数理解析研究所講究録
1166 巻 2000 年 161-169

String-Objects in P Systems

Carlos MARTÍN-VIDE
Research group in Mathematical Linguistics and Language Engineering
Rovira i Virgili University, Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: cmv@astor.urv.es

Gheorghe PĂUN[1]
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
E-mail: gpaun@imar.ro

Abstract. We first survey some results concerning the generative power of P systems with objects described by strings processed by rewriting and by splicing, then we consider P systems with multisets of string-objects processed by replication, splitting, point mutation, and crossover. A combination of these variants (systems with multisets of string-objects processed by rewriting and crossing-over, without using priorities) is shown to characterize the recursively enumerable languages (moreover, systems with five membranes suffice).

1 Introduction

P systems are distributed parallel computing models which start from the observation that the processes which take place in the complex structure of a living cell can be (and it has been) interpreted as a computation. The basic ingredients are a *membrane structure*, consisting of several membranes embedded in a main membrane (called the *skin*) and delimiting *regions* where multisets of certain *objects* are placed (Figure 1 illustrates these notions); the objects evolve according to given *evolution rules*, which are applied nondeterministically (the rules to be used and the objects to evolve are randomly chosen) in a maximally parallel manner (in each step, all objects which can evolve must do it). The objects can also be communicated from a region to another one. In this way, we get *transitions* from a *configuration* of the system to the next one. A sequence of transitions constitutes a *computation*; with each *halting computation* we associate a *result*, the number of objects in a specified *output membrane*.

Since these computing devices were introduced ([6]) several variants were considered, [11], [8], [9], [1], etc. Many of them were proved to be computationally complete, able to compute all recursively enumerable sets of natural numbers. When membrane division is allowed, NP-complete problems are shown to be solved in linear time: see [9] for SAT, [3] for the Hamiltonian Path Problem and the Node Covering Problem (and [12] for five NP-complete problems from logic and five from graph theory, solved in a quite uniform way by using P systems with cooperative rules; by such rules, several objects may evolve together).

In most of these variants, the objects are described by symbols from a given alphabet. It is also possible (this was considered already in [6]) to work with objects described by

[1]Research supported by the Direcció General de Recerca, Generalitat de Catalunya (PIV).

strings. The evolution rules should then be string processing rules, such as rewriting and splicing rules. As a result of a computation we can either consider the language of all strings computed by a system, or the number of strings produced by a system and "stored" in a specified membrane. In the first case one works with usual sets of strings (languages), not with multisets (each string is supposed to appear in arbitrarily many copies), while in the latter case we have to work with multisets. In such a framework we have to consider operations on strings which can increase and decrease the number of strings, rewriting and splicing are not sufficient. Such operations are the replication and the splitting of strings – see precise definitions in Section 4.

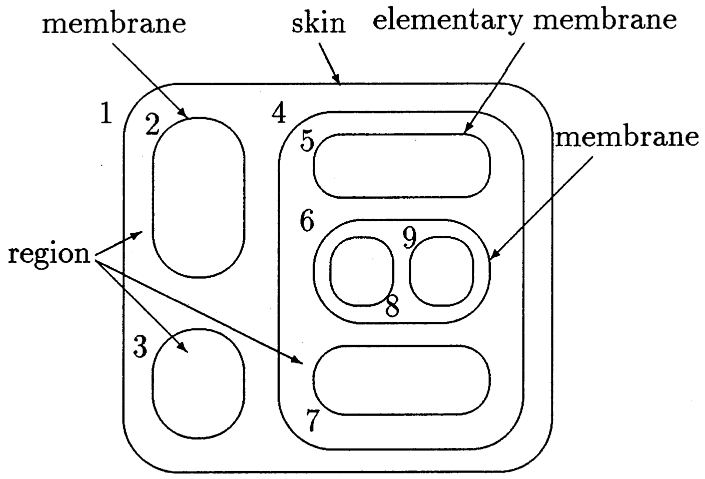

Figure 1: A membrane structure

In all these cases, characterizations of recursively enumerable languages or of recursively enumerable sets of natural numbers can be found. We here recall results of this type from [6], [5], [3], [13], [1], [4].

In the case of rewriting, the characterization of recursively enumerable languages is obtained by using a priority relation among rewriting rules, which is a very powerful feature. We show here that we can avoid using a priority relation, providing that we compensate the loss in power by adding crossovering rules to rewriting rules and that we take into account the number of copies of each string (that is, we work with multisets). Systems with only five membranes, arranged in a linear structure (the tree describing the membrane structure is a line), can generate all recursively enumerable languages.

2 Rewriting P Systems

If the objects in a P system are described by strings, then their evolution will correspond to a string transformation. In this section we consider transformations in the form of rewriting steps, as usual in formal language theory, but we associate with them *target* indications, telling us the region where the result of the rewriting should be placed after applying a rule.

We refer to [14] for all elements of formal language theory we use. We only mention that we denote by *RE* the family of recursively enumerable languages.

Always we use only context-free rules, that is, the rules of our systems are of the form $(X \to v; tar)$, where $X \to v$ is a context-free rule and $tar \in \{here,\ out,\ in_j\}$, with the obvious meaning: the string produced by using this rule will go to the membrane indicated by tar (j is the label of a membrane).

Formally, a *rewriting P system* is a construct

$$\Pi = (V, T, \mu, L_1, \dots, L_m, (R_1, \rho_1), \dots, (R_m, \rho_m), i_0),$$

where V is an alphabet, $T \subseteq V$ (the terminal alphabet), μ is a membrane structure with m membranes labeled with $1, 2, \dots, m$, $L_1, \dots, L_m$ are finite languages over V (initial strings placed in the regions of μ), $R_1, \dots, R_m$ are finite sets of context-free evolution rules, $\rho_1, \dots, \rho_m$ are partial order relations over $R_1, \dots, R_m$, and i_0 is the output membrane.

The language generated Π is denoted by $L(\Pi)$ and it is defined as follows: we start from the initial configuration of the system and proceed iteratively, by transition steps performed by using the rules in parallel, to all strings which can be rewritten, obeying the priority relations; when the computation halts, we collect the terminal strings generated in the output membrane. Note that each string is processed by one rule only, the parallelism refers here to processing simultaneously all available strings by all applicable rules.

We denote by $RP_m(Pri)$ the family of languages generated by rewriting P systems of degree at most $m, m \geq 1$, using priorities; when priorities are not used, we replace Pri with $nPri$.

In order to illustrate the way of working of a rewriting P system, we consider an example (which also proves that the family $RP_2(nPri)$ contains non-context-free languages):

$$\begin{aligned}
\Pi &= (\{A, B, a, b, c\}, \{a, b, c\}, [_1[_2\ ]_2]_1, \emptyset, \{AB\}, (R_1, \emptyset), (R_2, \emptyset), 2),\\
R_1 &= \{(B \to cB; in_2)\},\\
R_2 &= \{(A \to aAb; out),\ (A \to ab; here),\ (B \to c; here)\}.
\end{aligned}$$

It is easy to see that $L(\Pi) = \{a^n b^n c^n \mid n \geq 1\}$ (if a string $a^i A b^i c^i B$ is rewritten in membrane 2 to $a^i A b^i c^{i+1}$ and then to $a^{i+1} A b^{i+1} c^{i+1}$ and sent out, then it will never come back again in membrane 2, the computation stops, but the output membrane will remain empty).

For $RP_3(Pri)$, the following result is proved in [6]; the improvement to $RP_2(Pri)$ was given independently in [3] and [5].

Theorem 1. $RP_2(Pri) = RE$.

3 Splicing P Systems

The strings in a P system can also be processed by using the *splicing* operation introduced in [2] as a formal model of the DNA recombination under the influence of restriction enzymes and ligases (see a comprehensive investigation of splicing systems in [10]).

Consider an alphabet V and two symbols $\#, \$$ not in V. A *splicing rule* over V is a string $r = u_1\#u_2\$u_3\#u_4$, where $u_1, u_2, u_3, u_4 \in V^*$ (V^* is the set of all strings over V). For such a rule r and for $x, y, w, z \in V^*$ we define

$$\begin{aligned}
(x, y) \vdash_r (w, z) \ \text{ iff } \ & x = x_1 u_1 u_2 x_2,\ y = y_1 u_3 u_4 y_2,\ w = x_1 u_1 u_4 y_2,\ z = y_1 u_3 u_2 x_2,\\
& \text{for some } x_1, x_2, y_1, y_2 \in V^*.
\end{aligned}$$

(One cuts the strings x, y in between u_1, u_2 and u_3, u_4, respectively, and one recombines the fragments obtained in this way.)

A *splicing P system* (of degree $m, m \geq 1$) is a construct

$$\Pi = (V, T, \mu, L_1, \ldots, L_m, R_1, \ldots, R_m),$$

where V is an alphabet, $T \subseteq V$ (the *output* alphabet), μ is a membrane structure consisting of m membranes (labeled with $1, 2, \ldots, m$), $L_i, 1 \leq i \leq m$, are languages over V associated with the regions $1, 2, \ldots, m$ of μ, $R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* associated with the regions $1, 2, \ldots, m$ of μ, given in the following form: $(r; tar_1, tar_2)$, where $r = u_1\#u_2\$u_3\#u_4$ is a usual splicing rule over V and $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$.

Note that, as usual in splicing systems, when a string is present in a region of our system, it is assumed to appear in arbitrarily many copies.

A transition in Π is defined by applying the splicing rules from each region of μ, in parallel, to all possible strings from the corresponding regions, and following the target indications associated with the rules. More specifically, if x, y are strings in region i and $(r = u_1\#u_2\$u_3\#u_4; tar_1, tar_2) \in R_i$ such that we can have $(x, y) \vdash_r (w, z)$, then w and z will go to the regions indicated by tar_1, tar_2, respectively. Note that after splicing, the strings x, y are still available in region i, because we have supposed that they appear in arbitrarily many copies (an arbitrarily large number of them were spliced, arbitrarily many remain), but if a string w, z, resulting from a splicing, is sent out of region i, then no copy of it remains here.

The result of a computation consists of all strings over T which are sent out of the system at any time during the computation. We denote by $L(\Pi)$ the language of all strings of this type. We say that $L(\Pi)$ is *generated* by Π. Note that in this section we do not consider halting computations, but we leave the process to continue forever and we just observe it from outside and collect the terminal strings leaving the system.

We denote by $SP(tar, m, p)$ the family of languages $L(\Pi)$ generated by splicing P systems as above, of degree at most $m, m \geq 1$, and of depth at most $p, p \geq 1$.

If all target indications tar_1, tar_2 in the evolution rules of a P system are of the form *here, out, in*, then we say that Π is of the *i/o type*; the strings produced by splicing and having associated the indication *in* are moved into any lower region immediately below the region where the rule is used (that is, the target membrane is nondeterministically chosen from the adjacently lower membraes). The family of languages generated by P systems with this weaker target indication and of degree at most m and depth at most p is denoted by $SP(i/o, m, p)$.

Proofs of the following results, showing the computational universality of splicing P systems of rather simple forms, can be found in [13].

Theorem 2. $SP(i/o, 3, 3) = SP(tar, 3, 2) = SP(i/o, 5, 2) = RE$.

4 P Systems with Worm-Objects

In P systems with symbol-objects we work with multisets and the result of a computation is a natural number or a vector of natural numbers; in the case of string-object P systems we work with sets of strings and the result of a computation is a string. We can combine the two ideas: we can work with multisets of strings and consider as the result of a computation the number of strings present in the halting configuration in a given membrane. To this aim, we need operations with strings which can increase and decrease the number of occurrences of strings.

The following four operations were considered in [1] (they are slight variants of the operations used in [15]):

1. *Replication.* If $a \in V$ and $u_1, u_2 \in V^+$, then $r : a \to u_1||u_2$ is called a *replication rule.* For strings $w_1, w_2, w_3 \in V^+$ we write $w_1 \Longrightarrow_r (w_2, w_3)$ (and we say that w_1 is replicated with respect to rule r) if $w_1 = x_1ax_2$, $w_2 = x_1u_1x_2$, $w_3 = x_1u_2x_2$, for some $x_1, x_2 \in V^*$.
2. *Splitting.* If $a \in V$ and $u_1, u_2 \in V^+$, then $r : a \to u_1|u_2$ is called a *splitting rule.* For strings $w_1, w_2, w_3 \in V^+$ we write $w_1 \Longrightarrow_r (w_2, w_3)$ (and we say that w_1 is splitted with respect to rule r) if $w_1 = x_1ax_2$, $w_2 = x_1u_1$, $w_3 = u_2x_2$, for some $x_1, x_2 \in V^*$.
3. *Mutation.* A *mutation rule* is a context-free rewriting rule, $r : a \to u$, over V. For strings $w_1, w_2 \in V^+$ we write $w_1 \Longrightarrow_r w_2$ if $w_1 = x_1ax_2$, $w_2 = x_1ux_2$, for some $x_1, x_2 \in V^*$.
4. *Recombination.* Consider a string $z \in V^+$ (as a *crossing-over block*) and four strings $w_1, w_2, w_3, w_4 \in V^+$. We write $(w_1, w_2) \Longrightarrow_z (w_3, w_4)$ if $w_1 = x_1zx_2$, $w_2 = y_1zy_2$, and $w_3 = x_1zy_2$, $w_4 = y_1zx_2$, for some $x_1, x_2, y_1, y_2 \in V^*$.

Note that replication and splitting increase the number of strings, mutation and recombination not; by sending strings out of the system, their number can also be decreased.

We work here only with multisets $\sigma : V^* \to \mathbf{N}$ such that only finitely many elements have a non-null multiplicity, thus we can specify σ in the form $A = \{(x_1, \sigma(x_1)), \ldots, (x_k, \sigma(s_k))\}$, where $x_i, 1 \leq i \leq k$, are those elements of V^* for which $\sigma(x_i) > 0$.

A *P system* (of degree $m, m \geq 1$) *with worm-objects* is a construct

$$\Pi = (V, \mu, A_1, \ldots, A_m, (R_1, S_1, M_1, C_1), \ldots, (R_m, S_m, M_m, C_m), i_0),$$

where:

- V is an alphabet;
- μ is a membrane structure of degree m (with the membranes labeled with $1, 2, \ldots, m$);
- $A_1, \ldots, A_m$ are multisets of finite support over V^*, associated with the regions of μ;
- for each $1 \leq i \leq m$, R_i, S_i, M_i, C_i are finite sets of replication rules, splitting rules, mutation rules, and crossing-over blocks, respectively, given in the following forms:
  a. replication rules: $(a \to u_1||u_2; tar_1, tar_2)$, for $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
  b. splitting rules: $(a \to u_1|u_2; tar_1, tar_2)$, for $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
  c. mutation rules: $(a \to u; tar)$, for $tar \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
  d. crossing-over blocks: $(z; tar_1, tar_2)$, for $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;

- $i_0 \in \{1, 2, \ldots, m\}$ specifies the *output membrane* of the system; it should be an elementary membrane of μ.

The $(m+1)$-tuple $(\mu, A_1, \ldots, A_m)$ constitutes the *initial configuration* of the system. By applicating the operations defined by the components $(R_i, S_i, M_i, C_i), 1 \leq i \leq m$, we can define transitions from a configuration to another one. This is done as usual in P system area, according to the following specific rules: A string which enters an operation is "consumed" by that operation, its multiplicity is decreased by one. The multiplicity of strings produced by an operation is accordingly increased. A string is processed by only one operation. For instance, we cannot apply two mutation rules, or a mutation rule and a replication one, to the same string. The strings resulting from an operation are communicated to the region specified by the target indications associated with the used rule.

The result of a halting computation consists of the number of strings in region i_0 at the end of the computation. A non-halting computation provides no output. For a system Π, we denote by $N(\Pi)$ the set of numbers computed in this way. By $NWP_m, m \geq 1$, we denote the sets of numbers computed by all P systems with at most m membranes.

In [1] it is proved that each recursively enumerable set of natural numbers (their family is denoted by nRE) can be computed by a P system as above; the result is improved in [4], where it is proved that the hierarchy on the number of membranes collapses:

Theorem 3. $nRE = NWP_6$.

It is an *open problem* whether or not the bound 6 in this theorem can be improved; we expect a positive answer.

5 One More Characterization of *RE*

The characterization of recursively enumerable languages in Theorem 1 is obtained at the price of using priorities among the rewriting rules. The use of this powerful feature can be avoided if we combine the rewriting with other features, of the types used in the previous section. More precisely, let us consider P systems working with multisets of worm-objects, processed by rewriting and crossovering rules, but let us consider as the result of a computation the language of all strings present at the end of halting computations in a specified output membrane. The work of such a system is exactly as the work of a P system with worm-objects, only the way of defining the result of a computation is different.

Let us denote by $RXP_m(i/o), m \geq 1$, the family of languages generated by such systems with at most m membranes, using as communication commands the indications *here, out, in* (it turns out that *in* is sufficient, we do not need to indicate the label of the target membrane). Somewhat expected, we get one further characterization of recursively enumerable languages.

Theorem 4. $RE = RXP_5(i/o)$.

Proof. We only prove the inclusion $RE \subseteq RXP_5(i/o)$, for the opposite one we can invoke the Turing-Church thesis.

Let $G = (N, T, S, P)$ be a type-0 grammar in the Kuroda normal form, that is, with the rules in P of one of the following forms: $A \rightarrow BC, A \rightarrow a, A \rightarrow \lambda$, and $AB \rightarrow CD$, for $A, B, C, D \in N, a \in T$. We assume that all the non-context-free rules in P are labeled in a one-to-one manner with elements of a given set *Lab.*

We construct the P system (of degree 5)

$$\Pi = (V, \mu, A_1, A_2, A_3, A_4, A_5, (R_1, \emptyset), (R_2, C_2), (R_3, C_3), (R_4, \emptyset), (R_5, \emptyset), 5),$$

with

$$\begin{aligned}
V &= N \cup T \cup \{A', A'', A''' \mid A \in N\} \cup \{A'_r, B''_r \mid r : AB \to CD \in P\} \cup \{Y\},\\
\mu &= [_1[_2[_3[_4[_5\ ]_5]_4]_3]_2]_1,\\
A_5 &= \{(S, 1)\},\\
A_3 &= \{(A'_r B'_r, 1) \mid r : AB \to CD \in P\},\\
A_1 &= A_2 = A_4 = \emptyset,
\end{aligned}$$

and with the following sets of rules:

$$\begin{aligned}
R_1 &= \{(B''_r \to B'''; here),\ (A'_r \to C'D''; in) \mid r : AB \to CD \in P\},\\
R_2 &= \{(A''' \to \lambda; in),\ (A' \to y; here) \mid A \in N\}\\
&\quad \cup \{(Y \to Y; here)\},\\
C_2 &= \{(A'_r B''_r; in, out) \mid r : AB \to CD \in P\},\\
R_3 &= \{(A' \to A; in),\ (A'' \to Y; out) \mid A \in N\}\\
&\quad \cup \{(\alpha \to Y; out) \mid \alpha \in N \cup T\}\\
&\quad \cup \{(Y \to Y; here)\},\\
C_3 &= \{(A'_r B''_r; out, out) \mid r : AB \to CD \in P\},\\
R_4 &= \{(B \to B''_r; out),\ (A'_r \to Y; out) \mid r : AB \to CD \in P\}\\
&\quad \cup \{(A'' \to A; in) \mid A \in N\}\\
&\quad \cup \{(Y \to Y; here)\},\\
R_5 &= \{(A \to x; here) \mid A \to x \in P\}\\
&\quad \cup \{(A \to A'_r; out) \mid r : AB \to CD \in P\}\\
&\quad \cup \{(A \to Y; out) \mid A \in N\}\\
&\quad \cup \{(Y \to Y; here)\}.
\end{aligned}$$

This system works as follows.

Let us assume that we have a string w in membrane 5, in only one copy; initially, this is the case with the axiom S of G. The context-free rules from P are present in R_1 as rewriting rules, hence they can be simulated without any difficulty. Assume that we use the rule $A \to A'_r$ corresponding to some rule $r : AB \to CD$ from P. The obtained string, $w_1 A'_r w_2$, is sent to membrane 4. If we apply the rule $A'_r \to Y$, then the trap-symbol Y is introduced, and the string will be rewritten forever by using the rule $Y \to Y$, hence the computation never ends. Thus, we have to use a rule of the form $B \to B''_p$, for some rule $p : XB \to EF$ from P. A string which contains both the symbols A'_r and B''_p is sent to membrane 3. If we apply a rule of the form $\alpha \to Y$, for any $\alpha \in N \cup T$, then the computation will never finish. The only way to correctly continue the computation is by using a crossovering block and this implies the fact that $r = p$ and $r : AB \to CD$. This means that the initial string was of the form $w = w_1ABw'_2$, hence in membrane 3 we have the string $w_1A'_rB''_rw'$; this string is recombined with $A'_rB''_r$, which waits here from the beginning of the computation (in only one copy) and we get the strings $w_1A'_rB''_r, A'_rB''_rw'_r$, which are both sent to membrane 2. In membrane 2 we can perform two crossovering operations:

$$(w_1A'_rB''_r, A'_rB''_rw'_r) \vdash (w_1A'_rB''_rw'_r, A'_rB''_r), \quad (in, out), \text{ and}$$
$$(A'_rB''_rw'_r, w_1A'_rB''_r) \vdash (A'_rB''_r, w_1A'_rB''_rw'_2), \quad (in, out).$$

The two cases are identical if $w_1 = w'_2 = \lambda$. If this does not hold, then in the first case we send the string $w_1A'_rB''_rw'_2$ to membrane 3 and, because we have no copy of the crossovering block $A'_rB''_r$ here, we have to use a rule $\alpha \to Y$ and the computation will never finish. Therefore, we have to proceed as in the second case, that is, we send the string $w_1A'_rB''_rw'_2$ to membrane 1 and the string $A'_rB''_r$ to membrane 3. This latter string is an axiom, it will stay in membrane 3, waiting for a possible future use. If the string $w_1A'_rB''_rw'_2$ is rewritten in membrane 1 by the rule $A'_r \to C'D''$, then the resulting string is sent to membrane 2. The only rule which can be applied is $C \to Y$ and the computation will never finish. Thus, before using the rule $A'_r \to C'D''$, in membrane 1 we have to use the rule $B''_r \to B'''$. This means that the string $w_1C'D''B'''w'_2$ arrives to membrane 2; the symbol B''' is erased and the obtained string is sent to membrane 3. By the use of $D'' \to Y$ or $\alpha \to Y, \alpha \in N \cup T$, we lead to non-halting computations, hence we have to continue by using the rule $C' \to C$. The string is sent to membrane 4. If here we apply a rule $X \to X''_p$, for some rule $p : ZX \to YW \in P$, then the obtained string is sent to membrane 3, where the only applicable rules are $D'' \to Y$ and, if any symbol $\alpha \in N \cup T$ appears in the string, $\alpha \to Y$. The computation never finishes, hence the way to proceed in membrane 2 is by using the rule $D'' \to D$. We send to membrane 5 the string $w_1CDw'_2$, which is the correct result of simulating the rewriting of the string w by means of the rule $r : AB \to CD$.

The process can be iterated, hence each derivation in G can be simulated in Π and, conversely, all halting computations in Π correspond to correct derivations in G. Note that the multiplicities of strings in membrane 5 and in membrane 3, the only ones where we have strings in the beginning of a computation, are restored. As long as any nonterminal symbol A is still present in the current string from membrane 5, the rule $A \to Y$ can be used. Therefore, the computation in Π can stop only after reaching a terminal string with respect to G.

In conclusion, $L(G) = L(\Pi)$, which concludes the proof. □

It is an *open problem* whether or not the bound 5 in the previous theorem can be decreased. Anyway, it is easy to see that $RXP_1(i/o)$ contains all context-free languages, while the example from the end of Section 2 shows that $RXP_2(i/o)$ contains non-context-free languages.

References

[1] J. Castellanos, A. Rodriguez-Paton, Gh. Păun, Computing with membranes: P systems with worm-objects, submitted, 2000.

[2] T. Head, Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology*, 49 (1987), 737–759.

[3] S. N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP-complete problems, *Romanian J. of Information Science and Technology*, 2, 4 (1999).

[4] C. Martin-Vide, Gh. Păun, Computing with membranes. One more collapsing hierarchy, submitted, 2000.

[5] A. Păun, M. Păun, On the membrane computing based on splicing, in vol. *Words, Sequences, Languages* (C. Martin-Vide, V. Mitrana, ed.), Kluwer, Dordrecht, 2000.

[6] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, in press, and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).

[7] Gh. Păun, Computing with membranes. An introduction, *Bulletin of the EATCS*, 67 (1999), 139–152.

[8] Gh. Păun, Computing with membranes – A variant: P systems with polarized membranes, *Intern. J. of Foundations of Computer Science*, 11, 1 (2000), and *Auckland University, CDMTCS Report* No 098, 1999 (www.cs.auckland.ac.nz/CDMTCS).

[9] Gh. Păun, P systems with active membranes: Attacking NP complete problems, *J. Automata, Languages, and Combinatorics*, to appear, and *Auckland University, CDMTCS Report* No 102, 1999 (www.cs.auckland.ac.nz/CDMTCS).

[10] Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Heidelberg, 1998.

[11] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, to appear, and *Turku Center for Computer Science-TUCS Report* No 218, 1998 (www.tucs.fi).

[12] Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori, Further remarks on P systems with membrane division, submitted, 2000.

[13] Gh. Păun, T. Yokomori, Membrane computing based on splicing, *Preliminary Proc. of Fifth Intern. Meeting on DNA Based Computers* (E. Winfree, D. Gifford, eds.), MIT, June 1999, 213–227.

[14] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.

[15] M. Sipper, Studying Artificial Life using a simple, general cellular model, *Artificial Life Journal*, 2, 1 (1995), 1–35.