Membrane Computing Schema: A New Approach to Computation Using String Insertions

Mario J. Pérez-Jiménez and Takashi Yokomori

Abstract In this paper, 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. 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 by the 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,

M.J. Pérez-Jiménez (✉)
Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda Reina Mercedes s/n, 41012 Sevilla, Spain
e-mail: marper@us.es

for example, in the context of P system on graph structure [15], of the tissue P systems [11], and of spiking neural P systems [3, 7].

On the other hand in DNA computing theory, a string generating device called insertion-deletion system has been proposed and investigated from the unique viewpoint of non-rewriting nature in generating string objects [10, 14]. 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. More specifically, let M be a given computing device (grammar or machine) based on string manipulation. Then by a membrane computing schema Π, we represent the *core* structure of M in question. At the same time, we also consider an interpretation I to Π which specifies the *details* of M. In this manner, we are able to 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 by 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 over 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* [10] 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* over V 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 $\mapsto$. An *insertion language* generated by γ is defined as follows: $L(\gamma) = \{w \in V^* \mid s \mapsto^* w, s \in A\}$. 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 non-terminal 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. A matrix of type 3 is called *appearance checking matrix rule*.

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 [5].)

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 $u, v \in (N \cup T)^*$, $(A \to x, Q, R) \in P$ and 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 $RC = RE$ [5].

Note that in what concerns the definitions above for matrix and random context grammars, we only deal with the type 2 (context-free) grammar as the core grammar.

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., [11]).

3.1 Membrane Computing Schema

A *membrane computing schema* of degree $(\mathtt{p}, \mathtt{k}, \mathtt{t})$ is a construct

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

where

- V is a finite alphabet with an involution relation ρ called the *working alphabet*.
- T is a subset of V called the *terminal alphabet*.
- $Com = \{Com_1, \ldots, Com_{\mathtt{p}}\}$ is a finite set with $\mathtt{p}$ elements, called *communication cells*.
- $Ope = \{Ope_1, \ldots, Ope_{\mathtt{k}}\}$ is a finite set with $\mathtt{k}$ elements, called *operation cells*.
- $Fil = \{SF, FF\} \cup \{FF_1, \ldots, FF_{\mathtt{t}}\}$ is a finite set with $(\mathtt{t} + 2)$ elements, called *filtering cells*. (More specifically, SF and FF are called *structured filter* and *final filter*, respectively, and $FF_1, \ldots, FF_{\mathtt{t}}$ are called *sub-filter cells*).
- Syn is a subset of $(Com \times Ope) \cup (Ope \times Com) \cup (Com \times (SubFil \cup \{SF\})) \cup (\{FF\} \times Com) \cup (SubFil \times Ope) \cup \{(SF, FF), (FF, Out)\}$, where $SubFil = \{FF_1, \ldots, FF_{\mathtt{t}}\}$, which determines the tissue membrane structure of Π. (See Fig. 1.)
- i_s $(= Com_1)$ is the distinguished cell (to designate some specific role).
- Out is the *output cell* (for obtaining the outputs).

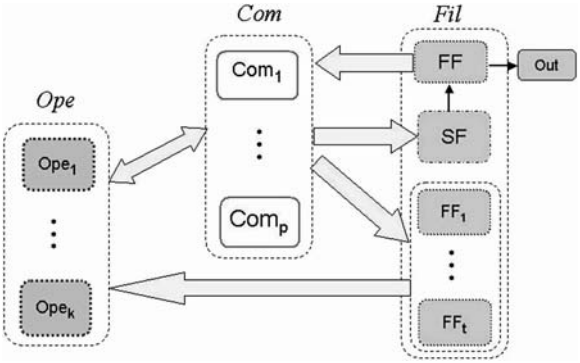


Fig. 1 Modular structure of membrane network in Π

Remark

(1) For $i = 1, \ldots, \mathtt{p}$, 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*.
(2) For $j = 1, \ldots, \mathtt{k}$, each cell Ope_j consists of a finite number of rules $\{\sigma_{j1}, \ldots, \sigma_{js}\}$, where each σ_{ji} is a string insertion operation of the form: $\lambda \to u$, where $u \in V^*$.
(3) *SF* and *FF* are associated with two languages L_{SF} and L_{FF} over V, respectively. Further, for $\ell = 1, \ldots, \mathtt{t}$, each cell FF_ℓ is also associated with a language L_{FF_ℓ}.

3.2 Interpretation of Π

In order to embody a membrane computing schema Π, we need to give further information for Π specifying the initial configuration, each operation in *Ope*, and materializing the filtering cells *SF*, *FF*, and FF_ℓ $(1 \leq \ell \leq \mathtt{t})$. Let us call such a notion an interpretation I to the schema Π which enables us to have an embodied computing model $I(\Pi)$ that is feasible in a usual sense. In what follows, we use the following notation:

Notation For any x in $Com \cup SubFil \cup \{FF\}$, let Syn-from$(x) = \{y \mid (x, y) \in Syn\}$, and for any y in $Ope \cup SupFil \cup \{SF\}$, let Syn-to$(y) = \{x \mid (x, y) \in Syn\}$.

Formally, an *interpretation* I to Π of degree $(\mathtt{p}, \mathtt{k}, \mathtt{t})$ is a construct

$$I = \big(w_0, \{R_1, \ldots, R_\mathtt{k}\}, L_{SF}, L_{FF}, \{L_{FF_\ell} \mid 1 \leq \ell \leq \mathtt{t}\}\big),$$

where

- w_0 is a string in V^* called *the axiom*, where V is the working alphabet of Π.
- R_i specifies a set of insertion operations used in Ope_j (for $j = 1, \ldots, \mathtt{k}$)
- 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:

1. 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, a 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*.
2. *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 in Syn-from(*FF*).
3. Each FF_ℓ receives as input a string u over V. Then a string v filters through if it is in L_{FF_ℓ} and is sent out to all cells in Syn-from(FF_ℓ). Otherwise, it is lost.
4. 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 $I = (w_0, \{R_1, \ldots, R_{\mathtt{k}}\}, L_{SF}, L_{FF})$.

3.3 Transitions and Languages

Given a schema Π of degree $(\mathtt{p}, \mathtt{k}, \mathtt{t})$ and an interpretation I to Π, 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 $(\mathtt{p}+1)$-tuple of languages over V represented by $(L_1, \ldots, L_{\mathtt{p}}, L_{\text{out}})$ constitutes a *configuration* of the system, where each L_i represents the set of all strings in the cell Com_i (for all $i = 1, \ldots, \mathtt{p}$), and L_{out} is the set of strings presented in the output cell (of the system at some time instance).

Let $C_1 = (L_1, \ldots, L_{\mathtt{p}}, L_{\text{out}})$ and $C_2 = (L'_1, \ldots, L'_{\mathtt{p}}, L'_{\text{out}})$ 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 $\ell = 1, \ldots, \mathtt{t}$, consider $L[\ell] = \bigcup_{i=1}^{q} L_{\ell_i}$, where Syn-to($FF_\ell$) $= \{Com_{\ell_1}, \ldots, Com_{\ell_q}\}$. 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 in Syn-from(FF_ℓ). (In the case when $\mathtt{t} = 0$, i.e., *SubFil* $= \emptyset$, this step is skipped.)

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

Suppose that we apply operations $\sigma_{ji} : \lambda \to u_{ji}$ $(1 \le i \le s)$ 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 non-deterministically 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 $L(j) = \bigcup_{m=1}^{d} L_{j_m}$, where Syn-to($Ope_j$) $\cap$ Com = $\{Com_{j_1}, \ldots, Com_{j_d}\}$. Then the total result performed by Ope_j to $L(j)$ is defined as

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

This result is then sent out to all cells in Syn-from(Ope_j) simultaneously.

(2) *Filtering Step*: For each $i = 1, \ldots, \mathfrak{p}$, let $\tilde{L}_i = \bigcup_{n=1}^{r} Ope_{j_n}(L(j_n))$, where Syn-from($Com_i$) = $\{Ope_{j_1}, \ldots, Ope_{j_r}\}$. Further, let $L_e = \bigcup_{m=1}^{g} \tilde{L}_{i_m}$, where Syn-to($SF$) = $\{Com_{i_1}, \ldots, Com_{i_g}\}$.

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 all Com_i in Syn-from(*FF*) or they are all lost if Syn-from(FF) = $\emptyset$.

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

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

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

Remark

(1) Each cell in *Com* not only provides a buffer for storing intermediate results in the computation process but also *transmit* them to the cells specified by *Syn*.
(2) Each cell in *Ope* takes as input a set of strings L and applies insertion operations to L, and sends out the result to the cells specified by *Syn*.
(3) Each filtering cell in *Fil* also takes as input a set of strings L and performs filtering of L in a way previously described, and sends out the result to the cells specified by *Syn*.
(4) 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 (described in (1), (2), and (3) for *Com*, *Ope* and *Fil*, respectively), irrespective of the existence of strings in it.

Let $I = (w_0, \{R_1, \ldots, R_{\mathfrak{k}}\}, L_{SF}, L_{FF}, \{L_{FF_\ell} \mid 1 \le \ell \le \mathfrak{t}\})$ be an interpretation to Π. We write $C_1 \Longrightarrow C_2$ if there is a transition from C_1 to C_2 of $I(\Pi)$.

A configuration $C_0 = (\{w\}, \overbrace{\emptyset, \ldots, \emptyset}^{\mathtt{p}})$ with w in V^* is called the *initial configuration*. (We assume that filtering cells are all *empty* in the initial configuration.)

For any $n \geq 0$, let $C_0 \Longrightarrow^n C_n = (L_{1,n}, \ldots, L_{\mathtt{p},n}, L_{\text{out}}^{(n)})$ be a sequence of n-transitions which we call a *computation with n transitions* from C_0. Then a language $L_{\text{out}}^{(n)}$ is called the *nth output* from C_0.

Now, 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 concern all computations whose results are present in the output cell.

We define the language generated by $I(\Pi)$ as follows:

$$L_g\big(I(\Pi)\big) = \bigcup_{n \geq 0} L_{\text{out}}^{(n)},$$

where $L_{\text{out}}^{(n)}$ is the nth output from $(w_0, \emptyset, \ldots, \emptyset)$ and w_0 is the axiom of I.

[*Accepting Model*] In the case of an accepting model, we make a slight modification on an interpretation I and consider the following $I = (\lambda, \{R_1, \ldots, R_{\mathtt{k}}\}, L_{SF}, L_{FF}, \{L_{FF_\ell} \mid 1 \leq \ell \leq \mathtt{t}\})$ with L_{FF} which functions in such a way that if a string v filters through L_{FF}, then (not v but) a special symbol "*Yes*" is sent to *Out*.

Let w be an input string to be recognized. Then w is accepted iff the result *Yes* is present in the output cell after a certain number of transitions from $C_0 = (\{w\}, \emptyset, \ldots, \emptyset)$. Thus, we define the language accepted by $I(\Pi)$ as follows:

$$L_a\big(I(\Pi)\big) = \big\{w \in T^* \mid \mathit{Yes} \in L_{\text{out}}^{(n)}\text{: the } n\text{th output from } (w, \emptyset, \ldots, \emptyset) \text{ for some } n \geq 0\big\}.$$

Finally, for a class of schemes $\mathcal{S} = \{\Pi \text{ of degree } (\mathtt{p}, \mathtt{k}, \mathtt{t}) \mid \mathtt{p}, \mathtt{k}, \mathtt{t} \geq 0\}$ and for a class of interpretations $\mathcal{I}$, we denote by $\mathcal{LMS}_x(\mathcal{S}, \mathcal{I})$ the family of all languages $L_x(I(\Pi))$ specified by those systems as above, where Π is in $\mathcal{S}$ and I is in $\mathcal{I}$. That is,

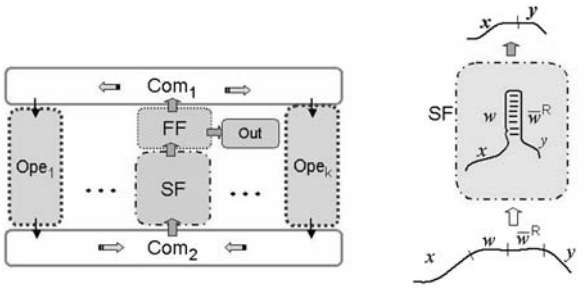
$$\mathcal{LMS}_x(\mathcal{S}, \mathcal{I}) = \big\{L_x(I(\Pi)) \mid I \in \mathcal{I} \text{ is an interpretation to } \Pi \in \mathcal{S}\big\}$$

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 of the universal computability by finding an appropriate interpretation. In what follows, we will show that such a universal computability can be realized by much simpler schemes together with appropriate interpretations of moderately simple filtering cells.

Fig. 2 Membrane computing schema: $\Pi_{0,k}$



(a) Organization of $\Pi_{0,k}$ (b) Functioning of SF

First, we consider the following simple membrane computing schema of degree $(2, \mathtt{k}, 0)$:

$$\Pi_{0,\mathtt{k}} = (V, T, Com, Ope, Fil, Syn, i_s, Out),$$

where:

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

Let $\Pi_0 = \bigcup_{\mathtt{k}\geq 0}\ \Pi_{0,\mathtt{k}}$.

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

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

Proof We prove only the inclusion $\subseteq$. (The opposite inclusion is due to 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 $I_G = (S, R_G, L_{SF}, L_{FF})$ to $\Pi_{0,\mathtt{k}}$, 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 $R_G = \{R_r \mid r \in P\}$. (Note that the cardinality of R_G gives $\mathtt{k}$ of $\Pi_{0,\mathtt{k}}$.)
(ii) • L_{SF} is given as the following language: $L_{\mathrm{mir}} = \{xw\overline{w}^R y \mid x, y, w \in V^*\}$. 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 single hairpin formed by $w\overline{w}$, and then it removes the portion of a hairpin from the structure and sends out the rest part of the string to FF (see (b) of Fig. 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.

For any $n \geq 1$, let $C_0 = (\{S\}, \emptyset, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{2,n}, L_{\text{out}}^{(n)})$ be a computation with n transitions in $I_G(\Pi_{0,\text{k}})$. 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

(a) α is in $L_{1,(n-1)}$,
(b) $\beta' = xvru\overline{u}^R\overline{r}y$ is in SF, and
(c) 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 $C_n = (L_{1,n}, L_{2,n}, L_{\text{out}}^{(n)})$. Otherwise, β is sent to $L_{\text{out}}^{(n)}$. That is, if β is in $L(G)$, then we have β is in $L_g(I_G(\Pi_{0,\text{k}}))$.

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 exists a unique rule $r : u \to v$ 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 $\alpha' = \alpha_1' vru\overline{u}^R\overline{r}\alpha_2$ 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. (For more details, consult discussion in Sect. 3 of [13].)

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 $L_g(I_G(\Pi_{0,\text{k}}))$, then it is also in $L(G)$. Thus, we have $L(G) = L_g(I_G(\Pi_{0,\text{k}}))$. Clearly, considering for $\mathcal{I}_G$ the class of interpretations I_G for all type-0 grammars G, we complete the proof. □

Note The language L_{mir} used for L_{SF} can be replaced with simpler (regular) language $L_G = \bigcup_{r \in P} V^*\{ru\overline{u}^R\overline{r}\}V^*$, where $r : u \to v \in P$ and P is the set of productions of G. However, we choose L_{mir} here because of its independence of G.

Theorem 4.2 *There exists a class of interpretations $\mathcal{I}_M$ 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 $L(M)$ be any language in RE accepted by $M = (Q, T, U, \delta, p_0, F)$, where $\delta \subseteq Q \times U \times Q \times U \times \{L, R\}$. For $(p, a, q, c, i) \in \delta$, we write $(p, a) \to (q, c, i) \in \delta$. 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 $(x, y \in U^*, a \in U)$, M is in the state $p(\in Q)$ and the tape head is on a, and $\#(\#')$ is the left-boundary (right-boundary).

Given an input $w(\in T^*)$, M starts computing w from the state p_0, represented by an ID: $\#p_0w\#'$. (We may assume that the tape content is one-way extendible to the right.) 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 $i = R$),
- $\#xbpay\#' \Longrightarrow \#xqbcy\#'$ (if $i = L$),
- $\#xbp\#' \Longrightarrow \#xbcq\#'$ (if $i = R$, and $y = \lambda$),
- $\#xbp\#' \Longrightarrow \#xqbc\#'$ (if $i = L$, and $y = \lambda$).

In each case, one can consider a rewriting rule, for example, a rule $pa \to cq$ for the first case, or a rule $p\#' \to cq\#'$ for the third case. Let P_M be the set of all rewriting rules obtained from δ in the manner mentioned above. Further, we define $L(M)$ as $\{\#p_0w\#' \mid \#p_0w\#' \Longrightarrow^* \#xqy\#'$ for some $q \in F, x, y \in U^*\}$.

We now consider the following interpretation $I_M = (R_M, L_{SF}, L_{FF})$:

(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 $R_M = \{R_r \mid r \in P_M\}$, where the cardinality of R_M gives $\mathtt{k}$ in the degree of $\Pi_{0,\mathtt{k}}$.
(ii) L_{SF} is given in the same way as in I_G in the proof for Theorem 4.1.
(iii) L_{FF} is given as V^*FV^*, where $V = U \cup \{\#, \#'\} \cup \{r \mid r \in P_M\}$.

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 $Yes \in L_{\text{out}}^{(n)}$ such that $C_0 = (\{\#p_0w\#'\}, \emptyset, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{2,n}, L_{\text{out}}^{(n)})$. Thus, we have $L(M) = L_a(I_M(\Pi_{0,\mathtt{k}}))$. Let $\mathcal{I}_M$ be the class of interpretations I_M for all Turing machines M, which completes the proof. □

5 Further Results on Some Variants of Membrane Schema

We now introduce two classes of membrane computing schemes $\Pi_{1,5}$ and Π_1 which are variants of Π_0. With an appropriate class of interpretations $\mathcal{I}$, both are able to induce a family of computing devices that can again characterize *RE*.

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

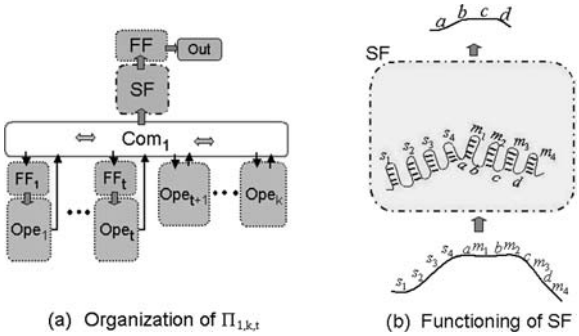
$$\Pi_{1,\mathtt{k},\mathtt{t}} = (V, T, Com, Ope, Fil, Syn, i_s, Out),$$

where:

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

(Note: In (3) and (4) above, $\mathtt{t}$ can take any integer in $\{0, 1, \ldots, \mathtt{k}\}$, and when $\mathtt{t} = 0$, it means the corresponding set is empty. $\Pi_{1,\mathtt{k},\mathtt{t}}$ is a membrane computing schema of degree $(1, \mathtt{k}, \mathtt{t})$ with $\mathtt{t} \leq \mathtt{k}$.)

Fig. 3 Membrane computing schema: $\Pi_{1,\mathtt{k},\mathtt{t}}$



Notation We now consider the following two classes of schemes:

$$\Pi_{1.5} = \bigcup_{\mathtt{k}>\mathtt{t}\geq 0} \Pi_{1,\mathtt{k},\mathtt{t}},$$

$$\Pi_{1} = \bigcup_{\mathtt{k}\geq 0} \Pi_{1,\mathtt{k},\mathtt{k}}.$$

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

Proof sketch We use an argument similar to the one in 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_m is in the binary normal form (see Sect. 2).

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

(i) XA is the string in $(S \to XA)$ of G_m.

(ii)-1: $\mathtt{k}$ is the cardinality of M and $\mathtt{t}$ is the number of appearance checking matrix rules in M. For each appearance checking rule $m_i : (X \to Y, A \to \#)$ $(1 \leq i \leq \mathtt{t})$, construct $R_{m_i} = \{\lambda \to Y s_{m_i},\ \lambda \to \overline{X}\overline{s}_{m_i}\}$.

(ii)-2: For other rules $m_j : (X \to Y, A \to x)$ in M (where $Y \in N_1 \cup \{\lambda\}, x \in T \cup N_2 \cup N_2^2 \cup \{\lambda\}$; $\mathtt{t}+1 \leq j \leq \mathtt{k}$), 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 $R_{G_m} = \{R_{m_1}, \ldots, R_{m_\mathtt{k}}\}$.

(iii) • L_{SF} is given as the following regular language: $L_{\mathrm{mat}} = L_s L_r$, where

$$L_s = \left\{ s_{m_i} X \overline{X} \overline{s}_{m_i} \mid m_i : (X \to Y, A \to y) \in M,\ 1 \leq i \leq \mathtt{k} \right\}^*, \quad \text{and}$$

$$L_r = \Big(T \cup N_2 \cup \{ r_{m_j} A \overline{A} \overline{r}_{m_j} \mid m_j : (X \to Y, A \to x) \in M,\ \mathtt{t}+1 \leq j \leq \mathtt{k} \} \Big)^*.$$

- L_{FF_i} is given as follows: For each appearance checking rule $m_i : (X \to Y, A \to \#)$ $(1 \leq i \leq \mathtt{t})$, 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 sends 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 $C_0 = (\{XA\}, \emptyset)$ and consider a transition sequence: $C_0 \Longrightarrow^n C_n = (L_{1,n}, L_{\text{out}}^{(n)})$. 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 $L(G_m) = L_g(I_{G_m}(\Pi_{1,\mathtt{k},\mathtt{t}}))$. Let $\mathcal{I}_{G_m}$ be the class of interpretations I_{G_m} for all matrix grammars G_m, which completes the proof. □

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

Proof sketch We use the same argument as the one in Theorem 5.1, but start with a random context grammar $G_r = (N, T, S, P)$ generating arbitrary recursively enumerable language L (see Sect. 2). Then consider the following interpretation $I_{G_r} = (S, R_{G_r}, L_{SF}, L_{FF}, \{L_{FF_i} \mid 1 \leq i \leq \mathtt{k}\})$, where

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

(ii)
- L_{SF} is defined by the regular language:

$$L_m = \left(T \cup \{r A \overline{A r} \mid r : (A \to x, Q, R) \in P\}\right)^*.$$

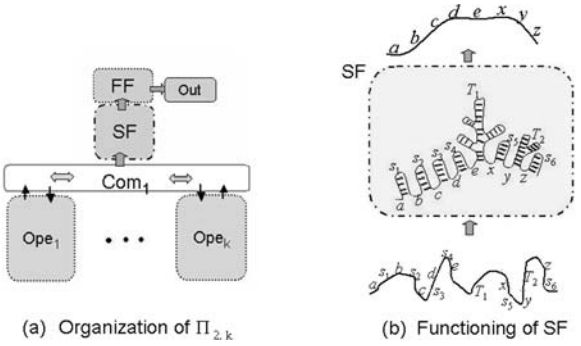
- L_{FF_i} is given as follows: For each rule $r_i : (A \to x, Q, R)$, let

$$\begin{aligned} L_{r_i} = {} & (V \cup \overline{V})^* \{A\} (V \cup \overline{V})^* \\ & \cap \bigcap_{X \in Q} (V \cup \overline{V})^* \{X\} (V \cup \overline{V})^* \\ & \cap \bigcap_{X \in R} \left((V \cup \overline{V})^* - (V \cup \overline{V})^* \{X\} (V \cup \overline{V})^*\right), \end{aligned}$$

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 sends 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 $C_0 = (\{S\}, \emptyset)$ (note that S is the starting symbol of G_r) and consider a transition sequence: $C_0 \Longrightarrow^n C_n = (L_{1,n}, L_{\text{out}}^{(n)})$. Then for any $\alpha \in (N \cup T)^*$ and $n \geq 0$,

Fig. 4 Membrane computing schema: $\Pi_{2,\mathrm{k}}$



it holds that $\alpha \in L_{1,n}$ iff $S \Longrightarrow^n \alpha$ in G_r. Thus, we have $L(G_r) = L_g(I_{G_r}(\Pi_{1,\mathrm{k},\mathrm{k}}))$. Letting $\mathcal{I}_{G_r}$ be the class of interpretations I_{G_r} for all random context grammars G_r, we complete the proof. □

We present yet another class of membrane computing schemes Π_2 which is simpler than Π_0 but still able to provide the universal computability of the models induced from the schema with appropriate interpretations, but at the sacrifice of increase of the structural complexity in the filtering function *SF*.

A membrane computing schema $\Pi_{2,\mathrm{k}}$ is given as follows:

$$\Pi_{2,\mathrm{k}} = (V, T, Com, Ope, Fil, Syn, i_s, Out),$$

where

(1) V, T, Ope, Fil, i_s and Out are the same as in $\Pi_{0,\mathrm{k}}$.
(2) $Com = \{Com_1\}$.
(3) $Syn = \{(Com_1, Ope_i),\ (Ope_i, Com_1) \mid 1 \leq i \leq \mathrm{k}\} \cup \{(Com_1, SF), (SF, FF), (FF, Out)\}$ (see (a) of Fig. 4).

Let $\Pi_2 = \bigcup_{\mathrm{k} \geq 0} \Pi_{2,\mathrm{k}}$.

From this simpler class of schemes Π_2, we can induce a family of computing devices $\mathcal{I}(\Pi_2)$ (with an appropriate class of interpretations $\mathcal{I}$) that can characterize *RE*.

Theorem 5.3 *There exists a class of interpretations* $\mathcal{I}'_G$ *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 $I'_G = (S, \{R_G\}, L_{SF}, L_{FF})$, 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 $R_G = \bigcup_{r \in P} R_r$. The cardinality of R_G gives k in the degree of $\Pi_{2,\mathrm{k}}$.
(ii) L_{SF} adopts the Dyck language D over the alphabet $N \cup T \cup \{r \mid r \in P\}$.
(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: $C_0 = (\{S\}, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{\text{out}}^{(n)})$. 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, it holds that $L(G) = L_g(I'_G(\Pi_{2,k}))$. In order to complete the proof, we have only to consider for $\mathcal{I}'_G$ the class of interpretations I'_G for all type-0 grammars G. (The proof is based on the following result that each recursively enumerable language L can be represented in the form $L = h(L' \cap D))$, where L' is an insertion language, h is a projection and D 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 and h, respectively.) □

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 [2, 6] to prove the computational completeness of new type of P systems based on the framework of the random context grammars for both string and multi-set 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 re-write, 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., [4, 12]), and of the hybrid networks of evolutionary processors (e.g., [8, 9]). Among others, the notion of observers in G/O systems proposed in [1] may be of special interests in that it seems to have a close relation to the filtering mechanism (for structured or string objects) of the membrane computing schema introduced in this paper.

Table 1 summarizes the results we have obtained. From the table, one can have a unified view of the existing various models of computation based on *string rewrit-*

Table 1

Computing model	Schema	SF	FF	$SubFil$
Chomsky type-0 grammar	Π_0	$L_G(\in RG)$ or $L_{\text{mir}}(\in LIN)$	T^*	(N.A.)
Turing machine	Π_0	$L_G(\in RG)$ or $L_{\text{mir}}(\in LIN)$	V^*FV^*	(N.A.)
Random context grammar	Π_1	$L_m(\in RG)$	T^*	$L_{r_i}(\in RG)$
Matrix$_{ac}$ grammar	$\Pi_{1.5}$	$L_{\text{mat}} = L_s L_r(\in RG)$	T^*	$L_{m_i}(\in RG)$
Chomsky type-0 grammar	Π_2	$D(\in CF)$	T^*	(N.A.)

ing. 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 in Table 1:

(i) L_G is a finite union of occurrence checking languages,
(ii) L_s, L_r 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 a Dyck language D for nested hairpin checking. This kind of trade-off can also be seen in complexity between a series of schemes (Π_1, $\Pi_{1.5}$, Π_2) and the corresponding SFs(L_m, L_{mat}, D).

In this paper, we have just made the first step in the new direction toward understanding and characterizing the nature of the Turing computability from the novel viewpoint of *modularity* in the membrane computing schema. There seems to remain many left 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 mechanism can be seen from the new schema?

Acknowledgements The first author wishes acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, co-financed by FEDER funds and the support of the project of excellence TIC-581 of the Junta de Andalucía. The second author gratefully acknowledges the support of the Grant of Faculty Development Award, Waseda University and Grant-in-Aid for Scientific Research on Priority Area No. 14085202, Ministry of Education, Culture, Sports, Science, and Technology, Japan.

References

1. Cavaliere M, Leupold P (2004) Evolution and observation—a non-standard way to generate formal languages. Theor Comput Sci 321:233–248
2. Cavaliere M, Freund R, Oswald M, Sburlan D (2007) Multiset random context grammars, checkers, and transducers. Theor Comput Sci 372:136–151
3. Chen H, Freund R, Ionescu M, Păun Gh, Pérez-Jiménez MJ (2006) On string languages generated by spiking neural P systems. In: Gutierrez-Naranjo MA, Păun Gh, Riscos-Nunez A, Romero-Campero FJ (eds) Reports on fourth brainstorming week on membrane computing, vol 1, pp 169–193

4. Csuhaj-Varju E, Kari L, Păun Gh (1996) Test tube distributed systems based on splicing. Comput Artif Intell 15(2–3):211–232
5. Dassow J, Păun Gh (1989) Regulated rewriting in formal language theory. Springer, Berlin
6. Freund R, Oswald M (2004) Modeling grammar systems by tissue P systems working in the sequential mode. In: Proceedings of grammar systems workshop, Budapest
7. Ionescu M, Păun Gh, Yokomori T (2006) Spiking neural P systems. Fund Inform 71(2–3):279–308
8. Margenstern M, Mitrana V, Pérez-Jiménez M (2005) Accepting hybrid networks of evolutionary processors. In: Lecture notes in computer science, vol 3384. Springer, Berlin, pp 235–246
9. Martin-Vide C, Mitrana V, Pérez-Jiménez M, Sancho-Caparrini F (2003) Hybrid networks of evolutionary processors. In: Proceedings of GECCO. Lecture notes in computer science, vol 2723. Springer, Berlin, pp 401–412
10. Martin-Vide C, Păun Gh, Salomaa A (1998) Characterizations of recursively enumerable languages by means of insertion grammars. Theor Comput Sci 205:195–205
11. Martin-Vide C, Păun Gh, Pazos J, Rodriguez-Paton A (2003) Tissue P systems. Theor Comput Sci 296:295–326
12. Păun Gh (1998) Distributed architectures in DNA computing based on splicing: limiting the size of components. In: Calude CS, Casti J, Dinneen MJ (eds) Unconventional models of computation. Springer, Berlin, pp 323–335
13. Păun Gh, Pérez-Jiménez MJ, Yokomori T (2007) Representations and characterizations of languages in Chomsky hierarchy by means of insertion-deletion systems. In: Proceedings of DCFS2007, High Tatras, Slovakia, pp. 129–140. Also, to appear in Int J Found Comput Sci, 2008
14. Păun Gh, Rozenberg G, Salomaa A (1998) DNA computing. Springer, Berlin
15. Păun Gh, Sakakibara Y, Yokomori T (2002) P-systems on graph of restricted forms. Publ Math Debrecen 60:635–660
16. Rozenberg G, Salomaa A (1997) Handbook of formal languages. Springer, Berlin