Membrane Computing Schema Based on String Insertions

Mario J. Pérez-Jiménez[1], Takashi Yokomori[2]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

[2] Department of Mathematics
Faculty of Education and Integrated Arts and Sciences
Waseda University, 1-6-1 Nishiwaseda, Shinjuku-ku
Tokyo 169-8050, Japan
yokomori@waseda.jp

Summary. In this note we introduce the notion of a membrane computing schema for string objects. We propose a computing schema for a membrane network (i.e., tissue-like membrane system) where each membrane performs unique type of operations at a time and sends the result to others connected through the channel. The distinguished features of the computing models obtained from the schema are:

1. only *context-free insertion operations* are used for string generation,
2. some membranes assume filtering functions for *structured objects(molecules)*,
3. the generating model and accepting model are obtained in the *same schema*, and both are computationally universal,
4. several known rewriting systems with universal computability can be reformulated in terms of membrane computing schema in a uniform manner.

The first feature provides the model with a simple uniform structure which facilitates a biological implementation of the model, while the second feature suggests further feasibility of the model in terms of DNA complementarity.

Through the third and fourth features, one may have a unified view of a variety of existing rewriting systems with Turing computability in the framework of membrane computing paradigm.

1 Introduction

In the theory of bio-inspired computing models, membrane systems (or P systems) have been widely studied from various aspects of the computability such as the optimal system designs, the functional relations among many ingredients in different levels of computing components, the computational complexity and so forth.

Up to the present, major concerns are focused on the computational capability of multisets of certain objects in a membrane structure represented by a rooted tree, and there are a relatively limited amount of works in the membrane structure of other types (like a network or graph) on string objects and their languages; those are, for example, in the context of P system on graph structure ([15]), of the tissue P systems ([10]) and of spiking neural P systems ([2, 6]).

On the other hand, in DNA computing theory, a string generating device called insertion-deletion system has been proposed and investigated from the uniqueness of non-rewriting nature in generating string objects. Among others, string insertion operation with no context is of our particular interests, because of the relevance to biological feasibility in terms of DNA sequences.

In this paper, we are concerned with tissue-like membrane systems with string insertion operations and investigate the computational capability of those systems. By using the framework of tissue-like membrane systems, however, our major focus is on studying the new aspects of the computational mechanisms used in a variety of existing models based on string rewriting.

To this aim, we propose the notion of a membrane computing schema which provides a unified view and framework to investigate new aspects of the variety of computational mechanisms. That is, by a membrane computing schema Π, we represent a *core* structure of the computing model M at issue. At the same time, we also consider an interpretation I to Π which specifies the *details* of M. Then, we have M that is embodied as a tissue-like membrane system $I(\Pi)$ with string insertion operation.

The advantages of this schematic approach to computing are the following: (1) High transparency of the computing mechanism is obtained from separating the skeletal (core) part from other detailed specificity of the computation. (2) Structural modularity of the computing model facilitates our better understanding of the computing mechanism.

With this framework we will present not only new results of the computing models with universal computability but also a unified view of those models from the framework of tissue-like membrane system with string insertion operations.

2 Preliminaries

We assume the reader to be familiar with all formal language notions and notations in standard use. For unexplained details, consult, e.g., [14, 16].

For a string x over an alphabet V (i.e., x in V^*), $lg(x)$ denotes the length of x. For the empty string, we denote it by λ. For an alphabet V, $\overline{V} = \{\overline{a} \mid a \in V\}$. A binary relation ρ over V is called an *involution* if ρ is injective and ρ^2 is an identity (i.e., for any $a \in V$, if we write $\rho(a) = \overline{a}$, then it holds that $\rho(\overline{a}) = a$). A *Dyck* language D_k over $V \cup \overline{V}$ is a language generated by a context-free grammar $G = (\{S\}, V, P, S)$, where $P = \{S \to SS, S \to \lambda\} \cup \{S \to aS\overline{a} \mid a \in V\}$ and k is the cardinality of V.

An *insertion system* ([9]) is a triple $\gamma = (V, P, A)$, where V is an alphabet, A is a finite set of strings over V called axioms, and P is a finite set of insertion rules. An *insertion rule* is of the form (u, x, v), where $u, x, v \in V^*$. We define the relation $\mapsto$ on V^* by $w \mapsto z$ iff $w = w_1uvw_2$ and $z = w_1uxvw_2$ for some insertion rule $(u, x, v) \in P$, $w_1, w_2 \in V^*$. As usual $\mapsto^*$ denotes the reflexive and transitive closure of $\mapsto$. The *insertion language* generated by γ is defined as follows: $L(\gamma) = \{w \in V^* | \ s \mapsto^* w, \ s \in A\}$. An insertion rule of the form (λ, x, λ) is said to be *context-free*, and we denote it by $\lambda \to x$.

We denote by RE, CF, LIN and RG the families of recursively enumerable languages, of context-free languages, of linear languages, and of regular languages, respectively.

A *matrix grammar with appearance checking* 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 $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, 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 (we say that 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 $(A_1 \to x_1, \ldots, A_n \to x_n)$ 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 $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If $F = \emptyset$, then the grammar is said to be without appearance checking (and F is no longer mentioned).

We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$.

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}. When we use only grammars without appearance checking, then the obtained family is denoted by MAT. It is known that $MAT \subset MAT_{ac} = RE$.

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 of one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in N_2 \cup N_2^2 \cup T \cup \{\lambda\}$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T \cup \{\lambda\}$.

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; $\#$ is a trap-symbol, once introduced, it is never removed. A matrix of type 4 is used only once, at the last step of a derivation.

For each matrix grammar (with appearance checking) there effectively exists an equivalent matrix grammar (with appearance checking) in the binary normal

form. (Note that the definition of the binary normal form presented here is a variant of the one in [4].)

A *random context grammar* is a construct $G = (N, T, S, P)$, where N, T are disjoint alphabets, $S \in N$, P is a finite set of rules of the form $(A \to x, Q, R)$, where $A \to x$ is a context-free rule $(A \in N, x \in (N \cup T)^*)$, Q and R are subsets of N.

For $\alpha, \beta \in (N \cup T)^*$ we write $\alpha \Longrightarrow \beta$ iff $\alpha = uAv$, $\beta = uxv$ for some $u, v \in (N \cup T)^*$, $(A \to x, Q, R) \in P$, all symbols of Q appear in uv, and no symbol of R appears in uv. We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$.

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 RC. It is known that $RC = RE$ ([4]).

3 Membrane Computing Schema, Interpretation and Languages

We now introduce the notion of a membrane computing schema in a general form, then we will present a restricted version, from which a variety of specific computing models based on insertion operations and filtering can be obtained in the framework of a tissue-like membrane computing. That is, a membrane computing schema is given as a skeletal construct consisting of a number of *membranes* connected with synapses (or channels) whose structure may be taken as a kind of *tissue P systems* (e.g., [10]).

3.1 Membrane Computing Schema

A *membrane computing schema* of degree (k, p) is a construct

$$\Pi = (V, T, Syn, Com, Ope, Fil, i_s, Out),$$

where:

(i) V is a finite alphabet with an involution relation ρ called the *working alphabet*.
(ii) T is a subset of V called the *terminal alphabet*.
(iii) $Syn \subseteq (Com \times Ope) \cup (Ope \times Com) \cup (Com \times (SubFil \cup \{SF\})) \cup (\{FF\} \times Com) \cup (SubFil \times Ope) \cup \{(SF, FF), (FF, Out)\}$,
where $Com = \{Com_1, \cdots, Com_p\}$ called a set of *communication cells*,
$Ope = \{Ope_1, \cdots, Ope_k\}$ called a set of *operation cells*,
$Fil = \{SF, FF\} \cup SubFil$ called a set of *filtering cells*,
where $SubFil = \{FF_1, \cdots, FF_t\}$ called a set of *subfilter cells*.
Syn determines the tissue membrane structure of Π. (See Figure 1. In the figure, the thick arrow indicates multiple arrows of one-way or two-way directions.)

(iv) Each cell Com_i serves as a communication channel. That is, any string x in the cell Com_i is sent out to all the cells indicated by $Syn(Com_i)$.
(v) Each cell Ope_j consists of a finite number of rules $\{\sigma_{j1}, \cdots, \sigma_{js}\}$, where each σ_{ji} is a string insertion operation of the form : $\lambda \rightarrow u$, where $u \in V^*$.
(vi) SF and FF, called *structured filter* and *final filter*, are associated with two languages L_{SF} and L_{FF} over V, respectively. Further, each cell FF_i, called *subfilter*, is also associated with a language L_{FF_i}.
(vii) $i_s(= Com_1)$ is the distinguished cell (to designate some specific role).
(viii) Out is the *output cell* (for obtaining the outputs).

3.2 Interpretation of Π

In order to embody a membrane computing schema Π, we need to give more information about Π which specifies each operation in Ope, and materializes SF and FF. Let us consider such an interpretation I to the schema Π which enables us to have a computing model $I(\Pi)$ that is feasible in a practical sense.

[Notation] For any x in $Com \cup SubFil \cup \{FF\}$, let $Syn(x) = \{\ j\ \mid\ (x, y_j) \in Syn\}$ and for any y in $Ope \cup SupFil \cup \{SF\}$, let $Syn^{-1}(y) = \{\ i\ \mid\ (x_i, y) \in Syn\}$. (Note that both $Syn(x)$ and $Syn^{-1}(y)$ refer to sets of *indices*.)

Formally, an *interpretation* I to Π is a construct $I = (\{R_1, \cdots, R_k\}, L_{SF}, L_{FF}, \{L_{FF_i} \mid 1 \leq i \leq t\})$, where

(i) R_i specifies a set of insertion operations used in Ope_i (for $i = 1, \cdots, k$)
(ii) L_{SF} (L_{FF}) materializes a concrete specification about the function of SF (FF, respectively). In practical operational phases (described below), we assume the following:
• In the cell SF, each string is assumed to form a certain *structure* (e.g., structured molecule based on hybridization in terms of H-bonds via minimal energy principle). SF takes as input a string u over V and allows it to filter through if it is in L_{SF} (otherwise, the string u is lost). Then, after building up a structured form $s(u)$ of u, SF removes all parts of structures from $s(u)$ and produces as output the concatenation of all remaining strings. The output v is sent out to the cell FF.
• FF receives as input a string v over V (from SF). A string v filters through if it is in L_{FF} and is sent out to Out. Otherwise, it is sent out to all cells indicated by $Syn(FF)$.
• Each FF_i receives as input a string u over V. Then, a string v filters through if it is in L_{FF_i} and is sent out to all cells indicated by $Syn(FF_i)$. Otherwise, it is lost.
• Filtering applies simultaneously to all strings in the filtering cell.

Note that in the case $SubFil$ is empty in a given Π, an interpretation to Π is simply written as $I = (\{R_1, \cdots, R_k\}, L_{SF}, L_{FF})$.

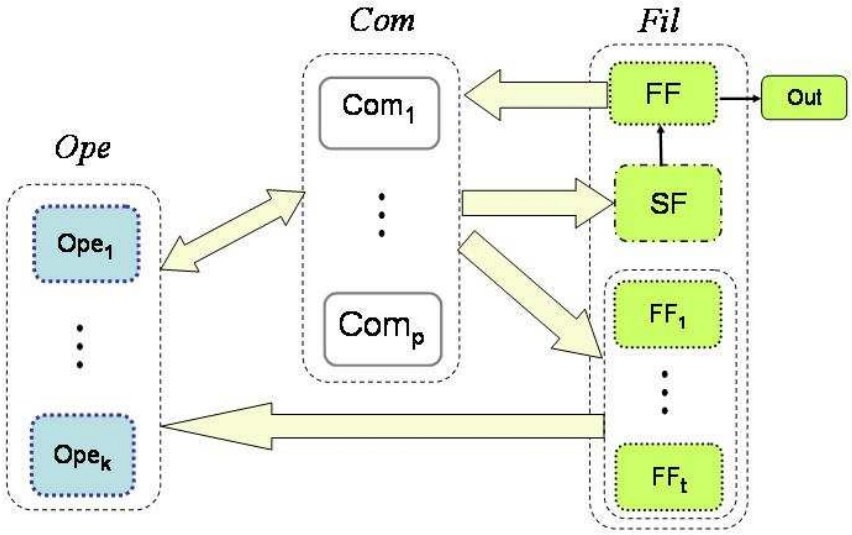

Fig. 1. Modular Structure of Membrane Network in Π

3.3 Transitions and Languages

Given a schema Π and its interpretation I, we now have a membrane system $I(\Pi)$ based on string insertions. In what follows, we define a transition sequence of $I(\Pi)$ and the language associated with $I(\Pi)$.

The $(p+1)$-tuple of languages over V represented by $(L_1, \ldots, L_p, L_{out})$ constitutes a *configuration* of the system, where each L_i represents the set of all strings in the cell Com_i (for all $i = 1, \cdots, p$), and L_{out} is the set of strings presented in the output cell at some time instance.

Let $C_1 = (L_1, \ldots, L_p, L_{out})$ and $C_2 = (L'_1, \ldots, L'_p, L'_{out})$ be two configurations of the system.

We define one transition from C_1 to C_2 in the following steps:

(0) **Pre-checking Step**: For each $\ell = 1, \cdots, t$, let $Syn^{-1}(FF_\ell) = \{\ell_1, \cdots, \ell_q\}$ and consider $L[\ell] = \cup_{i=1}^{q} L_{\ell_i}$. Then, each cell FF_ℓ filters out all strings of $L[\ell]$ that are not in L_{FF_ℓ}, and all strings that have passed through are sent to all the cells indicated by $Syn(FF_\ell)$. (In the case of $t = 0$, this step is skipped.)

(1) **Evolution Step**: For each $j = 1, \cdots, k$, let $\sigma_{j1}, \cdots, \sigma_{js}$ be all the operations given in Ope_j.

Suppose that we apply operations $\sigma_{ji} : \lambda \to u_{ji}$ $(1 \leq i \leq s)$ to a string v which means that each σ_{ji} is applied to v *simultaneously*. Further, when we apply

σ_{ji} to v, the location in v to insert u_{ji} is nondeterministically chosen and the result $\sigma_{ji}(v)$ is considered as the set of *all possible strings* obtained from v by σ_{ji}. (Note that if two or more rules share the same location to insert, then all possible permutations of those rules are considered to apply to the location.) The result of such an application of all operations in Ope_j to v is denoted by $Ope_j(v)$.

Let $L(j) = \cup_{m=1}^{t} L_{j_m}$, where $Syn^{-1}(Ope_j) = \{j_1, \cdots, j_t\}$. Then, the total result performed by Ope_j to $L(j)$ is defined as

$$Ope_j(L(j)) = \cup_{v \in L(j)} Ope_j(v).$$

This result is then sent out to all cells indicated by $Syn(Ope_j)$ simultaneously.

(2) **Filtering Step**: For each $i = 1, \cdots, p$, let $\tilde{L}_i = \cup_{n=1}^{r} Ope_{i_n}(L(i_n))$, where $Syn(Com_i) = \{i_1, \cdots, i_r\}$. Further, let $L_e = \cup_{n=1}^{g} \tilde{L}_{i_n}$, where $Syn^{-1}(SF) = \{i_1, \cdots, i_g\}$.

SF takes as input the set L_e and produces as output a set of strings L_f. (Recall that in the cell SF, each string u is assumed to form a certain structure, and the output of SF is the reduced string by removing structural parts from u in L_{SF}.) Then, SF sends out L_f to FF. (Any element of L_e that was filtered off by SF is assumed to be lost.)

Finally, the cell FF filters out strings of L_f depending upon whether they are in L_{FF} or not. All strings in L_f that passed through FF are sent out to Out, while others are simultaneously sent to Com_i for all $i \in Syn(FF)$ or they are all lost if $Syn(FF) = \emptyset$.

Let L_{ff} be the set of all strings that were filtered off by FF. Then, we define $C_2 = (L'_1, \ldots, L'_p, L'_{out})$ by setting for each $i = 1, \cdots, p$

$$L'_i = \begin{cases} L_{ff}, \text{ if } i \in Syn(FF), \\ \tilde{L}_i, \quad \text{otherwise.} \end{cases}$$

Further, let $L'_{out} = L_{out} \cup L_f(Out)$, where $L_f(Out) = L_f - L_{ff}$ (the set of all stings that have passed through FF and been sent to Out).

Remarks.
(1) Each cell in Com not only provides a buffer for storing intermediate results in the computation process but also *transmits* them to the cells specified by Syn.
(2) The system has a global clock and counts time in such a way that every cell (including communication cells) takes one unit time to perform its task irrespective of the existence of strings in it, while a pre-checking step by a subfilter cell FF_i is assumed to be executed within the unit time for its corresponding operation cells in $Syn(FF_i)$.

Let I be an interpretation of Π. When we have a transition from C_1 to C_2 of $I(\Pi)$, we write $C_1 \Longrightarrow C_2$. A configuration $C_0 = (\{w\}, \emptyset, \cdots, \emptyset)$, where $w \in V^*$, is called the *initial configuration*. A sequence of transitions between configurations of the system $I(\Pi)$, starting from the initial configuration, is called a *computation* of $I(\Pi)$.

Let $\Longrightarrow^*$ be the reflexive and transitive closure of $\Longrightarrow$. For any $n \geq 0$, let $C_0 \Longrightarrow^n C_n = (L_{1,n}, \cdots, L_{p,n}, L_{out}^{(n)})$ be a computation of $I(\Pi)$ with n transitions from C_0. We consider two types of computing models induced from $I(\Pi)$; one is the generating model and the other the accepting model.

[*Generating Model*] In the case of a generating model, we consider all computations whose results are present in the output cell. That is, we define the language generated by $I(\Pi)$ as follows:

$$L_g(I(\Pi)) = \cup_{n \geq 0} L_{out}^{(n)}.$$

[*Accepting Model*] In the case we consider an accepting model, let w be an input string. Then, we assume that if w is recognized, then the result *Yes* or *No* is present in the output cell after a certain number of transitions from $C_0 = (\{w\}, \emptyset, \cdots, \emptyset)$. Thus, we define the language accepted by $I(\Pi)$ as follows:

$$L_a(I(\Pi)) = \{w \in T^* \mid Yes \in L_{out}^{(n)} \text{ for some } n \geq 0\}.$$

For a class of interpretations $\mathcal{I}$ to Π, we denote by $\mathcal{LMS}_x(\Pi, \mathcal{I})$ the family of all languages $L_x(I(\Pi))$ specified by those systems as above, where I is in $\mathcal{I}$. That is,

$$\mathcal{LMS}_x(\Pi, \mathcal{I}) = \{L_x(I(\Pi)) \mid I \in \mathcal{I}\},$$

where x is in $\{a, g\}$.

4 Characterizations by Membrane Schema Π_0

The structure of the membrane computing schema Π introduced in the previous section seems to be general enough to induce a computing device with universal computability power by finding an appropriate interpretation. In what follows, we will show that such a universal computability can be realized by much simpler schemas together with appropriate interpretations of moderately simple filtering cells.

First we consider the following simple membrane computing schema:

$$\Pi_0 = (V, T, Syn, Com, Ope, Fil, i_s, Out),$$

where:

(1) V, T, Ope, i_s and Out are the same as in Π,
(2) $Fil = \{SF, FF\}$ (i.e., $SubFil$ is empty),
(3) $Com = \{Com_1, Com_2\}$,
(4) $Syn = \{Com_1, Ope_i), (Ope_i, Com_2) \mid 1 \leq i \leq k\}$
$\cup\{(FF, Com_1), (Com_2, SF), (SF, FF), (FF, Out)\}$ (see (a) of Figure 2).

We are now in a position to present our first result.

Theorem 4.1 *There exists* $\mathcal{I}_G$ *such that* $RE = \mathcal{LMS}_g(\Pi_0, \mathcal{I}_G)$.

Proof. We prove only the inclusion $\subseteq$. (The opposite inclusion is a consequence of the Turing-Church thesis.)

Let L be any language in RE that is generated by a Chomsky type-0 grammar $G = (N, T, S, P)$. Then, we consider the following interpretation $I_G = (R_G, L_{SF}, L_{FF})$, where

(i) For each $r : u \to v$ in P, construct $R_r = \{\lambda \to vr,\ \lambda \to \overline{u}^R\overline{r}\}$, and let $R_G = \{\ R_r \mid r \in P\}$.

(ii) • L_{SF} is given as following language:

$$L_{mir} = V^*\{\ w\overline{w}^R \mid w \in V^*\}V^* = \{\ xw\overline{w}^R y \ \mid\ x, y, w \in V^*\ \}\ .$$

That is, a string filters through SF iff it is an element in L_{mir}, where $V = N \cup T \cup \{r \mid r \in P\}$. (Recall that, by definition of SF, any string in SF is assumed to form a structure, and we assume the hybridization by involution relation ρ over V. Specifically, SF performs two functions: it only accepts all structures of molecules containing a special hairpin $w\overline{w}$, and then it removes the portion of a hairpin from the structure and send out the rest part of the string to FF(see (b) of Figure 2). The structures rejected by SF are *all lost*.)

• L_{FF} is simply given as T^*, so that only strings in T^* can pass through FF and are sent to the output cell Out. Other strings are all sent to Com_1. (Note that (FF, Com_1) is in Syn of Π_0.)

For any $n \geq 1$, let $C_0 = (\{S\}, \emptyset, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{2,n}, L_{out}^{(n)})$ be a computation with n transitions in $I_G(\Pi_0)$. Suppose that $S \Longrightarrow^{n-1} \alpha \Rightarrow_r \beta$ in G, where $\alpha = xuy$, $\beta = xvy$ and $r : u \to v$ is used. Then, we can show that

(i) α is in $L_{1,(n-1)}$,

(ii) $\beta' = xvru\overline{u}^R\overline{r}y$ is in SF, and

(iii) after filtering by SF, the reduced string of β', i.e., a string $xvy(= \beta)$ is sent to FF.

In FF, if β is not in T^*, then it is sent to Com_1 and, therefore, in $L_{1,n}$, where $C_n = (L_{1,n}, L_{2,n}, L_{out}^{(n)})$. Otherwise, β is sent to $L_{out}^{(n)}$. That is, if β is in $L(G)$, then we have that β is in $L_g(I_G(\Pi_0))$.

Conversely, suppose that for any $n \geq 1$, α is in $L_{1,n}$. Then, there exists $\alpha' = \alpha_1 w\overline{w}^R\alpha_2$ in Com_2 such that $\alpha = \alpha_1\alpha_2$. From the way of constructing R_G, there uniquely exists $r : u \to v$ in P such that $R_r = \{\lambda \to vr, \lambda \to \overline{u}^R\overline{r}\}$ and $w = ru$. (No other $R_{r'}(r' \neq r)$ can make a substring $w\overline{w}^R$ by insertion operations because of the uniqueness of r.)

Therefore, we can write $\alpha' = \alpha_1 ru\overline{u}^R\overline{r}\alpha_2$ for some α_1, α_2. Then, there must exist α_1' such that $\alpha_1 = \alpha_1' v$ because of the rule $\lambda \to vr$. Hence, we have that $\alpha' = \alpha_1' vru\overline{u}^R\overline{r}\alpha_2$ from which we can derive that $\alpha_1' u\alpha_2$ is in $L_{1,(n-1)}$. Thus, there exists a derivation: $\alpha_1' u\alpha_2 \Longrightarrow_r \alpha_1' v\alpha_2 = \alpha$ in G. By iteratively applying the above argument, we eventually conclude that there exists a derivation $S \Longrightarrow^n \alpha$ in G.

Taking $L_{1,0} = \{S\}$ into consideration, it holds that for any $n \geq 0$, $L_{1,n} = \{\alpha \mid S \Longrightarrow^n \alpha \text{ in } G\}$. If α is in $L_g(I_G(\Pi_0))$, then it is also in $L(G)$. Thus, we have that $L(G) = L_a(I_G(\Pi_0))$, which completes the proof. □

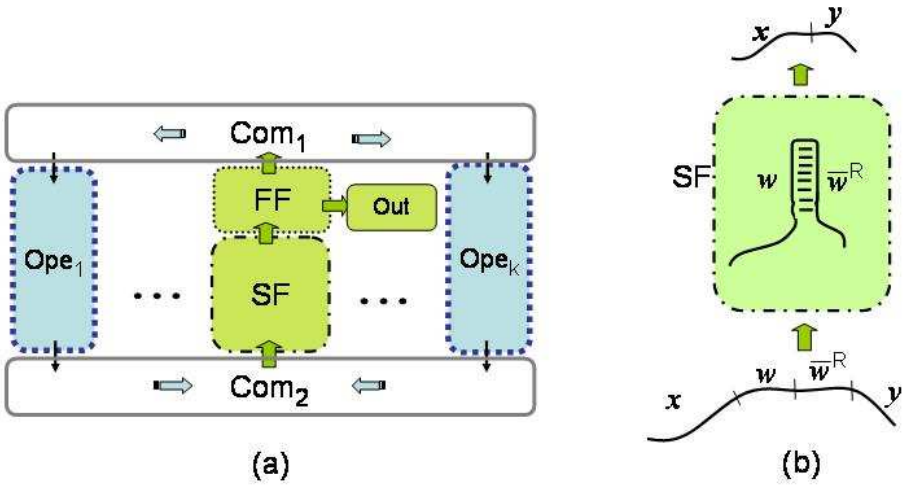

Fig. 2. Membrane Computing Schema: Π_0

Theorem 4.2 *There exists $\mathcal{I}_M$ such that $RE = \mathcal{LMS}_a(\Pi_0, \mathcal{I}_M)$.*

Proof. We use the same strategy as in Theorem 4.1, but start with a (nondeterministic) Turing machine M. That is, let L be any language in RE accepted by $M = (Q, T, U, \delta, p_0, B, F)$, where $B(\notin U)$ is a blank symbol. (Without loss of generality, we may assume that M immediately stops as soon as it enters into a final state of F.) An instantaneous description (ID) of M is represented by a string $xpay$ in U^*QU^*, where xay is the tape content ($x, y \in U^*$, $a \in U$) and M is in the state $p(\in Q)$ and the tape head is on a.

Given an input $w(\in T^*)$, M starts computing w from the state p_0, which is represented by an ID: p_0w. (We assume that the tape content has the left-boundary (the leftmost of w) and no right-boundary where blank symbols B are initially filled.) In general, suppose that a transition rule $(p, a) \to (q, c, i) \in \delta$ is applied to an ID $xbpay$. Then, we have a transition between IDs of M:

- $xbpay \Longrightarrow xbcqy$ (if $i = R$ and $a \neq B$),
- $xbpay \Longrightarrow xqbcy$ (if $i = L$ and $a \neq B$),
- $xbpa \Longrightarrow xbcq$ (if $i = R$, $a = B$ and $y = \lambda$),

- $xbpa \Longrightarrow xqbc$ (if $i = L$, $a = B$ and $y = \lambda$).

Thus, in each case one can consider a rewriting rule : for example, a rule $pa \to cq$ for the case (i). Let P_M be the set of rewriting rules obtained from δ in this manner. We define an interpretation $I_M = (R_M, L_{SF}, L_{FF})$ as follows:

(i) For each $r : u \to v$ in P_M, construct $R_r = \{\lambda \to vr,\ \lambda \to \overline{u}^R\overline{r}\}$, and let $R_M = \{\ R_r \mid r \in P_M\}$.
(ii) L_{SF} is the same as the one in I_G.
(iii) L_{FF} is given as V^*FV^*, where $V = U \cup Q \cup \{B\} \cup \{r \mid r \in P_M\}$.

Let w be any string in T^* and $n \geq 0$. Then, from the way of constructing I_M together with discussion above, it is easily seen that $p_0w \Longrightarrow^n xqy$ for some $q \in F$, $x, y \in U^*$ iff there exists $Yes \in L_{out}^{(n)}$ such that $C_0 = (\{w\}, \emptyset, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{2,n}, L_{out}^{(n)})$. Thus, we have that $L(M) = L_a(I_M(\Pi_0))$, which completes the proof. □

5 Further Results by Some Variants of Membrane Schema

We give now another membrane computing schema $\Pi_{1,t}$ which is a variant of Π_0 and also able to induce a family of computing devices $I(\Pi_{1,t})$ (with an appropriate interpretation I) that can characterize RE.

The membrane computing schema $\Pi_{1,t}$ is given as follows:

$$\Pi_{1,t} = (V, T, Syn, Com, Ope, Fil, i_s, Out),$$

where:

(1) V, T, Ope, i_s and Out are the same as in Π_0.
(2) $Com = \{Com_1\}$.
(3) $Fil = \{SF, FF\} \cup SubFil$, where $SubFil = \{FF_i \mid 1 \leq \forall i \leq t\}$ or $= \emptyset$ $(t = 0)$.
(4) $Syn = \{(Com_1, FF_i), (FF_i, Ope_i) \mid 1 \leq i \leq t\} \cup \{(Com_1, Ope_j) \mid t{+}1 \leq j \leq k\}$
$\cup (Ope_i, Com_1) \mid 1 \leq \forall i \leq k\} \cup \{(Com_1, SF), (SF, FF), (FF, Out)\}$.
(aee (a) of Figure 3).

Note: In (3) and (4) above, t can take any integer between 0 and k, and when $t = 0$, it means the corresponding set is empty.

The difference between $\Pi_{1,t}$ and Π_0 is that $\Pi_{1,t}$ has only one Com_1, while several Ope_is possibly require subfiltering cells FF_i for prechecking of strings.

Notation. We denote by $\Pi_{1.5}$ a schema $\Pi_{1,t}$ where $1 \leq t < k$, and write Π_1 for a schema $\Pi_{1,k}$(i.e., t coincides with k).

Theorem 5.1 *There exists $\mathcal{I}_{G_m}$ such that $RE = \mathcal{LMS}_g(\Pi_{1.5}, \mathcal{I}_{G_m})$.*

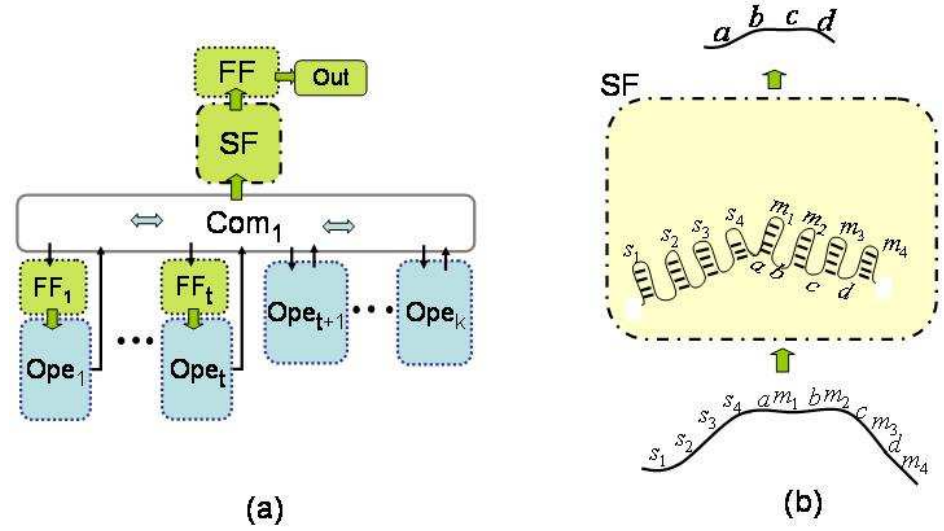

Fig. 3. Membrane Computing Schema $\Pi_{1.5}$

Proof sketch. We use a similar argument to the one in the proof of Theorem 4.1 and start with a matrix grammar. That is, let L be any language in RE generated by a matrix grammar $G_m = (N, T, S, M, F)$ with appearance checking, where $N = N_1 \cup N_2 \cup \{S, \#\}$, and we may assume that G is in the binary normal form.

We consider the following interpretation $I_{G_m} = (R_{G_m}, L_{SF}, L_{FF}, \{L_{FF_i} \mid 1 \leq i \leq t\})$, where

(i) (1) Let k be the cardinality of M and t be the number of appearance checking matrix rules in M. For each appearance checking rule $m_i : (X \to Y, A \to \#)$ $(1 \leq i \leq t)$, construct $R_{m_i} = \{\lambda \to Y s_{m_i},\ \lambda \to \overline{X}\overline{s}_{m_i}\}$.
(2) For other rules $m_j : (X \to Y, A \to x)$ in M (where $Y \in N_1 \cup \{\lambda\}, x \in T \cup N_2 \cup N_2^2 \cup \{\lambda\};\ t+1 \leq j \leq k$), construct $R_{m_j} = \{\lambda \to Y s_{m_j},\ \lambda \to \overline{X}\overline{s}_{m_j}, \lambda \to x r_{m_j}, \lambda \to \overline{A}\overline{r}_{m_j}\}$. Then, let $R_{G_m} = \{R_{m_1}, \cdots, R_{m_k}\}$.

(ii) • L_{SF} is given as the regular language $L_{mat} = L_s L_m$, where

$$L_s = \{s_m X \overline{X} \overline{s}_m \mid m : (X \to Y, A \to \#) \in M\}^*, \text{ and}$$
$$L_m = (T \cup \{r_m A \overline{A} \overline{r}_m \mid m : (X \to Y, A \to y) \in M\})^*.$$

• L_{FF_i} is given as follows: Let t be the number of appearance checking matrix rules in M. For each appearance checking rule $m_i : (X \to Y, A \to \#)$ $(1 \leq i \leq t)$, consider $L_{m_i} = (V \cup \overline{V})^* - (V \cup \overline{V})^* \{A\} (V \cup \overline{V})^*$, where $V = T \cup N \cup \{s_m, r_m \mid m \in M\}$. Then, L_{FF_i} is given as L_{m_i}. (Thus, FF_i performs in such a way that it allows only strings in L_{m_i} to pass through and send them to the cell Ope_i. Other strings are all lost.)

• Finally, L_{FF} is given as T^*, so that only strings in T^* can pass through FF and are sent to Out.

Let $C_0 = (\{XA\}, \emptyset)$, where $(S \to XA) \in M$, and consider a transition sequence $C_0 \Longrightarrow^n C_n = (L_{1,n}, L_{out}^{(n)})$. Then, for any $\alpha \in (N \cup T)^*$ and $n \geq 0$, it holds that $\alpha \in L_{1,n}$ iff $XA \Longrightarrow^n \alpha$ in G_m. Thus, we have $L(G_m) = L_g(I_{G_m}(\Pi_{1.5}))$. □

Theorem 5.2 *There exists $\mathcal{I}_{G_r}$ such that $RE = \mathcal{LMS}_g(\Pi_1, \mathcal{I}_{G_r})$.*

Proof sketch. We use the same argument as the one in the proof of Theorem 5.1, but we start with a random context grammar $G_r = (N, T, S, P)$ generating an arbitrary recursively enumerable language L. (See Preliminary section.) Then, consider the following interpretation $I_{G_r} = (R_{G_r}, L_{SF}, L_{FF}, \{L_{FF_i} \mid 1 \leq i \leq k\})$, where

(i) Let k be the cardinality of P. For each rule $r_i : (A \to x, Q, R)$ $(1 \leq i \leq k)$, construct $R_{r_i} = \{\lambda \to xr_i,\ \lambda \to \overline{A}\overline{r_i}\}$. Then, let $R_{G_{r_i}} = \{R_{r_i} \mid r_i \in P\}$.

(ii) • L_{SF} is defined by the regular language L_m (in L_{SF} for I_{G_m}).
• L_{FF_i} is given as follows: For each rule $r_i : (A \to x, Q, R)$, if $A \in Q$, then let $L_{r_i} = (V \cup \overline{V})^*\{A\}(V \cup \overline{V})^*\{A\}(V \cup \overline{V})^* \cap \cap_{X \in Q - \{A\}}(V \cup \overline{V})^*\{X\}(V \cup \overline{V})^* \cap \cap_{X \in R}((V \cup \overline{V})^* - (V \cup \overline{V})^*\{X\}(V \cup \overline{V})^*)$. Otherwise (i.e., $A \notin Q$), let $L_{r_i} = \cap_{X \in Q}(V \cup \overline{V})^*\{X\}(V \cup \overline{V})^* \cap \cap_{X \in R}((V \cup \overline{V})^* - (V \cup \overline{V})^*\{X\}(V \cup \overline{V})^*)$, where $V = N \cup T \cup \{r \mid r \in P\}$. Then, L_{FF_i} is defined by L_{r_i}. (That is, each FF_i performs in such a way that it allows only strings in L_{r_i} to pass through and send them to the cell Ope_i. Other strings are all lost.)
• Finally, L_{FF} is given as T^*, so that only strings in T^* can pass through FF and are sent out to Out.

Let $C_0 = (\{S\}, \emptyset)$, where S is the starting symbol of G_r, and consider a transition sequence $C_0 \Longrightarrow^n C_n = (L_{1,n}, L_{out}^{(n)})$. Then, for any $\alpha \in (N \cup T)^*$ and $n \geq 0$, it holds that $\alpha \in L_{1,n}$ iff $XA \Longrightarrow^n \alpha$ in G_r. Thus, we have that $L(G_r) = L_g(I_{G_r}(\Pi_1))$. □

We give yet another membrane computing schema Π_2 which is simpler than Π_0 but still able to provide the universal computability of those models induced from the schema with appropriate interpretations, but at the price of increasing the structural complexity in the filtering function SF.

The membrane computing schema Π_2 is given as follows:

$$\Pi_2 = (V, T, Syn, Com, Ope, Fil, i_s, Out),$$

where:

(1) V, T, Ope, Fil, i_s and Out are the same as in Π_0.
(2) $Com = \{Com_1\}$.
(3) $Syn = \{(Com_1, Ope_i),\ (Ope_i, Com_1) \mid 1 \leq i \leq k\}$
$\cup\{(Com_1, SF), (SF, FF), (FF, Out)\}$ (see (a) of Figure 4).

From this simpler schema Π_2, we can induce a family of computing devices $I(\Pi_2)$ (with an appropriate interpretation I) that can characterize RE. (Note that Π_2 is nothing but a schema $\Pi_{1,0}$.)

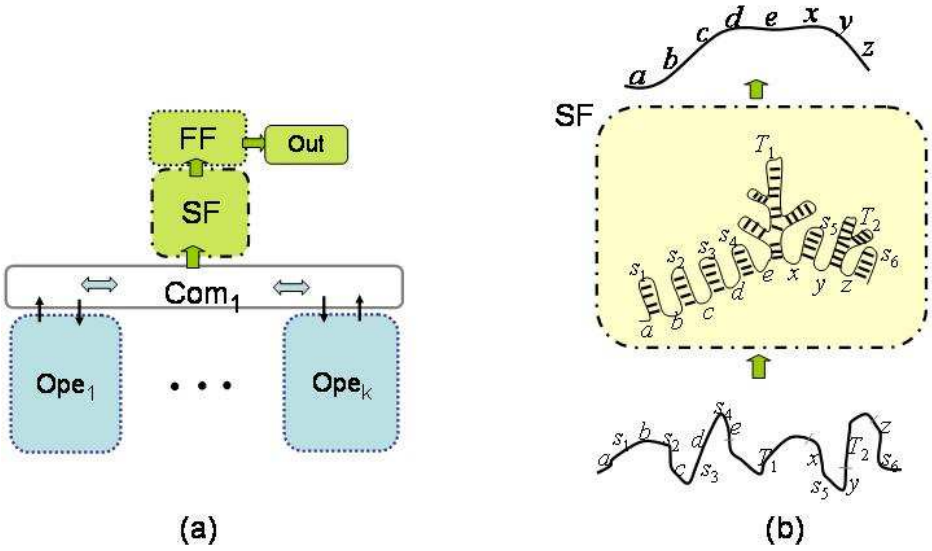

Fig. 4. Membrane Computing Schema: Π_2

Theorem 5.3 *There exists $\mathcal{I}'_G$ such that $RE = \mathcal{LMS}_g(\Pi_2, \mathcal{I}'_G)$.*

Proof sketch. Let L be any language in RE that is generated by a Chomsky type-0 grammar $G = (V, T, S, P)$. Then, consider the following interpretation $I'_G = (\{R_G\}, L_{SF}, L_{FF})$, where

(i) For each $r : u \to v$ in P, construct $R_r = \{\lambda \to vr,\ \lambda \to \overline{u}^R\overline{r}\}$, and let $R_G = \cup_{r\in P} R_r$.
(ii) L_{SF} adopts the Dyck language D_n, where $n = |N \cup T \cup \{r \mid r \in P\}|$.
(iii) L_{FF} is given as T^*, so that only strings in T^* can pass through FF and are sent out to *Out*.

Consider a transition sequence $C_0 = (\{S\}, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L^{(n)}_{out})$. Then, for any $\alpha \in (N \cup T)^*$ and $n \geq 0$, it holds that $\alpha \in L_{1,n}$ iff $S \Longrightarrow^n \alpha$ in G. Thus, we have that $L(G) = L_g(I'_G(\Pi_2))$. (The proof is based on the fact that each recursively enumerable language L can be represented in the form $L = h(L' \cap D_k))$, where L' is an insertion language, h is a projection and D_k is a Dyck language. (Theorem 3.1 in [13]). In order to understand the idea of the proof, it would be helpful to

note that R_G in Com_1 generates the insertion language L', while a pair of SF and FF plays the same role as a pair of D_k and h.) □

6 Concluding Remarks

In this paper we have introduced the notion of a membrane computing schema and showed that several known computing models with the universal computability can be reformulated in a uniform manner in terms of the framework of the schema together with its interpretation. A similar idea in the context of grammar schema has been proposed and discussed in [1, 5] to prove the computational completeness of new type of P systems based on the framework of the random context grammars for both string and multiset languages. (Note that the definition of random context in those papers is not on a string to be rewritten but on the applicability of rules to rewrite, different from the standard notion.) As for the communication by sending objects and the use of filtering function, there are several papers that have been devoted to studying the computational powers of communicating distributed H systems (e.g., [3, 11]), and of the hybrid networks of evolutionary processors (e.g., [7, 8]).

Table 1 summarizes the results we have obtained. From the table, one can have a unified view of a variety of computing models based on *string rewriting*. For example, it is seen that there exists a trade-off between the complexity of network structure in the schema and the complexity of the filtering SF.

More specifically, for new terminologies, L is a *star language* iff $L = F^*$ for some finite set F. Further, L is an *occurrence checking language* iff $L = V^*FV^*$ for some finite set F. Then, it should be noted that
(i) L_G is a finite union of occurrence checking languages,
(ii) L_s and L_m are star languages,
(iii) L_{r_i} is a finite intersection of occurrence checking languages and their complements,
(iv) L_{m_i} is the complement of an occurrence checking language.

Since Π_0 (or Π_1) is more complex than Π_2, one may see a trade-off between the complexity of the schema and that of SF, telling that L_G for single (or L_{mat} for multiple) hairpin checking is simpler than D_k for nested hairpin checking. This kind of trade-off can also be seen in complexity between a series of schemas $(\Pi_1, \Pi_{1.5}, \Pi_2)$ and the corresponding SFs (L_m, L_{mat}, D_k).

It should be remarked that if we start with a programmed context-free grammar G_{pr} ([4]), then we have the result that $RE = \mathcal{LMS}_g(\Pi_1, \mathcal{I}_{G_{pr}})$, where an interpretation $I_{G_{pr}}$ consists of L_m (for SF), T^* (for FF) and L_{m_i} (for FF_i), which suggests that programmed context-free grammars can be regarded as hybrid systems between matrix grammars with appearance checking and random context grammars.

Table 1

	Schema	SF	FF	SubFil
Chomsky type-0 Grammar	Π_0	$L_G(\in RG)$ or $L_{mir}(\in LIN)$	T^*	(N.A.)
Turing Machine	Π_0	$L_M(\in RG)$ or $L_{mir}(\in LIN)$	V^*FV^*	(N.A.)
Random Context Grammar	Π_1	$L_m(\in RG)$	T^*	$L_{r_i}(\in RG)$
Matrix$_{ac}$ Grammar	$\Pi_{1.5}$	$L_{mat} = L_s L_m(\in RG)$	T^*	$L_{m_i}(\in RG)$
Chomsky type-0 Grammar	Π_2	$D_k(\in CF)$	T^*	(N.A.)

In this paper we have just made the first step in the new direction towards understanding and characterizing the nature of the Turing computability from the novel viewpoint of *modularity* in the membrane computing schema. There remain many questions for the future works:

- It would be the most interesting to study the relation between the complexity of the language classes and that of SF within a given schema. For instance, we can show that within the schema Π_2, CF can be characterized by star (regular) languages for SF.
- Instead of insertion operations we adopted in this paper, what kind of operations can be considered for the unique operation in the cells *Ope*? What kind of different landscape of the computing mechanisms can be seen from the new schema?

References

1. M. Cavaliere, R. Freund, M. Oswald, D. Sburlan: Multiset random context grammars, checkers, and transducers. In *Fourth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.), Fenix Editora, Sevilla, 2006, Vol. 1, 113–131.
2. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In *Fourth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.), Fenix Editora, Sevilla, 2006, Vol. 1, 169–193.
3. E. Csuhaj-Varju, L. Kari, Gh.Păun: Test tube distributed systems based on splicing. *Computers and AI*, 15, 2-3 (1996), 211–232.
4. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.
5. R. Freund, M. Oswald: Modeling grammar systems by tissue P systems working in the sequential mode. In *Proc. of Grammar Systems Workshop*, Budapest, 2004.
6. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.

7. M. Margenstern, V. Mitrana, M.J. Pérez-Jiménez: Accepting hybrid networks of evolutionary processors. *Lecture Notes in Computer Science*, 3384, Springer-Verlag, Berlin, 2005, 235–246.
8. C. Martin-Vide, V. Mitrana, M.J. Pérez-Jiménez, F. Sancho-Caparrini: Hybrid networks of evolutionary processors. In *Proc. of GECCO, Lecture Notes in Computer Science*, 2723, Springer-Verlag, Berlin, 2003, 401–412.
9. C. Martin-Vide, Gh. Păun, A. Salomaa: Characterizations of recursively enumerable languages by means of insertion grammars. *Theoretical Computer Science*, 205 (1998), 195–205.
10. C. Martin-Vide, Gh. Păun, J. Pazos, A. Rodriguez-Paton: Tissue P systems, *Theoretical Computer Science*, 296 (2003), 295–326.
11. Gh. Păun: Distributed architectures in DNA Computing based on splicing: Limiting the size of components. In *Proc. Conf. Unconventional Models of Computation* (C.S. Calude, J. Casti, M.J. Dinneen, eds.), Springer-Verlag, Berlin, 1998, 323–335.
12. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
13. Gh. Păun, M.J. Pérez-Jiménez, T. Yokomori: Representations and characterizations of languages in Chomsky hierarchy by means of insertion-deletion systems. Submitted. 2007.
14. Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms*. Springer-Verlag, Berlin, 1998.
15. Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graph of restricted forms. *Publicationes Mathematicae Debrecen*, 60 (2002), 635–660.
16. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.