is empty, then the grammar is said to be without appearance checking.

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac} . It is known that $CF \subset MAT \subset MAT_{ac} = RE, NREG = NCF = NMAT \subset NCS$ (for instance, the one-letter languages in MAT are known to be regular.).

A matrix grammar $G = (N, T, S, M, F)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in M are in one of the following forms:

1. $(S \rightarrow XA)$, with $X \in N_1, A \in N_2$,
2. $(X \rightarrow Y, A \rightarrow x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$,
3. $(X \rightarrow Y, A \rightarrow \#)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \rightarrow \lambda, A \rightarrow x)$, with $X \in N_1, A \in N_2$, and $x \in T^*, |x| \leq 2$.

Moreover, there is only one matrix of type 1 (that is why one uses to write it in the form $(S \rightarrow X_{init}A_{init})$, in order to fix the symbols X, A present in it), and F consists exactly of all rules $A \rightarrow \#$ appearing in matrices of type 3; $\#$ is a trap-symbol, because once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of a derivation.

For each matrix grammar there is an equivalent matrix grammar in the binary normal form. Details can be found in [6].

2.3 Register Machines

We will also use in our paper the register machines [8], that is why we recall here this notion. Such a machine runs a program consisting of numbered instructions of several simple types. Several variants of register machines were shown to be computationally universal.

A *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 , and j, k, l are labels from the set $Lab(M)$ (which numbers the instructions in a one-to-one manner),
- i is the initial label, and
- h is the final label.

The machine is capable of the following instructions:

$(add(r), k, l)$: Add one to the contents of register r and proceed to instruction k or to instruction l ; in the deterministic variants usually considered in the literature we demand $k = l$.

$(sub(r), k, l)$: If register r is not empty, then subtract one from its contents and go to instruction k , otherwise proceed to instruction l .

$halt$: This instruction stops the machine. This additional instruction can only be assigned to the final label h .

A deterministic m -register machine can analyze an input $(n_1, \dots, n_\alpha) \in \mathbf{N}_0^\alpha$ in registers 1 to α , which is recognized if the register machine finally stops by the halt instruction with all its registers being empty (this last requirement is not necessary). If the machine does not halt then the analysis was not successful.

3 Inhibiting/De-Inhibiting Rules in P Systems

A P system *with inhibiting/de-inhibiting rules* (in short, an ID P system), of degree $m \geq 1$, is a construct

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

where:

- $m \geq 1$ is the degree of the system;
- O is the alphabet of *objects*;
- $C \subseteq O$ is the set of catalysts;
- To each rule in $R = R_1 \cup R_2 \dots \cup R_m$ is associated an unique label. The set of all labels is $H = \{r_1, \dots, r_k\}$. We denote $\neg H = \{\neg r_i \mid r_i \in H\}$;
- μ is a *membrane structure*, consisting of m membranes, labeled $1, 2, \dots, m$;
- w_1, \dots, w_m are strings over O , describing the *multisets of objects* placed in the m regions of μ ;
- R_i is a finite set of *developmental rules*, associated with region i . The rules in R_i are of the forms: $r_i : \neg(a \rightarrow w)S$ or $r_i : (a \rightarrow w)S$ where $r_i \in H, a \in O, w \in (O \times TAR)^*, S \subseteq \neg H$;
- i_0 is the output region.

Note that we consider here only non-cooperative rules.

A configuration of an ID P system is described by using the m -tuple of multisets of objects, present in the m regions of the system. To each region a finite number of objects is associated together with a finite number of rules. The m -tuple (w_1, w_2, \dots, w_m) is the initial configuration of the system. Some of the rules are initially inhibited (if the symbol \neg is written immediately before the rule). A transition between two configurations is governed by the application in a non-deterministic and maximally parallel way of the rules that are not inhibited.

When a rule $r_i : (a \rightarrow w)S$ is applied the object a is rewritten with the objects in w (as in standard P systems) and the rules from S are de-inhibited (if they were inhibited) or inhibited (if they were de-inhibited).

A sequence of configurations is called a *computation*. The system continues in parallel steps until there remain no applicable rules in any region of the system. Then the system halts (the computation is successful) and we consider the number of objects contained in the output region i_0 as the result of the computation.

We use the notation

$$PsID_m(\alpha), \alpha \in \{ncoo\} \cup \{cat_k \mid k \geq 0\},$$

to denote the family of sets of vectors of natural numbers generated by ID 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, $*$ indicates that the corresponding number is not bounded).

4 An Example

We now illustrate the definitions from the previous section with an example.

Let us construct an ID P system of degree 1,

$$\Pi = (Aa, \emptyset, \{r_1, r_2, r_3\}, []_1, A, R, 1),$$

where:

- $R = \{r_1 : A \rightarrow AA, r_2 : A \rightarrow AA\neg(r_1)\neg(r_2)\neg(r_3), r_3 : \neg(A \rightarrow a)\},$

When the computation starts in the initial configuration with the object A in the region, the rules r_1 or r_2 can be applied but the rule r_3 cannot be applied since it is inhibited. We use the rule r_1 $m-1$ times and then we apply the rule r_2 . Then we produce 2^m copies object A , simultaneously the rules r_1 and r_2 are inhibited (so they cannot be used any more), and the rule r_3 is de-inhibited. We have used context-free and inhibiting/de-inhibiting rules to obtain a^{2^m} , hence

$$PsIDP_1(ncoo) = \{2^m \mid m \geq 1\}$$

which is not in $PsCF$.

5 Universality Results

5.1 The Generative Case

Here, we present an universality result of P systems with inhibiting/de-inhibiting rules. The proof is based on the simulations of matrix grammar with appearance checking.

Theorem 5.1 $PsIDP_1(cat_1) = PsRE$.

Proof. Consider a matrix grammar $G = (N, T, S, M, F)$ with appearance checking, in the binary normal form, hence with $N = N_1 \cup N_2 \cup \{S, \#\}$ and with the matrices introduced in the section 2.2 (the four forms). Assume that all matrices are labeled in an injective manner with $m_i, 1 \leq i \leq n$, and each terminal matrix $(X \rightarrow \lambda, A \rightarrow x)$ is replaced by $(X \rightarrow f, A \rightarrow x)$, where f is a new symbol.

We construct the P system of degree 1,

$$\begin{aligned} \Pi &= (O, C, H, []_1, w_1, R, i_0), \text{ where:} \\ H &= \{r_i \mid 1 \leq i \leq 10\}, \\ O &= N_1 \cup T \cup N_2 \cup \{p, d, f, \#\}, \\ C &= \{c\}, \\ w_1 &= pX_{init}A_{init}, \\ i_0 &= 0. \end{aligned}$$

and the set R containing the following rules:

- The simulation of a matrix $m_i : (X_i \rightarrow Y_i, A_i \rightarrow x_i)$, with $X_i \in N_1, Y_i \in N_1 \cup \{f\}, A_i \in N_2, x_i \in (N_2 \cup T)^*, |x_i| \leq 2, 1 \leq i \leq j$, is done in 3 steps using the next rules:

$$\begin{aligned} r_1 &: p \rightarrow \neg(r_2)\neg(r_3), \\ r_2 &: \neg(cX_i \rightarrow cY_i), \\ r_3 &: \neg(cA_i \rightarrow cx_id), \\ r_4 &: d \rightarrow p. \end{aligned}$$

The execution of rule r_1 de-inhibits rules r_2 and r_3 . In the next step, rules r_2 and r_3 are applied freely. We have used catalyst c in order to inhibit the parallelism. One can notice that rule r_3 introduces an object d which is used in rule r_4 (the last step) to iterate the process.

- The simulation of a matrix $m_i : (X_i \rightarrow Y_i, A_i \rightarrow \#)$, with $X_i, Y_i \in N_1$ and $A_i \in N_2, j + 1 \leq i \leq n$, is done also in 3 steps, using the next rules:

$$\begin{aligned} r_5 &: p \rightarrow \neg(r_6)\neg(r_7), \\ r_6 &: \neg(cX_i \rightarrow cY_id), \\ r_7 &: \neg(A_i \rightarrow \#), \\ r_8 &: d \rightarrow p. \end{aligned}$$

The purpose of rule r_5 is to de-inhibit the inhibited rules r_6 and r_7 . Then rules r_6 and r_7 can perform freely in the next step of the computation. If any copy of an object A_i is present, then it introduces the trap-object $\#$ and the computation never stops. If no A_i is present, then object d , which was introduced by rule r_6 , is rewritten in the auxiliary object p , to iterate the procedure.

R also contains the following rules:

$$\begin{aligned} r_9 &: a \rightarrow (a, out), \\ r_{10} &: \# \rightarrow \#. \end{aligned}$$

The equality $\Psi_T(L(G)) = PsIDP_1(\Pi)$ easily follows from the above explanations. \square

5.2 The Accepting Case

The following theorem illustrates the computational universality (in its accepting variant) of P systems with object rewriting non-cooperative rules and inhibiting/de-inhibiting rules. The systems we propose simulates the moves of deterministic register machines. Moreover, the P systems obtained are also deterministic.

Theorem 5.2 $PsIDP_1(cat_1) = PsRE$.

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 r is represented by the multiplicity of the object a_r .

Formally, we define the P system of degree 1,

$$\begin{aligned}
 \Pi &= (O, C, H, []_1, w_1, R, i_0), \text{ where:} \\
 O &= \{a_r, S_r, S'_r, S''_r, S'''_r \mid 1 \leq r \leq n\} \cup \{F\} \\
 &\quad \cup \{l_k, e \mid l_k : (add(r), l_i, l_i) \in P\} \\
 &\quad \cup \{l_k, e, e' \mid l_k : (sub(r), l_i, l_j) \in P\}, \\
 C &= \{c\}, \\
 H &= \{r_i \mid 1 \leq i \leq 13\}, \\
 w_1 &= \{e, a_r^{k_r}, 1 \leq r \leq n, k_r \in N\}, \\
 i_0 &= 0.
 \end{aligned}$$

and R is defined as follows:

- for each instruction $l_k : (add(r), l_i, l_i) \in P$, we add to R the rules:

$$\begin{aligned}
 r_1 &: l_k \rightarrow S_r, \\
 r_2 &: S_r \rightarrow a_r e, \\
 r_3 &: e \rightarrow l_i,
 \end{aligned}$$
- for each instruction $l_k : (sub(r), l_i, l_j) \in P$, we add to R the rules:

$$\begin{aligned}
 r_4 &: l_k \rightarrow e S_r, \\
 r_5 &: e \rightarrow e', \\
 r_6 &: S_r \rightarrow S'_r \neg(r_7), \\
 r_7 &: \neg(ca_r \rightarrow cF), \\
 r_8 &: S'_r \rightarrow S''_r, \\
 r_9 &: F \rightarrow \neg(r_7) \neg(r_{12}), \\
 r_{10} &: S''_r \rightarrow S'''_r, \\
 r_{11} &: S'''_r \rightarrow \neg(r_{13}), \\
 r_{12} &: \neg(e' \rightarrow l_i), \\
 r_{13} &: \neg(e' \rightarrow l_j).
 \end{aligned}$$

The system works as following. Initially the P system starts the computation having in its input region the objects $a_1^{k_1}, \dots, a_n^{k_n}$ and the label i of the first instruction of the register machine we want to simulate. The vector (k_1, \dots, k_n) represents the vector that has to be accepted by our P system.

The P system starts the computation by simulating the first instruction of the register machine program. Let us suppose that the current instruction to be executed is of type

$(l_k : add(r), l_i, l_i) \in P$. Then, the rule $l_k \rightarrow S_r$ is executed. The object S_r indicates that the number of objects a_r has to be incremented. This will be realized, in the second computational step, by the evolution rule $S_r \rightarrow a_r e$. Next rule r_3 is executed, rule r_2 inhibited, and the operation allows our P system to further simulate the instruction of the register machine indicated by label l_i .

If a subtraction instruction $(l_k : sub(r), l_i, l_j) \in P$ is simulated then the rule $l_k \rightarrow e S_r$ is executed. The object S_r is used to de-inhibit rule r_7 , and to produce object S'_r in rule r_6 , while the object e evolves into e' . In the next step, if the number of objects a_r in register r is greater than 0, then the execution of the de-inhibited rule $ca_r \rightarrow cF$ decreases the number of objects a_r by 1, and produces object F for the next step. We have used the catalyst c in order to inhibit the parallelism (because more copies of a_r might be present in the membrane). Meanwhile, the object S'_r evolves into S''_r as showed in rule r_8 . The object F used in rule r_9 to inhibit the de-inhibited rule $\neg r_7$ guarantees that only one object a_r was deleted, and the inhibited rule can not apply any further at this stage. The rule r_{12} is also de-inhibited by the execution of the rule r_9 . In this way we generate the label l_i of the next register machine instruction. Otherwise (i.e., if there is no object a_r in region) the rule $a_r \rightarrow F$ is not executed, therefore the object F is not produced and rule r_9 cannot be applied. On the other hand, object S''_r evolves into S'''_r by the execution of rule r_{10} and at the next step S'''_r de-inhibits rule r_{13} , so e' evolves to label l_j . The object e' still appears, because rule r_{12} did not apply in the previous steps, so rule r_{13} can generate label l_j of the next register machine instruction.

The simulations of the instructions which were presented above, $l_k(add(r), l_i, l_i) \in P$ and $l_k(sub(r), l_i, l_j) \in P$, are iterated according to the register machine program. The simulation stops when label h (which stands for *halt* instruction) is generated and no other objects a_r , $1 \leq r \leq n$, are in region 1.

□

6 Conclusion

In this paper, we have considered a mechanism of chemical reactions in the cell biology, which is used to control the computation steps in P systems by inhibiting/de-inhibiting their general rules. We have inhibited/de-inhibited only the evolution rules. In general, all types of rules can be inhibited/de-inhibited in P systems. Moreover, the mechanism could be used in neural-like P systems to excite/inhibit impulses transmitted through the neural cell.

The following problems are expecting future work: direct simulations of P systems with promoters/inhibitors, signals, creation of rules during a computation. P systems with active membranes could also use this mechanism.

Acknowledgments. The authors acknowledge IST-2001-32008 project “MolCoNet”. The first and second authors were supported by the FPU fellowship from the Spanish Ministry of Education, Culture and Sport. The third author acknowledges The State Training Fund of the Ministry of Science, Technology, Education and Culture of Mongolia.

References

- [1] Alhazov, A., Ishdorj, T.-O., Membrane Operations in P Systems with Active Membranes, *Technical Report 01/2004*, University of Seville, Second Brainstorming Week on Membrane Computing, Sevilla, 2004, pp. 37-44.
- [2] Ardelean, I.I., Cavaliere, M., Sburlan, D., Computing Using Signals: From Cells to P Systems, *Technical Report 01/2004*, University of Seville, Second Brainstorming Week on Membrane Computing, Sevilla, 2004, pp. 60-73.
- [3] Arroyo, F., Baranda, A.V., Castellanos, J., Păun, Gh., Membrane Computing: The Power of (Rule) Creation, *Journal of Universal Computer Science*, 8, 3 (2002), 369-381.
- [4] Bottoni, P., Martín-Vide, C., Păun, Gh., Rozenberg, G., Membrane Systems with Promoters/Inhibitors, *Acta Informatica*, 38, 10 (2002), 695-720.
- [5] Calude, C.S., Păun, Gh., Rozenberg, G., Salomaa, A., *Multiset Processing*, LNCS 2235, Springer-Verlag, Berlin, 2001.
- [6] Dassow, J., Păun, Gh., *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
- [7] Ionescu, M., Sburlan, D., On P Systems with Promoters/Inhibitors, *Technical Report 01/2004*, University of Seville, Second Brainstorming Week on Membrane Computing, Sevilla, 2004, pp. 264-280.
- [8] Minsky, M.L., *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, 1967.
- [9] Păun, Gh., *Membrane Computing: An introduction*, Springer-Verlag, Berlin, 2002.
- [10] Păun, Gh., Rozenberg G., A Guide to Membrane Computing, *Theoretical Computer Science*, 287, 1 (2002), 73-100.
- [11] Rozenberg, G., Salomaa, A.(eds.): *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.