Journal of Computer and System Sciences **61**, 108–143 (2000)
doi:10.1006/jcss.1999.1693, available online at http://www.idealibrary.com on IDEAL®

Computing with Membranes

Gheorghe Păun

Institute of Mathematics of the Romanian Academy, P.O. Box 1-764, 70700 Bucharest, Romania
E-mail: gpaun@imar.ro

Received January 13, 1999; revised June 30, 1999

We introduce a new computability model, of a distributed parallel type, based on the notion of a *membrane structure*. Such a structure consists of several cell-like membranes, recurrently placed inside a unique "skin" membrane. A plane representation is a Venn diagram without intersected sets and with a unique superset. In the regions delimited by the membranes there are placed *objects*. These objects are assumed to evolve: each object can be transformed in other objects, can pass through a membrane, or can dissolve the membrane in which it is placed. A priority relation between evolution rules can be considered. The evolution is done in parallel for all objects able to evolve. In this way, we obtain a computing device (we call it a *P system*): start with a certain number of objects in a certain membrane and let the system evolve; if it will halt (no object can further evolve), then the computation is finished, with the result given as the number of objects in a specified membrane. If the development of the system goes forever, then the computation fails to have an output. We prove that the P systems with the possibility of objects to cooperate characterize the recursively enumerable sets of natural numbers; moreover, systems with only two membranes suffice. In fact, we do not need cooperating rules, but we only use *catalysts*, specified objects which are present in the rules but are not modified by the rule application. One catalyst suffices. A variant is also considered, with the objects being strings over a given alphabet. The evolution rules are now based on string transformations. We investigate the case when either the rewriting operation from Chomsky grammars (with respect to context-free productions) or the splicing operation from H systems investigated in the DNA computing is used. In both cases, characterizations of recursively enumerable languages are obtained by very simple P systems: with three membranes in the rewriting case and four in the splicing case. Several open problems and directions for further research are formulated.

Key Words: membrane structure; P system; recursively enumerable set; matrix grammar; splicing; natural computing.

0022-0000/00 $35.00

1. INTRODUCTION

The present paper can be considered a contribution to what has recently been given the generic name of *natural computing*, a field of research which tries to imitate nature in the way it computes, by learning new computing models and computing paradigms experimented for billions of years by nature and by implementing them in computations done *in vitro* (or, in many cases, *in info*, in symbolic terms only, maybe implemented in silicon media). Neural networks, genetic algorithms, and DNA computing are three areas of natural computing that are already well established and, in the first two cases, are also proved with practical usefulness.

However, nature computes not only at the neural or genetic level, but also at the cellular level. More generally, any nontrivial biological system is a hierarchical construct where an intricate flow of materials and information takes place and which can be interpreted as a computing process.

At a more specific level with respect to the computing models we are going to define, what is important to us is the fact that the parts of a biological system are well delimited by various types of *membranes*, in the broad sense of the term, starting from the cell membrane, going to the skin of organisms, and ending with more or less virtual membranes which delimit, for instance, parts of an ecosystem. In very practical terms, in biology and chemistry one knows membranes which keep together certain chemicals and leave other chemicals to pass in a selective manner, sometimes only in one direction. Membranes delimiting subsystems of a symbol manipulating system are also considered in the logical framework to the so-called metabolic systems, as defined in [19], or in the so-called chemical abstract machine, introduced in [4].

Another incentive of our work comes from distributed computing, where again rather different but well-delimitd computing units coexist and are hierarchically arranged in complex systems. It is a long way from single small processors to the World Wide Web, throughout which we can see the aspects mentioned above. The grammar systems theory mirrors in mathematical terms such distributed symbol processing systems (see, e.g, [7] and, for recent developments, [25]) and will have some resemblance to (some of) the computing models we consider here.

Starting from these observations, we first consider the notion of a *membrane structure* as a mathematical counterpart of hierarchical architectures composed of membranes recurrently distinguished in a given main membrane. We will represent such a structure as a Venn diagram, with all the considered sets being subsets of a unique set and not allowed to be intersected (two sets are either where one is the subset of the other or they are disjoint).

The next step is to consider the notion of a *super-cell*, which is nothing more than a membrane structure with certain *objects* placed in the regions delimited by the membranes. The objects are identified by their names (mathematically, by symbols from a given alphabet). Because several copies of the same object can appear in the same region, we work with multisets, sets with multiplicities associated with their elements.

If the objects of a super-cell are able to evolve, then we obtain a computing device. We call it a *P system.*

Thus, a P system is a membrane structure with objects in its membranes, with specified evolution rules for objects, and with given input–output prescriptions. Any object, alone or together with one more object, evolves, can be transformed into other objects, can pass through one membrane, and can dissolve the membrane into which it is placed. All objects evolve at the same time, in parallel; in turn, all membranes are active in parallel. The evolution rules are hierarchized by a priority relation, given in the form of a partial order relation, the rule with the highest priority among the applicable rules is always the one actually applied. If the objects evolve alone, then the system is said to be noncooperative; if there are rules which specify the evolution of several objects at the same time, then the system is cooperative; an intermediate case is that where there are certain objects (we call them *catalysts*), specified in advance, which do not evolve alone, but appear together with other objects in evolution rules and they are not modified by the use of the rules.

The systems of this basic type are called *transition P systems*, in order to distinguish this variant from other variants considered later.

It is somewhat surprising that with only these simple ingredients (and with designated input and output membranes), the cooperating P systems and the systems with catalysts have computational completeness, they can characterize the recursively enumerable sets of natural numbers. Moreover, very simple membrane structures are enough: two membranes suffice. The input and the output of a computation are codified in the number of objects placed in certain input and output membranes, respectively.

An attractive feature of P systems is their intrinsic parallelism. All objects having access to a rule should use that rule (with the restriction imposed by the priority relation). Moreover, all membranes work in parallel. The effect of this two-level parallelism on the complexity of computations done by P systems is not yet clarified.

The super-cell structure can be used as a support for a computing device based on any type of objects and any type of evolving rules associated with them. In the case above, the objects were single symbols (finitely many for each concrete system, taken from a denumerable alphabet). We can also take strings as objects. In this way, we can use an infinite set of objects, which can evolve in many ways, defined by string processing rules: rewriting, point mutations, insertion and deletion, etc. We consider here only two cases, when the strings evolve by rewriting (using context-free rules) and by splicing, the operation defined in [15] as a model of the recombinant behavior of DNA molecules under the influence of restriction enzymes. It is known that the splicing operation is powerful—see [22]. This observation is confirmed here: P systems based on splicing characterize the family of recursively enumerable languages. Moreover, very simple systems are enough: we need only four membranes, arranged in a two-level structure. Splicing P systems with two membranes can generate nonregular languages, while three membranes are sufficient to generate non-context-free languages.

A characterization of recursively enumerable languages is also obtained in the case of P systems based on rewriting. The proof uses the characterization of recursively enumerable languages by means of matrix grammars with appearance checking. The number of used membranes is still smaller than in the case of splicing: three.

We have mentioned that the membrane structure used in our systems is of the same type as that used in the chemical abstract machine of [4], while the basic idea of evolution rules as multiset transformations also appears in the so-called Γ-systems introduced in [3]. However, here we use additional ingredients: dissolving a membrane, specifying a target for an object produced by an evolution rule, priorities and input and output membranes. Moreover, the method of using the considered systems is totally different here from the mentioned papers: a P system is here a computing device; we are not looking (only) for a correct development of the process, but we are also interested in the relation between an input and an output. In some sense, our approach is much more classic because it pertains to natural computing approached in automata and formal grammars theory style. This makes it both *possible* to consider our devices as computing machineries and not only process models, and *necessary* to compare their power to the known hierarchies of languages, such as the Chomsky hierarchy and the Lindemnayer hierarchy.

2. SOME GENERAL PREREQUISITES

We here specify a few elementary notions and notations which will be useful in the subsequent sections.

We denote by $\mathbf{N}$ the set of natural numbers.

Let U be an arbitrary set. A *multiset* (over U) is a mapping $M: U \to \mathbf{N}$; $M(a)$, for $a \in U$, is the *multiplicity of a in the multiset M*. We indicate this fact also in the form $(a, M(a))$. (Of course, the multiplicity of each object with respect to any multiset is finite.) The *support* of a multiset M is the set $supp(M) = \{a \in U \mid M(a) > 0\}$. A multiset M is empty when its support is empty (it is then denoted by $\varnothing$).

Let $M_1, M_2: U \to \mathbf{N}$ be two multisets. We say that M_1 is included in M_2 iff $M_1(a) \leqslant M_2(a)$, for all $a \in U$. The union of M_1 and M_2 is the multiset $M_1 \cup M_2: U \to \mathbf{N}$ defined by $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$, for all $a \in U$. The difference $M_1 - M_2$ is here defined only when M_2 is included in M_1 and it is the multiset $M_1 - M_2: U \to \mathbf{N}$ given by $(M_1 - M_2)(a) = M_1(a) - M_2(a)$, for all $u \in U$.

A multiset M of finite support, $\{(a_1, M(a_1)), (a_2, M(a_2)), ..., (a_n, M(a_n))\}$, can be also represented by a string $a_1^{M(a_1)} a_2^{M(a_2)} \cdots a_n^{M(a_n)}$ and all permutations of this string precisely identify the objects in the support of M and their multiplicities. We will frequently use below this more compact representation of multisets of a finite support.

An *alphabet* is a finite nonempty set of abstract *symbols*. Given an alphabet V, we denote by V^* the sets of all finite strings of elements in V, including the empty string, λ. (Thus, V^* is the free monoid generated by V with the operation of concatenation and the identity λ.) The length of a string $x \in V^*$ is denoted by $|x|$, and $|x|_a$ is the number of occurrences in x of $a \in V$. A set of strings (over an alphabet V) is called a *language* (over V). Clearly, every string $x \in V^*$ describes a multiset over V, denoted by $m(x)$ and defined by $m(x) = \{(a, |x|_a) \mid a \in V\}$.

For elements of formal language theory we will use here we refer to [34]. Details about L systems, regulated rewriting, grammar systems, and DNA computing can be found in [33, 11, 7, and 22], respectively. Some notions and notations which will be used only locally will be introduced when necessary.

3. MEMBRANE STRUCTURES

We now introduce the basic structural ingredient of the computing devices we will define later: membrane structures.

Let us consider first the language MS over the alphabet $\{[,]\}$, whose strings are recurrently defined as follows:

1. $[\ ] \in MS$;
2. if $\mu_1, ..., \mu_n \in MS, n \geqslant 1$, then $[\mu_1 \cdots \mu_n] \in MS$;
3. nothing else is in MS.

Consider now the following relation over the elements of MS: $x \sim y$ if and only if we can write the two strings in the form $x = \mu_1\mu_2\mu_3\mu_4$, $y = \mu_1\mu_3\mu_2\mu_4$, for $\mu_1\mu_4 \in MS$ and $\mu_2, \mu_3 \in MS$ (two pairs of neighboring parentheses placed at the same level are interchanged, together with their contents). We also denote by $\sim$ the reflexive and transitive closure of the relation $\sim$. This is clearly an equivalence relation. We denote by $\overline{MS}$ the set of equivalence classes of MS with respect to this relation. The elements of $\overline{MS}$ are called *membrane structures*.

Each matching pair of parentheses $[,]$ appearing in a membrane structure is called a *membrane*. The number of membranes in a membrane structure μ is called the *degree* of μ and denoted by $deg(\mu)$. The external membrane of a membrane structure μ is called the *skin* membrane of μ. A membrane which appears in $\mu \in \overline{MS}$ in the form $[\ ]$ (where no other membrane appears inside the two parentheses) is called an *elementary* membrane.

The *depth* of a membrane structure μ, denoted by $dep(\mu)$, is defined recurrently as follows:

1. if $\mu = [\ ]$, then $dep(\mu) = 1$;
2. if $x = [\mu_1 ... \mu_n]$, for some $\mu_1, ..., \mu_n \in MS$, then $dep(\mu) = \max\{dep(\mu_i) \mid 1 \leqslant i \leqslant n\} + 1$.

A membrane structure can be represented in a natural way as a Venn diagram. This makes clear the fact that the order of neighboring membrane structures placed at the same level in a larger membrane structure is irrelevant; what matters is the topological structure, the relationships between membranes. In the subsequent sections we will make an extensive use of such a representation.

The Venn representation of a membrane structure μ also makes clear the notion of a *region* in μ: any closed space delimited by membranes is called a region of μ. It is clear that a membrane structure of degree n contains n regions, one associated with each membrane.

4. SUPER-CELLS

We now make one more step toward the definition of a computing device by adding objects to a membrane structure.

Let U be a denumerable set whose elements are called *objects*.

Consider a membrane structure μ of degree n, $n \geqslant 1$, with the membranes labeled in a one-to-one manner, for instance, with the numbers from 1 to n. In this way, also the regions of μ are identified by the numbers from 1 to n. If a multiset M_i: $U \to \mathbf{N}$ is associated with each region i of μ, $1 \leqslant i \leqslant n$, then we say that we have a *super-cell*.

Any multiset M_i mentioned above can be empty. In particular, all of them can be empty; that is, any membrane structure is a super-cell. On the other hand, each individual object can appear in several regions, in several copies in each of them.

Several notions defined for membrane structures are extended in the natural way to super-cells: degree, depth, region, etc.

The multiset corresponding to a region of a super-cell (in particular, it can be an elementary membrane) is called its *contents*. The total multiplicities of the elements in an elementary membrane m (the sum of their multiplicities) is called the *size* of m and is denoted by *size*(m).

If a membrane m' is placed in a membrane m such that m and m' contribute to delimiting the same region (namely, the region associated with m), then all objects placed in the region associated with m are said to be *adjacent* to membrane m' (so, they are immediately outside membrane m' and inside membrane m).

A super-cell can be described by a Venn diagram where both the membranes and the objects are represented (in the case of the objects, taking care of multiplicities).

Many further notions can be defined and investigated for super-cells as a goal *per se*. We do not step here into this direction, but we only mention that several operations with super-cells are natural: *merge* (putting together two or more super-cells in a new super-cell), *dissolve* a given membrane (but not the skin, because it defines the super-cell itself), *substitute* an elementary membrane with a given super-cell, *separate* membranes and/or objects of a super-cell, according to given criteria and producing two or more super-cells, etc. Such operations remind us of some of the operations with test tubes used in [1, 2, 17]; those test tube structures can be considered super-cells of depth two, with all objects—DNA molecules mainly—placed in the elementary membranes, the test tubes.

5. TRANSITION P SYSTEMS

We now introduce the main subject of our investigation, a computing mechanism essentially designed as a distributed parallel machinery, having as the underlying structure a super-cell. The basic additional feature is the possibility of objects to evolve, according to certain rules. Another feature refers to the definition of the input and the output of a computation.

A *transition P system* of degree n, $n \geqslant 1$, is a construct

$$\Pi = (V, \mu, w_1, ..., w_n, (R_1, \rho_1), ..., (R_n, \rho_n), i_0),$$

where:

(i) V is an alphabet; its elements are called *objects*;

(ii) μ is a membrane structure of degree n, with the membranes and the regions labeled in a one-to-one manner with elements in a given set Λ; in this section we always use the labels 1, 2, ..., n;

(iii) w_i, $1 \leqslant i \leqslant n$, are strings from V^* representing multisets over V associated with the regions 1, 2, ..., n of μ;

(iv) R_i, $1 \leqslant i \leqslant n$, are finite sets of *evolution rules* over V associated with the regions 1, 2, ..., n of μ; ρ_i is a partial order relation over R_i, $1 \leqslant i \leqslant n$, specifying a *priority* relation among rules of R_i. An evolution rule is a pair (u, v), which we will usually write in the form $u \to v$, where u is a string over V and $v = v'$ or $v = v'\delta$, where v' is a string over

$$(V \times \{here, out\}) \cup (V \times \{in_j \mid 1 \leqslant j \leqslant n\}),$$

and δ is a special symbol not in V. The length of u is called the *radius* of the rule $u \to v$.

(v) i_0 is a number between 1 and n which specifies the *output* membrane of Π.

Of course, any of the multisets $m(w_1)$, ..., $m(w_n)$ can be empty (that is, any w_i can be equal to λ) and the same is valid for the sets R_1, ..., R_n and their associated priority relations ρ_i.

The components μ and w_1, ..., w_n of a P system define a super-cell. Graphically, we will represent a P system by representing its underlying super-cell and also adding the rules to each region, together with the corresponding priority relation. In this way, we can have a complete picture of a P system that is much easier to understand than a symbolic description.

The components μ, w_1, ..., w_n, (R_1, ρ_1), ..., (R_n, ρ_n) constitute the *initial configuration* of Π. In general, any sequence μ', w'_{i_1}, ..., w'_{i_k}, (R_{i_1}, ρ_{i_1}), ..., (R_{i_k}, ρ_{i_k}), with μ' a membrane structure obtained by removing from μ all membranes different from i_1, ..., i_k (of course, the skin membrane is not removed), with $m(w'_j)$ multisets over V, $1 \leqslant j \leqslant k$, and $\{i_1, ..., i_k\} \subseteq \{1, 2, ..., n\}$, is called a *configuration* of Π.

The important detail that the membranes preserve the initial labeling in all subsequent configurations should be noted; in this way, the correspondence between membranes, multisets of objects, and sets of evolution rules is well specified by the subscripts of these elements.

A more compact and easy to read writing of a configuration, avoiding the use of subscripts for multisets and sets above is that where the objects of the multisets are written (using multisets or in the form of a string) directly in the region to which they belong, and, similarly, the rules are written in the region where they can act. This is in a good correspondence with the graphical representation of a transition P system and we will use it especially for configurations where many components are empty.

For two configurations

$$C_1 = (\mu', w'_{i_1}, ..., w'_{i_k}, (R_{i_1}, \rho_{i_1}), ..., (R_{i_k}, \rho_{i_k})),$$
$$C_2 = (\mu'', w''_{j_1}, ..., w''_{j_l}, (R_{j_1}, \rho_{j_1}), ..., (R_{j_l}, \rho_{j_l})),$$

of Π we write $C_1 \Rightarrow C_2$, and we say that we have a *transition* from C_1 to C_2, if we can pass from C_1 to C_2 by using the evolution rules appearing in R_{i_1}, ..., R_{i_k} in the following manner (rather than a completely cumbersome formal definition we prefer an informal one, explained by examples).

Consider a rule $u \to v$ in a set R_{i_t}. We look to the region of μ' associated with the membrane i_t. If the objects mentioned by u, with the multiplicities specified by u, appear in w'_{i_t} (that is, the multiset identified by u is included in $m(w'_{i_t})$), then these objects can evolve according to the rule $u \to v$. The rule can be used only if no rule of a higher priority exists in R_{i_t} and can be applied at the same time with $u \to v$. More precisely, we start to examine the rules in the decreasing order of their priority and assign objects to them. A rule can be used only when there are copies of the objects whose evolution it describes and which were not "consumed" by rules of a higher priority and, moreover, there is no rule of a higher priority, irrespective of which objects it involves, which is applicable at the same step. Therefore, all objects to which a rule *can* be applied *must* be the subject of a rule application. All objects in u are consumed by using the rule $u \to v$; that is, the multiset identified by u is subtracted from $m(w'_{i_t})$.

The result of using the rule is determined by v. If an object appears in v in a pair $(a, here)$, then it will remain in the same region i_t. (Often, when specifying rules, pairs $(a, here)$ are simply written a, the indication "here" is omitted.) If an object appears in v in a pair (a, out), then a will exit the membrane i_t and will become an element of the region immediately outside it (thus, it will be adjacent to the membrane i_t from which it was expelled). In this way, it is possible that an object leaves the super-cell itself: if it goes outside the skin of the system, then it never comes back. If an object appears in a pair (a, in_q), then a will be added to the multiset $m(w'_q)$, providing that a is adjacent to the membrane q. If (a, in_q) appears in v and the membrane q is not one of the membranes delimiting from below the region i_t, then the application of the rule is not allowed.

If the symbol δ appears in v, then the membrane i_t is removed (we say *dissolved*) and at the same time the set of rules R_{i_t} (and its associated priority relation) is removed. The multiset $m(w'_{i_t})$ is added (in the sense of multisets union) to the multiset associated with the region which was immediately external to the membrane i_t. We do not allow the dissolving of the skin, because this means that the super-cell is lost and we no longer have a correct configuration of the system.

All these operations are done in parallel, for all possible applicable rules $u \to v$, for all occurrences of multisets u in the region associated with the rules, for all regions at the same time. No contradiction appears because of multiple membrane dissolving or because of the simultaneous appearance of symbols of the form (a, out) and δ. If at the same step we have (a, in_i) outside a membrane i and δ inside this membrane, then, because of the simultaneity of performing these operations, again no contradiction appears: we assume that a is introduced in membrane i at the same time that it is dissolved. Thus a will remain in the region placed outside membrane i; that is, from the point of view of a, the effect of $(a, in_i), \delta$ is $(a, here)$.

If there are rules in a P system Π with the radius at least two, then the system is said to be *cooperative*; in the opposite case, it is called *noncooperative*. A system is said to be *catalytic* if there are certain objects $c_1, ..., c_n$ specified in advance, called *catalysts*, such that the rules of the system are either of the form $a \to v$ or of the form $c_i a \to c_i v$, where a is a noncatalyst object and v contains no catalyst. (So, the only cooperative rules involve catalysts, which are reproduced by the rule application and left in the same place. There are no rules for the separate evolution of

catalysts.) A transition P system with catalysts is given in the form $\Pi = (V, C, \mu, w_1, ..., w_n, (R_1, \rho_1), ..., (R_n, \rho_n), i_0)$, where $C \subseteq V$ is the set of catalysts. A system is said to be *propagating* if there is no rule which diminishes the number of objects in the system (note that this can be done by "erasing" rules, that is, rules of the form $a \to \lambda$, but also by sending objects out of the skin membrane).

Remark 1. Several observations are worth mentioning with respect to the relationships between the ingredients of our model and the biochemical reality which they remind. First, it should be clear that our goal is not *to model* a real cell, but to propose a (theoretical) computing model inspired by a cell structure. Second, the membranes we consider here are abstract items having two functions: they are *separators* of objects and *channels of communication*. Any kind of an actual or virtual membrane can play these two roles; that is, the notion of a membrane has here a mathematical meaning (similar to [4, 19]), not necessarily a biochemical meaning. Moreover, our membranes are passive components of a P system, in contrast to the actual biomembranes. The latter are, in general, bilayered lipidic structures which leave certain chemical compounds to pass through them by concentration/gradient reasons, because of electrical polarization or, much more selectively, via certain *protein channels*. This last way of using a membrane for communicating is somewhat related to the communication by means of commands of the form (a, in_j). The real membranes can be dissolved, but they can also be inhibited or made opaque to any kind of communication. We will not use this latter possibility here.

Remark 2. The mode of the evolution of objects in a P system provided with evolution rules as described above can be interpreted in the following—idealized—biochemical way. We have a cell, delimited by a skin (the cell membrane). Inside, there are cell organs and free molecules, organized hierarchically. The molecules and the organs float randomly in the cytoplasmic liquid of each membrane. Under specific conditions, the molecules evolve, alone or with the help of certain catalysts; these, of course, are not modified by the reactions (the evolution rules encode chemical reactions among the objects which evolve together). This is done in parallel, synchronously for all molecules (a universal clock is assumed to exist; we will see in the proofs from Sections 7–9 that this rather restrictive assumption of the existence of a universal clock is not essential in what concerns the power of P systems of the forms considered in this paper). The new molecules can remain in the same region where they have appeared or they can pass through the membranes delimiting this space, selectively. Some reactions not only modify molecules, but also break membranes. (We may imagine that certain chemicals are produced which break/dissolve the membrane.) When a membrane is broken, the molecules previously placed inside it will remain free in the larger space newly created, but the evolution rules of the former membrane are lost. The assumption is that the reaction conditions from the previous membrane are modified by the disparition of the membrane and in the newly created space only the rules specific to this space can act. Of course, when the external membrane is broken, then the cell ceases to exist, its parts fall apart.

Remark 3. The way of defining and of using the priority relation deserves a special discussion. Because each rule corresponds to a chemical reaction, the priority

corresponds to the probability that a reaction takes place (some input chemicals can be more active than others). However, we have interpreted the priority in a *strong* sense: if a rule with a higher priority is used, then no rule of a lower priority can be used, even if the two rules do not compete for objects (if $a \to b > c \to d$ and both a and c are available, then only the first rule is used, although it has nothing to do with object c). This interpretation corresponds to the way of using priorities in *ordered grammars* in the regulated rewriting area (see [11]), but it also has a biochemical meaning: imagine that each rule consumes not only objects, but also energy (or other common raw material); if a rule of a higher priority is used, then no energy remains available for rules of a lower priority.

Of course, also the *weak* interpretation of the priority is of interest: a rule is always used when objects exist which were not used by a rule of a higher priority. We do not investigate this variant here.

The following **example** will (hopefully) clarify the definition of a transition in a (cooperative) P system.

Consider the system of degree 4:

$$\Pi = (V, \mu, w_1, ..., w_4, (R_1, \rho_1), ..., (R_4, \rho_4), 4),$$

$$V = \{a, b, c, d\},$$

$$\mu = [_1[_2[_3]_3]_2[_4]_4]_1,$$

$$w_1 = aac,$$

$$w_2 = a,$$

$$w_3 = cd,$$

$$w_4 = \lambda,$$

$$R_1 = \{r_1 : c \to (c, in_4), r_2 : c \to (b, in_4), r_3 : a \to (a, in_2)\, b, dd \to (a, in_4)\},$$

$$\rho_1 = \{r_1 > r_3, r_2 > r_3\},$$

$$R_2 = \{a \to (a, in_3), ac \to \delta\},$$

$$\rho_2 = \varnothing,$$

$$R_3 = \{a \to \delta\},$$

$$\rho_3 = \varnothing,$$

$$R_4 = \{c \to (d, out), b \to b\},$$

$$\rho_4 = \varnothing.$$

(For the sake of simplicity, we have labeled only the rules which appear in the priority relation.)

The system and the configurations obtained after two possible transitions are represented in Fig. 1.

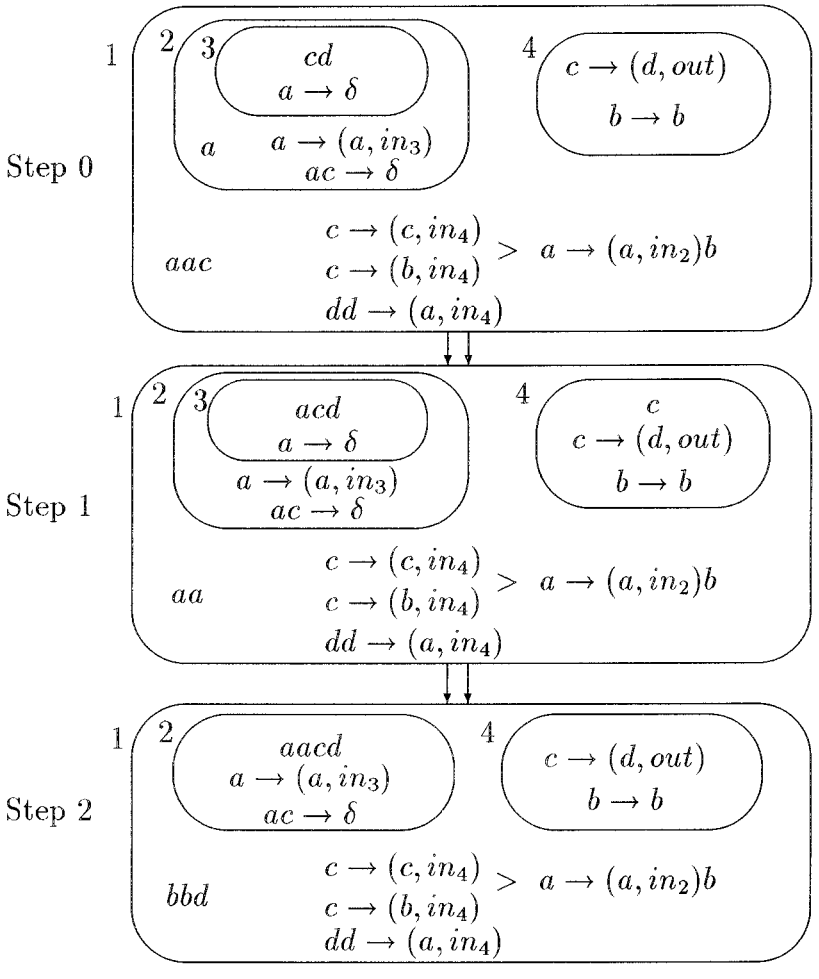

FIG. 1. An example of transitions in a P system.

In the initial configuration we can apply a rule in membrane 1 and one in membrane 2. If in membrane 1 we use the rule $c \to (b, in_4)$, then the computation will never halt: the rule $b \to b$ can be applied forever in membrane 4. Thus, we will not use the rule $c \to (b, in_4)$, but the rule $c \to (c, in_4)$. Because both these rules can be applied and they have priority over the rule $a \to (a, in_2)\, b$, this latter rule cannot be used. Thus, a symbol c is sent from membrane 1 to membrane 4 and at the same time a symbol a is sent from membrane 2 to membrane 3. We get the second configuration from Fig. 1.

Now, no c-rule in membrane 1 can be applied; hence the rule $a \to (a, in_2)\, b$ can be used. It has to be used for both copies of a in membrane 1; hence two copies of a will be sent to membrane 2 and two copies of b will remain in membrane 1. At the same time, the rule $a \to \delta$ will be used in membrane 3, dissolving it, and the rule $c \to (d, out)$ will be used in membrane 4, sending a copy of d to membrane 1. As a result of these operations, membrane 1 will contain the multiset (we write it as a string) bbd and membrane 2 will contain $aacd$, while membrane 4 is empty; membrane 3 no longer exists (hence the rule $a \to (a, in_3)$ in membrane 2 is useless from now on).

Two more transitions can be performed. First, the rule $ac \to \delta$ can be used in membrane 2, dissolving it and releasing the remaining objects *ad*. Thus, membrane 1 will contain the multiset *abbdd*, which makes possible for the first time the use of the rule $dd \to (a, in_4)$ from membrane 1. It consumes the two copies of d and sends a copy of a to membrane 4. At the same time, the rule $a \to (a, in_2)\ b$ sends a copy of a to membrane 2. No further rule can be applied, the "life" of the system stops here.

The computing flavor of such a game is obvious: we start from an initial configuration of our system provided with evolution rules and we get a sequence of transitions.

A sequence of transitions in a P system Π, starting from the initial configuration C_0, is called a *computation* with respect to Π.

A computation $C_0 \Rightarrow C_1 \Rightarrow \cdots \Rightarrow C_m, m \geqslant 0$, is *successful* if and only if each of the following two assertions are true:

1. There is no rule in C_m which can be applied to the objects present in C_m.
2. The membrane i_0 appears in C_m, namely, as an elementary membrane of it.

Reversing these statements, a computation as above is unsuccessful in each of the following two cases:

— It can continue; that is, there exists a configuration C_{m+1} such that $C_m \Rightarrow C_{m+1}$. Note that it is not necessary to have $C_m \neq C_{m+1}$.

— No rule can be applied, but either there is no membrane labeled with i_0 (it has been dissolved by a symbol δ) or there is such a membrane, but it is not an elementary membrane in C_m.

In this way, a P system Π can be seen as a device which generates multisets: start from the initial configuration of Π and let the system evolve. If a successful computation is found, then we say that the multiset contained by the membrane labeled with i_0 is *generated* by Π.

We can also consider the P systems as devices which generate numbers: work as above and say that the size of the membrane i_0 (remember that the size is the sum of multiplicities of objects in a membrane) is the generated number. In what follows, we consider this latter possibility. We denote by $N(\Pi)$ the set of natural numbers generated by Π in the previous sense.

A generalization is to use a P system Π for generating *relations*. For instance, we can specify in advance certain objects $a_{i_1}, ..., a_{i_k}$; if at the end of a successful computation the output membrane contains $n_1, ..., n_k$ occurrences of objects $a_{i_1}, ..., a_{i_k}$, respectively, then we say that $(n_1, ..., n_k)$ belongs to the relation generated by Π.

It is also possible to interpret a P system Π as a device *recognizing* a multiset (that initially placed in a distinguished elementary *input* membrane), a number (the size of an input elementary membrane), a relation (the number of occurrences of certain objects placed in a specified input membrane), or even a device *computing* a partial mapping from natural numbers to sets of natural numbers (give a number as an input, codified in the size of a distinguished elementary membrane, and collect all numbers obtained as outputs at the end of successful computations—if any). We will exemplify some of these possibilities in the next section.

6. EXAMPLES

Before going to investigate the power of P systems, we will examine two examples. Their aim is to further illustrate the way a P system works, as well as to give some hints to the power and the versatility of P systems. We are not concerned with the efficiency of the considered systems (in particular, with making use of the inherent parallelism of a P system for efficiently computing or solving problems).

EXAMPLE 1. Consider the P system of degree 4

$$\Pi_1 = (V, \mu, w_1, ..., w_4, (R_1, \rho_1), ..., (R_4, \rho_4), 4),$$
$$V = \{a, b, b', c, f\},$$
$$\mu = [_1[_2[_3]_3[_4]_4]_2]_1,$$
$$w_1 = \lambda,\ R_1 = \varnothing,\ \rho_1 = \varnothing,$$
$$w_2 = \lambda,\ R_2 = \{B' \to b,\ b \to b(c, in_4),\ r_1: ff \to af,\ r_2: f \to a\delta\},$$
$$\rho_2 = \{r_1 > r_2\},$$
$$w_3 = af,\ R_3 = \{a \to ab',\ a \to b'\delta,\ f \to ff\},\ \rho_3 = \varnothing,$$
$$w_4 = \varnothing,\ R_4 = \varnothing,\ \rho_4 = \varnothing.$$

The system is presented in Fig. 2.

No object is free in membrane 2; hence no rule can be applied here. The only possibility is to start in membrane 3, using the free objects a, f present in one copy each. Using the rules $a \to ab'$, $f \to ff$ in parallel for all occurrences of a and f currently available, after n steps, $n \geq 0$, we get n occurrences of b' and 2^n occurrences of f. In any moment, instead of $a \to ab'$ we can use $a \to b'\delta$ (note that we

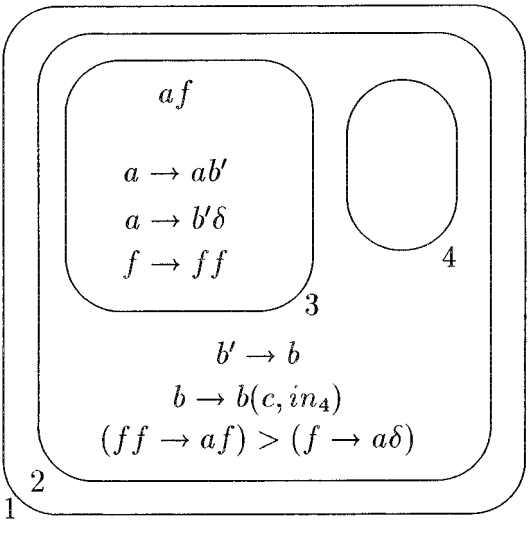

FIG. 2. A P system generating n^2, $n \geq 1$.

always have only one copy of a). In that moment we have $n+1$ occurrences of b' and 2^{n+1} occurrences of f and we dissolve membrane 3. The obtained configuration is

$$[_1[_2 b'^{n+1} f^{2^{n+1}}, b' \to b, b \to b(c, in_4), r_1\colon ff \to af, r_2\colon f \to a\delta, r_1 > r_2, [_4]_4]_2]_1.$$

(We have used again the more compact string notation, α^i, instead of the multiset notation (α, i).)

The rules of the former active membrane 3 are lost and the rules of membrane 2 are now active. Due to the priority relation, we have to use the rule $ff \to af$ as much as possible. In one step, we pass from b'^{n+1} to b^{n+1}, while the number of f occurrences is divided by two. In the next step, from b^{n+1}, $n+1$ occurrences of c are introduced in membrane 4 (each occurrence of the symbol b introduces one occurrence of c). At the same time, the number of f occurrences is divided again by two. We can continue. At each step, additional $n+1$ occurrences of c are introduced in the output membrane. This can be done in $n+1$ steps: n times when the rule $ff \to af$ is used (thus diminishing the number of f occurrences to one) and once when using the rule $f \to a\delta$ (it may now be used). In this moment, membrane 2 is dissolved, which entails the fact that its rules are removed. No further step is possible. The obtained configuration is

$$[_1 a^{2^{n+1}} b^{n+1}, [_4 c^{(n+1)^2}]_4]_1.$$

Consequently,

$$N(\Pi) = \{m^2 \mid m \geqslant 1\}.$$

If we omit membrane 4 (then the rule $b \to b(c, in_4)$ is replaced by $b \to bc$) and consider membrane 1 as the output membrane, then we can generate the set of numbers $\{2^n + n^2 + n \mid n \geqslant 1\}$ (all objects ever used contribute to the output). Furthermore, if we also distinguish the occurrences of b from those of c, then we generate the relation $\sigma = \{(n, m) \mid n \text{ is the square root of } m\}$.

Note that the P system Π_1 is propagating and it has only one cooperative rule.

The previous P system is a *generative* one: it starts from a unique initial configuration and, because of the nondeterministic evolution, it collects in its output membrane different values of n^2, $n \geqslant 1$. A variant of interest could be a P system just *computing* n^2 for a given n. We leave to the reader the task of constructing such a system.

EXAMPLE 2. Let us now consider a P system which has a *decidability* task: we introduce in the input configuration two numbers, n and k, and ask whether or not n is a multiple of k. In the affirmative case, we will finish with one object in the output membrane; in the negative case we will have two objects in the output membrane.

The system is the following (of degree 3):

$$\Pi_2 = (V, \mu, \lambda, a^n c^k d, a, (R_1, \varnothing), (R_2, \rho_2), (\varnothing, \varnothing), 3),$$
$$V = \{a, c, c', d\},$$
$$\mu = [_1[_2]_2[_3]_3]_1,$$
$$R_1 = \{dcc' \to a, in_3)\},$$
$$R_2 = \{r_1: ac \to c', r_2: ac' \to c, r_3: d \to d\delta\},$$
$$\rho_2 = \{r_1 > r_3, r_2 > r_3\}.$$

The structure of Π_2 is better seen in Fig. 3.

In membrane 2 we subtract k from n, repeatedly (by the rules $ac \to c'$, $ac' \to c$: at each step, k copies of a disappear, while c is reproduced, primed or not primed, alternating the priming from one step to another).

The rules $ac \to c'$, $ac' \to c$ have priority over the rule $d \to d\delta$; therefore we can dissolve membrane 2 only after exhausting the n occurrences of a. If n is a multiple of k—and only in this case—then we never have both occurrences of c and of c' simultaneously present in membrane 2 (or in membrane 1, after dissolving membrane 2). Therefore, the rule $dcc' \to (a, in_3)$ is used in membrane 1 if and only if n is not a multiple of k.

Note that this rule can be used at most once, because we have only one occurrence of d and that the computation stops after using the rule $dcc' \to (a, in_3)$.

In conclusion, the computation always stops and the output membrane contains two objects if and only if n is not a multiple of k (in the opposite case, we have here only one object).

The P systems considered above were cooperative systems and always the rules were either propagating or easy to modify in order to obtain propagating rules. We have insisted on the behavior of the P systems and not on their parallelism. This parallelism appears at two levels: the objects in each membrane evolve in parallel,

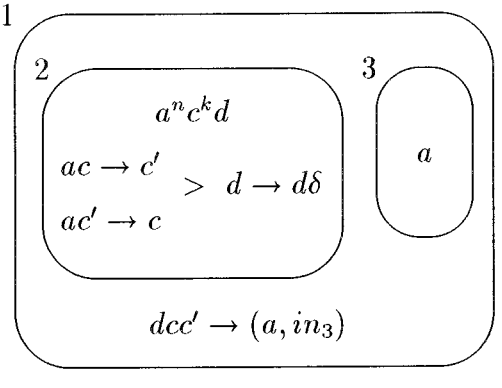

FIG. 3. A P system deciding whether k divides n.

while the membranes themselves evolve in parallel. The influence of the parallelism on the complexity of computing the output (in comparison with other computing models) is one of the main research topics left open.

7. THE POWER OF TRANSITION P SYSTEMS

The transition P systems are computationally complete; systems of a simple structure can compute all recursively enumerable sets of natural numbers. In the proof of this result we need the notion of a *matrix grammar with appearance checking*.

Such a grammar is a construct $G=(N, T, S, M, F)$, where N, T are disjoint alphabets, $S\in N$, M is a finite set of sequences of the form $(A_1\to x_1, ..., A_n\to x_n)$, $n\geqslant 1$, 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 $w\Rightarrow z$ if there is a matrix $(A_1\to x_1, ..., A_n\to x_n)$ in M and the strings $w_i\in(N\cup T)^*$, $1\leqslant i\leqslant n+1$, such that $w=w_1$, $z=w_{n+1}$, and, for all $1<i\leqslant n$, 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 $F=\varnothing$, then the grammar is said to be without appearance checking (and F is no longer mentioned).

We denote by $\Rightarrow^*$ the reflexive and transitive closure of the relation $\Rightarrow$. The language generated by G is defined by $L(G)=\{w\in T^* \mid S\Rightarrow w\}$. 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.

We denote by REG, CF, CS, RE the basic families in the Chomsky hierarchy, of regular, context-free, context-sensitive, and recursively enumerable languages, respectively. When dealing with numbers, RE denotes the family of recursively enumerable sets of natural numbers.

It is known that $CF\subset MAT\subset MAT_{ac}=RE$. Further details about matrix grammars can be found in [11] and in [34].

We also consider here *E0L systems*, which are constructs of the form $G=(V, T, w, P)$, where V is an alphabet, $T\subseteq V$, $w\in V^*$, and P is a finite set of context-free rules $a\to x$ over V; for each $a\in V$ there is at least one rule $a\to x$ in P (we say that P is *complete*). For $y, z\in V^*$ we write $y\Rightarrow z$ iff $y=a_{i_1}\cdots a_{i_k}$, $z=x_{i_1}\cdots x_{i_k}$, for $a_{i_j}\to x_{i_j}\in P$, $1\leqslant j\leqslant k$. The language generated by G is $L(G)=\{z\in T^* \mid w\Rightarrow^* z\}$. We denote by E0L the family of these languages and by Ls(E0L) the family of length sets of E0L languages: the length set of a language $L\subseteq V^*$ is the set $Ls(L)=\{|w| \mid w\in L\}$; E0L is an abbreviation for extended interactionless (zero-interaction) Lindenmayer.

Let us denote by $TP_n(\alpha, \delta)$ the family of sets $N(\Pi)$ of numbers computed by transition P systems of degree at most $n, n\geqslant 1$, of types $\alpha\in\{Coo, Cat, nCoo\}$, where Coo stands for cooperative, Cat for catalytic, and $nCoo$ for noncooperative.

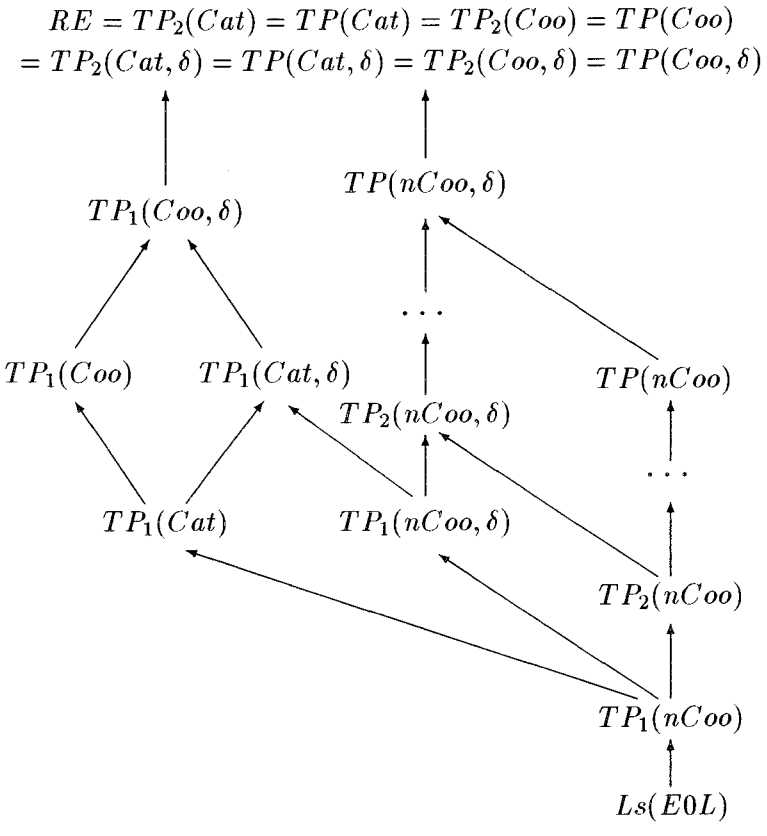

FIG. 4. The hierarchy of the $TP_n(\alpha)$ families.

The union of all families $TP_n(\alpha, \delta)$, $n \geqslant 1$, is denoted by $TP(\alpha, \delta)$. When the membrane dissolving action is not used, then δ is omitted.

THEOREM 1. *The relations in the diagram in Fig.* 4 *hold, where the arrows indicate inclusions which are not necessarily proper.*

Proof. The inclusions between TP families are obvious from the definitions. The inclusion $TP(Coo, \delta) \subseteq RE$ can be proved in a straightforward manner (or we can invoke the Church–Turing thesis). The inclusions $Ls(\text{E0L}) \subseteq TP_1(nCoo)$ and $RE \subseteq TP_2(Cat)$ are proved in the following lemmas. ▌

LEMMA 1. $Ls(E0L) \subseteq TP_1(nCoo)$.

Proof. Consider an E0L system $G = (V, T, w, P)$. For each symbol $a \in V$ we consider a new symbol a'. Let V' be the set of these symbols and h the morphism defined by $h(a) = a'$, for $a \in V$. Assume that P contains m rules, $p_i: a_i \to x_i$, $1 \leqslant i \leqslant m$.

We construct the transition P system of degree 1,

$$\Pi = (V \cup V' \cup \{d, e, \dagger\}, [\,_1]_1, dh(w), (R_1, \rho_1), 1),$$

with the following rules:

$$
\begin{aligned}
&r_1\colon d \to d,\\
&r_2\colon d \to e,\\
&r'_i\colon a'_i \to h(x_i), \qquad \text{for} \quad i = 1, 2, ..., m,\\
&r_3\colon e \to (e, out),\\
&r_a\colon a' \to a, \qquad \text{for} \quad a \in T,\\
&r'_a\colon a' \to \dagger, \qquad \text{for} \quad a \in V - T,\\
&r_\infty\colon \dagger \to \dagger,
\end{aligned}
$$

and the priority relations

$$
\begin{aligned}
&r_1 > r_a, r_1 > r'_a, r_2 > r_a, r_2 > r'_a, \qquad \text{for all possible } a,\\
&r_3 > r'_i, \qquad 1 \leqslant i \leqslant m.
\end{aligned}
$$

The system works as follows. To a multiset (represented here by a string) $dh(z)$ we can apply the rule r_1 and nothing is changed; this forbids the use of rules r_a, r'_a. As long as d is present, each symbol a' present in the current string should evolve by using a rule r'_i associated with the corresponding rule r_i in P. In this way, we simulate the derivations in G, using sentential forms composed of primed symbols. At any moment we can use the rule $d \to e$. Because r_3 is now applicable, no rule r'_i can be used. However, the rules r_a, r'_a are now applicable. If the obtained string is terminal with respect to G, then all primed symbols are replaced by their non-primed versions and the computation stops. If a symbol a' is present, with $a \in V - T$, then the trap-object $\dagger$ is introduced and the computation will continue forever.

Consequently, $Ls(L(G)) = N(\Pi)$. ▮

LEMMA 2 (The computational completeness lemma for transition P systems). $RE \subseteq TP_2(Cat)$.

Proof. Clearly, each set $Q \subseteq \mathbf{N}$ can be identified with the language $L(Q) = \{a^n \mid n \in Q\}$ and Q is recursively enumerable if and only if $L(Q)$ is recursively enumerable. Take a matrix grammar with appearance checking, $G = (N, \{a\}, S, M, F)$ generating the language $L(Q)$, for a given recursively enumerable, set Q of numbers.

According to Lemma 1.3.7 in [11], without loss of generality we may assume that $N = N_1 \cup N_2 \cup \{S, \dagger\}$, with these three sets mutually disjoint, and that 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 $X, Y \in N_1$, $A \in N_2$, $x \in (N_2 \cup \{a\})^*$,
3. $(X \to Y, A \to \dagger)$, with $X, Y \in N_1$, $A \in N_2$,
4. $(X \to x_1, A \to x_2)$, with $X \in N_1$, $A \in N_2$, and $x_1, x_2 \in \{a\}^*$.

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \to \dagger$ appearing in matrices of type 3. The symbol $\dagger$ 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.

Assume that all matrices of forms 2, 3, 4 are labeled in a one-to-one manner, by $m_1, m_2, ..., m_k$.

We construct the following transition P system with catalysts,

$$\Pi = (V, \{c\}, [_1[_2]_2]_1, w_1, \lambda, (R_1, \rho_1), (\varnothing, \varnothing), 2),$$

where

$$w_1 = XAcZ, \qquad \text{for } (S \to XA) \text{ the initial matrix in } M,$$
$$V = N_1 \cup N_2 \cup \{c, D, \dagger, Z\} \cup \{X_i, X_i', X_i'' \mid X \in N_1, 1 \leqslant i \leqslant k\},$$

and the set R_1 contains the following rules (h is the morphism defined by $h(\alpha) = \alpha$, $\alpha \in N_2$, and $h(a) = (a, in_2)$):

1. $X \to X_i$, for all $X \in N_1$ and $1 \leqslant i \leqslant k$.
2. $X_i \to Y'$, for m_i: $(X \to Y, A \to x)$ a matrix of type 2 in M.
3. $cA \to c\ h(x)\ D$, for m_i: $(X \to Y, A \to x)$ a matrix of type 2 in M.
4. $cD \to c$.
5. $cZ \to c\dagger$.
6. $\dagger \to \dagger$.
7. $Y' \to Y$, for all $Y \in N_1$.
8. $cX_i \to cY$, for m_i: $(X \to Y, A \to \dagger)$ a matrix of type 3 in M.
9. $A \to \dagger$, for all $A \in N_2$.
10. $X_i \to X_i'$, for m_i: $(X \to x_1, A \to x_2)$ a matrix of type 4 in M.
11. $cA \to c\ h(x_2)\ D$, for m_i: $(X \to x_1, A \to x_2)$ a matrix of type 4 in M.
12. $X_i' \to X_i''$, for m_i: $(X \to x_1, A \to x_2)$ a matrix of type 4 in M.
13. $Z \to \lambda$.
14. $X_i'' \to h(x_1)$, for m_i: $(X \to x_1, A \to x_2)$ a matrix of type 4 in M.

The priorities are the following (at the same time, we give explanations on the work of Π):

— each rule of type 1 has priority over all rules of other types;
(In the presence of a symbol from N_1 no rule can be used except a rule of type 1, which specifies a matrix to be simulated by the subscript of the symbol X.)

— each rule $X_i \to Y'$ of type 2 has priority over all rules of type 3 associated with matrices m_j with $j \neq i$, as well as over all rules of types 5, 9, 11, 13;
(If a symbol X_i is present, identifying a matrix m_i: $(X \to Y, A \to x)$ of type 2 from M, then the only rules which can be applied are $X_i \to Y'$, because only X_i is present, and $cA \to c\ h(x)\ D$, because all other rules are either of a lower priority than

$X_i \to Y'$ or do not have symbols to which they can be applied. Note that we always have exactly one occurrence of the catalyst; hence the rule $cA \to c\ h(x)\ D$ can be used at most once. By using this rule, one occurrence of the symbol D is introduced.)

— the rule of type 4 has priority, over the rule of type 5;
(This is a very important point of the construction, making a full use of the catalyst: if there is no occurrence of D in the multiset, then the rule $cZ \to c\dagger$ may—and must—be applied, introducing the trap-object $\dagger$ which will evolve forever by the rule $\dagger \to \dagger$. Thus, at the same time with $X_i \to Y'$ we have to use the corresponding rule $cA \to c\ h(x)\ D$, which means that the use of the matrix m_i is correctly simulated. Note that the rule $cZ \to c\dagger$ cannot be used at the previous steps, because of the priority of $X_i \to Y'$ over it.)

— each rule $Y' \to Y$ of type 7, for $Y \in N_1$, has priority over all rules of types 3, 9, 11, 13;
(At the same time with the rule $cD \to c$, providing that D is present, we can use the rule $Y' \to Y$; no rule associated with a rule appearing in the second position in a matrix can be applied and the simulation of the matrix m_i is completed.)

— each rule $cX_i \to cY$ of type 8 has priority over all rules $cA \to c\ h(x)\ D$ associated with matrices of types 2 and 4 and over all rules $B \to \dagger$ with $B \in N_2$ such that $B \neq A$, as well as over all rules of types 5, 11, 13;
(When the symbol X_i points to a matrix m_i of type 3, then the catalyst is kept busy by the rule $cX_i \to cY$, in order not to use the rule $cZ \to c\dagger$; no rule for evolving a symbol from N_2 can be used, because of the priority; if, however, the symbol A from m_i: $(X \to Y, A \to \dagger)$ appears in the current multiset, then the corresponding rule of type 9 should be used and the trap-object is introduced. In this way, we simulate the use of this rule in the appearance checking mode.)

— each rule $X_i \to X_i'$ of type 10 has priority over all rules of type 3 and of type 11 associated with matrices m_j with $j \neq i$, as well as over all rules of types 5, 9, 13;
(When simulating the use of a matrix of type 4, at the last step of a derivation in G, we proceed as for matrices of type 2, with the difference that at the end we have also to introduce a terminal string instead of the control symbol X and also we have to remove the primed successors of X.)

— each rule $X_i' \to X_i''$ of type 12 has priority over all rules of types 3, 11, 13;
(After introducing X_i we replace it with X_i' and, at the same dine, we use the corresponding rule $cA \to c\ h(x_2)\ D$. At the next step, we check whether or not D is introduced, that is, whether or not the simulation is correct. The symbol Z is still present, but it is not used, because of the priorities mentioned above. At the same time, we check whether or not any nonterminal symbol from N_2 is still present: the rules $A \to \dagger$ are available and no other rule using symbols from N_2 can be used; if any rule $A \to \dagger$ can be applied, then it has to be applied.)

— each rule of type 14 has priority over $cZ \to c\dagger$;
(If a symbol X_i' is present, then this means that the computation is finished; we replace this symbol with the corresponding string $h(x_1)$ and we remove the "semi-trap" object Z; the rule $cZ \to c\dagger$ cannot be used.)

From the previous explanations, it is easy to set that each derivation in G can be simulated by a computation in Π and, conversely, each computation in Π corresponds to a derivation in G. It is worth mentioning that this is possible because we deal with a language over the one-letter alphabet, hence the order of symbols appearing in a sentential form of G is not important, only their presence matters (exactly as in a multiset). Moreover, at each moment when an occurrence of a is introduced, it is introduced directly into the output membrane. Nothing else can reach the output membrane. If the derivation is not correctly simulated or it is not terminal, then at least a rule can be further applied, in particular, the rule $\dagger \to \dagger$ if this symbol was produced. Thus, we can conclude that, because $L(G) = L(Q)$, we have $N(\Pi) = Q$. ▮

In the previous construction we paid no attention to the propagating feature, but this can be easily done: just add a dummy object $\#$ which never evolves (and does not enter the output membrane) to the right-hand member of each rule which diminishes the number of objects, $cD \to c$ and $Z \to \lambda$, as well as to rules $X_i'' \to h(x_2)$ of type 14, if $x_2 = \lambda$. Note also that we never dissolve a membrane; hence this feature is useless in this case.

It is also easy to see that we can generate recursively enumerable relations with transition P systems of degree 2 as those used above: a relation $Q \subseteq \mathbf{N}^k$ is characterized by the language $P(Q)$ obtained as the permutation closure of the language $\{a_1^{n_1} \cdots a_k^{n_k} \mid (n_1, ..., n_k) \in Q\}$; starting from a matrix grammar with appearance checking for $P(Q)$, the construction above gives a transition P system for Q (the important observation is again that the order of symbols in the strings of $P(Q)$ is not relevant; hence we can work with multisets instead of strings).

We do not know which of the inclusions in the diagram in Fig. 4 are proper (but at least one should be, because $Ls(E0L)$ is strictly included in RE).

8. P SYSTEMS BASED ON REWRITING

Transition P systems can be interpreted as using no data structure for codifying the information: the numbers are encoded as the cardinality of multisets; hence they are represented in the base one. This can be adequate to a biochemical implementation, but it looks inefficient from a classic point of view. Moreover, in this way we can deal only with problems on numbers, not (directly, without a number codification) with symbolic computations. That is why we now look for representing information by using a data structure of a standard type, *strings*.

Thus, from now on, instead of objects of an atomic type (i.e., without "parts"), we consider objects which can be described by finite strings over a given finite alphabet. The evolution of an object will then correspond to a transformation of the string. In this section we consider transformations in the form of rewriting steps, as is usual in formal language theory.

Consequently, the evolution rules are given as rewriting rules.

Assume that we have an alphabet V. A usual rewriting rule is a pair (u, v) of words over V (we give it in the form $u \to v$). For $x, y \in V^*$ we write $x \Rightarrow y$ iff $x = x_1 u x_2$ and $y = x_1 v x_2$, for some strings $x_1, x_2 \in V^*$.

Here, the rules are also provided with indications on the target membrane of the produced string (we no longer consider the membrane dissolving action, because, similarly to the case of Theorem 1, it will not be necessary in order to obtain computational completeness; of course, if for other purposes it will be useful or necessary to use this action, then it can be introduced in the same way as in the transition P systems). We always use only context-free rules. Thus, the rules are of the form

$$X \to v(tar),$$

where $tar \in \{here, out, in_j\}$ ("tar" comes from "target," j is the label of a membrane), with the obvious meaning: the string produced by using this rule will go to the membrane indicated by tar.

Note the important difference from the way the transition P systems work: a string is now a unique object and hence it passes through membranes as a unique entity; its symbols do not follow different itineraries, as it was possible for the objects in a multiset. Of course, in the same region we can have several strings at the same time.

In this way, we obtain a language generating mechanism of the form

$$\Pi = (V, \mu, L_1, ..., L_n, (R_1, \rho_1), ..., (R_n, \rho_n), i_0),$$

where V is an alphabet, μ is a membrane structure, $L_1, ..., L_n$, are finite languages over V, $R_1, ..., R_n$ are finite sets of context-free evolution rules of the form $X \to v(tar)$, with $X \in V$, $v \in V^*$, $tar \in \{here, out\} \cup \{in_j \mid 1 \leqslant j \leqslant n\}$, $\rho_1, ..., \rho_n$ are partial order relations over $R_1, ..., R_n$, and i_0 is the output membrane. (Note that such a system is a noncooperative one.)

We call such a system a *rewriting P system.*

The language generated by a system Π is denoted by $L(\Pi)$ and it consists of all strings placed in the output membrane at the end of halting computations. A computation is defined in a way similar to that in Section 5, with the differences specific to an evolution based on rewriting: we start from the initial configuration of the system and proceed iteratively, by transition steps done by using the rules in parallel, to all strings which can be rewritten, obeying the priority relations, and collecting the strings generated in a designated membrane, the output one.

Note that each string is processed by one rule only: the parallelism refers here to processing simultaneously all available strings by all applicable rules. If several rules can be applied to a string, maybe in several places each, then we take only one rule and only one possibility to apply it and consider the obtained string as the next state of the object described by the string. It is important to have in mind the fact that the evolution of strings is not independent of each other, but interrelated in two ways: (1) if we have priorities, a rule r_1 applicable to a string x can forbid the use of another rule, r_2, for rewriting another string, y, which is present at that time in the same membrane; after applying the rule r_1, if r_1 is not applicable to y or to the string x' obtained from x by using r_1, then it is possible that the rule r_2 can now be applied to y; (2) even without priorities, if a string x can be rewritten forever, in the same membrane or on an itinerary through several membranes, and this

cannot be avoided, then all strings are lost, because the computation never stops, irrespective of the strings collected in the output membrane and which cannot evolve further.

Remark 4. It is worth noting the similarities and, mainly, the differences between rewriting P systems and parallel communicating grammar systems with communication by queries [26] or by command [8]. Both kinds of systems are distributed parallel devices, making an essential use of communication. In the grammar systems case, the component grammars work synchronously and send to each other sentential forms. Here, the synchronization is not obligatory; a component membrane can wait if its rules, if any, are not applicable. More important, the components of a grammar system are always the same, are arranged in the same level, and can communicate to each other without restrictions (a total graph is available as a communication graph); here the components can be hierarchically arranged in a specified architecture and they can disappear during the computation. Still, the two types of mechanisms meet each other in the generative power: also the parallel communicating grammar systems characterize the recursively enumerable languages, both when communicating by queries [10] and by command [8, 16]. As a common conclusion we call state the fact that communication is very powerful, irrespective of the ways it is done.

A similar comparison holds true with networks of language processors, as introduced in [9] as a generalization of parallel communicating grammar systems: both output and input filters are provided for each component of the system, controlling the flow of strings. Such filters could be considered also in the case of membranes as a substitute for the target indications given by the evolution rules. (They could be a more adequate model of the membrane selectivity due to porosity or to proteine channels present in it.) ▮

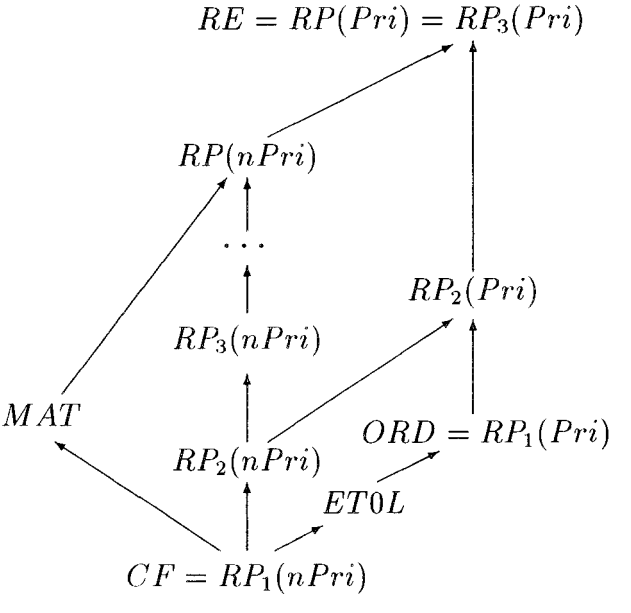

FIG. 5. The hierarchies of $RP_n(\alpha)$ families.

We denote by $RP_n(Pri)$ the family of languages generated by rewriting P systems of degree at most n, $n \geq 1$, using priorities; when priorities are not used, we replace Pri with nPr. The union of all families $RP_n(\alpha)$ is denoted by $RP(\alpha)$, $\alpha \in \{Pri, nPri\}$.

Because we will use below the notion of an ET0L system, we briefly introduce it: such a system is a construct $G = (V, T, w, P_1, ..., P_n)$, such that each (V, T, w, P_i), $1 \leq i \leq n$, is an E0L system. One step of a (parallel) derivation with respect to P_i is denoted by $\Rightarrow_i$, and defined as for E0L systems. The language generated by G is $L(G) = \{z \in T^* \mid w \Rightarrow_{i_1} w_1 \Rightarrow_{i_2} \cdots \Rightarrow_{i_k} w_k = z, \text{ for some } 1 \leq i_j \leq n, 1 \leq j \leq k\}$. The family of languages generated by ET0L systems is denoted by $ET0L$.

By ORD we denote the family of languages generated by context-free ordered grammars (that is, context-free grammars with a partial order relation on the set of rules; a rule can be applied only when no rule of a higher priority can be used).

It is known that $ET0L \subset ORD \subset RE$.

THEOREM 2. *The relations in the diagram in Fig. 5 hold where the arrows indicate inclusions which are not necessarily proper; the inclusion* $CF \subset RP_2(nPri)$ *is proper.*

Proof. The inclusions between the RP families follow from the definitions.

The equality $CF = RP_1(nPri)$ can be proved in the following way: For a context-free grammar $C = (N, T, S, P)$, we construct the rewriting P system

$$\Pi = (N \cup T, [\,]_1, \{S\}, (P \cup \{A \to A \mid A \in N\}, \varnothing), 1).$$

A computation is finished only when no rule $A \to A$ is applicable, which means that no nonterminal symbol is present in the obtained string; hence the computation corresponds to a terminal derivation in G.

Conversely, let Π be a rewriting P system of degree one over some alphabet V. Let P be the set of all rules appearing in Π and L_0 be the finite set of all strings initially present in the system. Denote by T the set of symbols $a \in V$ such that no rule $a \to x$ is in P and by N the set $(V - T) \cup \{S\}$, where S is a new symbol. A symbol $A \in V - T$ for which there is no derivation with respect to P of the form $A \Rightarrow w$ with $w \in T^*$ is said to be *poisoned* (there are rules $A \to x$ for these symbols, but they never lead to a string of terminals). If there is a string in L_0 which contains a poisoned symbol, then $L(\Pi) = \varnothing$. In the opposite case, the context-free grammar $G = (N, T, S, \{S \to x \mid x \in L_0\} \cup P)$ clearly generates the language $L(\Pi)$ (all strings in L_0 lead to strings in T^*).

By adding a partial order relation, we obtain in the same way the equality $ORD = RP_1(Pri)$ (the set of poisoned symbols is defined independent of the order relation among rules: if a rule cannot be applied because a rule with a higher priority is applicable to a nonpoisoned symbol, it will be applied later, when the nonpoisoned symbol is replaced by a terminal one).

The inclusions $RE \subseteq RP_3(Pri)$ and $MAT \subseteq RP(nPri)$ are proved in the following two lemmas.

The fact that the family $RP_2(nPri)$ contains non-context-free languages is proved by the following rewriting P system:

$$\Pi = (\{A, B, a, b, c\}, [_1[_2]_2]_1, \varnothing, \{AB\}, (R_1, \varnothing), (R_2, \varnothing), 2),$$
$$R_1 = \{B \to cB(in_2)\},$$
$$R_2 = \{A \to aAb(out), A \to ab, B \to c\}.$$

It is easy to see that $L(\Pi) = \{a^n b^n c^n \mid n \geqslant 1\}$ (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). This is not a context-free language. ▌

LEMMA 3 (The computational completeness lemma for rewriting P systems). $RE \subseteq RP_3(Pri)$.

Proof. Let $G = (N, T, S, M, F)$ be a matrix grammar with appearance checking in the normal form mentioned at the beginning of the proof of Lemma 2. For each matrix of type 4 $(X \to x_1, A \to x_2)$, with $x_1, x_2 \in T^*$, we also introduce the matrix $(X \to X' x_1, A \to x_2)$, which is considered of type 4′; we also add the matrices $(X' \to \lambda)$; X' is a new symbol associated with X. Clearly, the generated language is not changed. We assume the matrices of the types 2, 3, 4, 4′ labeled in a one-to-one manner with $m_1, ..., m_k$.

We construct the following rewriting P system:

$$\Pi = (V, \mu, L_1, L_2, L_3, (R_1, \rho_1), (R_2, \rho_2), (R_3, \rho_3), 2),$$
$$V = N_1 \cup N_2 \cup \{E, Z, \dagger\} \cup T \cup \{X_i, X_i' \mid X \in N_1, 1 \leqslant i \leqslant k\},$$
$$\mu = [_1[_2]_2[_3]_3]_1,$$
$$L_1 = \{XAE\}, \qquad \text{for } (S \to XA) \text{ the initial matrix in } M,$$
$$L_2 = L_3 = \varnothing,$$
$$R_1 = \{r_\alpha : \alpha \to \alpha \mid \alpha \in V - T, \alpha \neq E\}$$
$$\cup \{r_0 : E \to \lambda(in_2), \dagger \to \dagger\}$$
$$\cup \{X \to Y_i(in_2) \mid m_i : (X \to Y, A \to x) \text{ is a matrix of types 2 or 4}\}$$
$$\cup \{X \to Y_i(in_3) \mid m_i : (X \to Y, A \to \dagger) \text{ is a matrix of type 3}\}$$
$$\cup \{X \to X_i' x_1(in_2), X_i' \to \lambda \mid m_i : (X \to X' x_1, A \to x_2) \text{ is a matrix of type 4'}\}$$
$$\cup \{Y_i \to Y, Y_i' \to Y \mid Y \in N_1, 1 \leqslant i \leqslant k\},$$
$$\rho_1 = \{r_\alpha > r_0 \mid \alpha \in V - T, \alpha \neq E\},$$
$$R_2 = \{r_i : Y_i \to Y_i, r_i' : A \to x(out) \mid m_i : (X \to Y, A \to x) \text{ is a matrix of types 2 or 4}\}$$
$$\cup \{r_i : X_i' \to X_i', r_i' : A \to x_2(out) \mid m_i (X \to X' x_1, A \to x_2) \text{ is a matrix of type 4'}\}$$

$$\rho_2 = \{ r_i > r'_j \mid i \neq j, \text{ for all possible } i, j \},$$

$$R_3 = \{ p_i : Y_i \to Y'_i,\ p'_i : Y'_i \to Y_i,\ p''_i : A \to \dagger(out) \mid m_i : (X \to Y, A \to \dagger) \text{ is a matrix of type 3} \}$$
$$\cup \{ p_0 : E \to E(out) \},$$

$$\rho_3 = \{ p''_i > p_i,\ p_i > p_0 \mid \text{for all possible } i \}.$$

The system works as follows. Observe first that the rules $\alpha \to \alpha$ from membrane 1 change nothing, can be used forever, and prevent the use of the rule $E \to \lambda$, which sends the string to membrane 2, the output one.

Assume that in membrane 1 we have a string of the form XwE (initially, we have here the string XAE for $(S \to XA) \in M$). In membrane 1 one chooses the matrix to be simulated, m_i, and one simulates its first rule, $X \to Y$, by introducing Y_i; the string is sent to membrane 2 if we deal with a matrix of types 2 or 4 (without a rule which has to be applied in the appearance checking mode) and to membrane 3 if we have to simulate a matrix of type 3.

In membrane 2 we can use the rule $Y_i \to Y_i$ forever. The only way to quit this membrane is by using the rule $A \to x$ appearing in the second position of a matrix of type 2 or 4. Due to the priority relation, this matrix should be exactly m_i as specified by the subscript of Y_i. Therefore, we can continue the computation only when the matrix is correctly simulated.

The process is similar in membrane 3: The rules $Y_i \to Y'_i$, $Y'_i \to Y_i$ can be used forever. We can quit the membrane either by using a rule $A \to \dagger(out)$ or by using the rule $E \to E(out)$. In the first case the computation will never halt. Because of the priority relation, such a rule must be used if the corresponding symbol A appears in the string. If this is not the case, then the rule $Y_i \to Y'_i$ can be used. If we now use the rule $Y'_i \to Y_i$, then we get nothing. If we use the rule $E \to E(out)$, and this is possible because Y_i is no longer present, then we send out a string of the form $Y'_i wE$.

In membrane 1 we replace Y_i or Y'_i by Y, and thus the process of simulating the use of matrices of types 2, 3, 4 can be iterated.

A slightly different procedure is followed for the matrices of type 4′; they are of the form $m_i : (X \to X'x_1, A \to x_2)$. In membrane 1 we use $X \to X'_i x_1(in_2)$, which already introduces the substring x_1, and the string arrives in membrane 2. Again the only way to leave this membrane is by using the associated rule $A \to x_2(out)$. In membrane 1 we have to apply $X'_i \to \lambda$. If no symbol different from E and terminals is present, then we can apply the rule $E \to \lambda(in_2)$. Thus, a terminal string is sent to membrane 2, where no rewriting can be done, the computation stops. If any nonterminal symbol is still present, then the computation will never halt, because of the rules $\alpha \to \alpha$ from membrane 1.

Therefore, we collect in the output membrane exactly the terminal strings generated by the grammar G; that is $L(G) = L(\Pi)$. ▌

LEMMA 4. $MAT \subseteq RP(nPri)$.

Proof. Let $G=(N, T, S, M)$ be a matrix grammar without appearance checking. Assume that G is in the normal form used also in the proofs above (this time, the matrices of type 3 are missing), with the matrices labeled in a one-to-one manner. We construct a rewriting P system Π in the following way. In the skin membrane we introduce the initial string XAE, where E is a new symbol and $(S \to XA) \in M$, and rules of the form $X \to z(in_i)$ for each matrix m_i: $(X \to z, A \to x)$ in M, terminal or not. A membrane with the label i will contain the unique rule $A \to x(out)$. In this way, the use of matrices is correctly simulated by the way the strings are circulated among membranes. Note that if we do not use the rule of membrane i, then we cannot leave the membrane; hence the output membrane will remain empty. A special membrane, labeled with O, is the output one; in this membrane we put all the rules $A \to A$, for $A \in N_1 \cup N_2$. In the skin membrane we also consider the rule $E \to \lambda(in_O)$. It can send a string to the output membrane in any moment, but the computation halts only if the string is terminal. The details of this construction are left to the reader. It is obvious that we have $L(\Pi) = L(G)$. ▌

Several problems are *open* in this area: Is the hierarchy $RP_n(nPri)$ an infinite one? Is the result $RE \subseteq RP_3(Pri)$ optimal? Is the inclusion $MAT \subseteq RP(nPri)$ proper? (The difficulty in proving that $RP(nPri) \subseteq MAT$ lies in the dependence between the evolution of the words initially placed in a rewriting P system: even if a string has reached the output membrane and it cannot further evolve, in order to accept it we have to make sure that no other string present in the system can further evolve. This can easily be controlled in a matrix grammar with appearance checking, but we see no way to do it without appearance checking.)

9. SPLICING P SYSTEMS

We now relate the idea of computing with membranes to another important natural computing area, that of DNA computing. Specifically, we consider P systems with objects in the form of strings and with the evolution rules based on the splicing operation introduced in [15]. We first define this operation.

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 $u_1, u_2, u_3, u_4 \in V^*$. For such a rule r and for $x, y, w \in V^*$ we define

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

(One cuts the strings x, y in between u_1, u_2 and u_3, u_4, respectively, and one concatenates the "first half" of x with the "second half" of y.) We say that we *splice* the strings x, y at the *sites* $u_1, u_2, u_3 u_4$, respectively. When r is understood, we write $\vdash$ instead of $\vdash_r$. For clarity, we usually indicate by a vertical bar the place of splicing: $(x_1 u_1 \mid u_2 x_2, y_1 u_3 \mid u_4 y_2) \vdash x_1 u_1 u_4 y_2$.

Based on this operation, language generating devices were introduced: start from a given set of strings and apply to them iteratively the splicing rules from a given set. We obtain what is called an *H system*. If a terminal alphabet is considered, then

we obtain an *extended* H system. It is known that extended H systems with finite sets of axioms and finite sets of rules characterize the regular languages and that systems with certain controls on the use of rules or with certain distributed architectures characterize the recursively enumerable languages. Comprehensive details can be found in [22].

Because we need a string-to-string transformation, we shall consider here a variant of the relation $\vdash$, as a binary relation.

Specifically, with each splicing rule $r = u_1 \# u_2 \$ u_3 \# u_4$ over a given alphabet V we associate a string $z \in V^*$. For $x, y \in V^*$ we write

$$x \Rightarrow_{(r,z)} y \qquad \text{iff either} \quad (x, z) \vdash_r y \qquad \text{or} \qquad (z, x) \vdash_r y.$$

When (r, z) is understood, we write simply $\Rightarrow$ instead of $\Rightarrow_{(r,z)}$.

Such transformations can be used for defining transitions in P systems with string-objects.

A *splicing P system* (over a given alphabet V) is a P system Π with strings as objects, with evolution rules given in the form $(r, z)tar$, where r is a splicing rule over V, $z \in V^*$, and tar is a target indication for the resulting string, one of *here*, *out*, in_j. (As usual, the indication *here* will be omitted when writing a system.) With respect to such a rule we define a relation $x \Rightarrow_{(r,z)} y(tar)$ as mentioned above. Using this relation, we define the transition between configurations, taking into consideration also a possible priority relation among evolution rules. (We do not provide the membrane dissolving action, because it is again not necessary for obtaining computational completeness.) A computation is correctly finished in the same conditions as in the previous sections: no further move is possible since one elementary membrane is designated as the output one. The language generated by Π is the set of strings placed in the output membrane at the end of correctly finished computations.

We denote by $SP_n(Pri)$ the family of languages generated by splicing P systems of degree at most n, $n \geq 1$, using priorities; when priorities are not used, we replace *Pri* with *nPri*; the union of all families $SP_n(\alpha)$ is denoted by $SP(\alpha)$, $\alpha \in \{Pri, nPri\}$.

We also denote by EH the family of languages generated by extended splicing systems with finite sets of axioms and of splicing rules and by $EH(Ord)$ the family of languages generated by extended splicing systems with finite sets of axioms and of splicing rules and with a priority on the set of rules (we use a splicing rule for splicing two strings only if no rule with a higher priority can be used for splicing these strings).

It is known that $EH = REG$ and $EH(Ord) = RE$.

Rather expected, in view of the results from the previous sections and from [22], we have the following result:

THEOREM 3. *The relations in the diagram in Fig. 6 hold, where the arrows indicate inclusions which are not necessarily proper; the family* $SP_2(nPri)$ *contains nonregular languages, while* $SP_3(nPri)$ *contains languages which are not in the family MAT.*

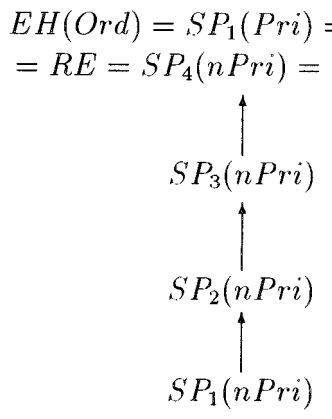

FIG. 6. The hierarchy of families $SP_n(\alpha)$.

Proof. The inclusions follow from the definitions. The equality $EH(Ord) = RE$ (see the proof of Theorem 8.3 in [22] for checking that the halt condition from splicing P systems is fulfilled by the ordered extended splicing system constructed for simulating a given type-0 grammar) implies the collapse of the hierarchy $SP_n(Pri)$ at its first level.

The inclusion $RE \subseteq SP_4(nPri)$ is proved in the next lemma.

In order to show that $SP_2(nPri)$ contains nonregular languages, let us consider the system

$$\Pi = (\{a, b, d\}, [_1[_2]_2]_1, \{dabd\}, \varnothing, (R_1, \varnothing), (R_2, \varnothing), 2),$$
$$R_1 = \{(da\#Z\$da, daZ)in_2\},$$
$$R_2 = \{(b\#d\$Z\#bd, Zbd)out, (b\#d\$Z\#b, Zb)\}.$$

It is easy to see that strings of the form da^nb^nd (initially, we have $n = 1$) get one more occurrence of a in the skin membrane and are passed to the output membrane. Here one more occurrence of b is added, the process is iterated, and, at any moment, in the output membrane we can use the rule $b\#d\$Z\#b$ and the right-hand occurrence of the symbol d is removed. No further splicing is possible; hence the computation is correctly finished. Consequently, we get

$$L(\Pi) = \{da^nb^n \mid n \geq 2\}.$$

This is not a regular language.

For the similar assertion $SP_3(nPri) - MAT \neq \varnothing$ we use the following system:

$$\Pi = (\{a, b, c, X, Y, Z\}, [_1[_2]_2[_3]_3]_1, \{XabY\}, \varnothing, \varnothing, (R_1, \varnothing),$$
$$(R_2, \varnothing), (R_3, \varnothing), 3),$$
$$R_1 = \{(X\#Z\$Xa\#, XZ)in_2, (X\#Z\$Xb\#, XZ)in_3,$$
$$(c\#Z\$Xb\#, cZ)in_3, (c\#Z\$c\#, cZ)\},$$

$$R_2 = \{(\#\, Y\$Z\#aaY, ZaaY)out\},$$

$$R_3 = \{(\#\, Y\$Z\#BY, ZbY)out\}, (\#\, Y\$Z\#c, Zc), (X\#Z\$X\#, XZ)\}.$$

Assume that we have a string of the form Xa^iba^jY in membrane 1; initially, we have $i=1$, $j=0$. If $i \geqslant 1$, then we have to use the rule $X\#Z\$Xa\#$ and the string $Xa^{i-1}ba^jY$ is sent to membrane 2. There is only one rule here; hence we get the string $Xa^{i-1}ba^{j+2}Y$, which is sent back to membrane 1. In this way; we will eventually obtain the string $Xba^{j+2i}Y$. If we use the rule $X\#Z\$Xb\#$, then we obtain the string $Xa^{j+2i}Y$ which is sent to membrane 3. If we use the rule $\#\, Y\$Z\#c$, then we obtain the string $Xa^{j+2i}c$ which will remain in membrane 3 and can be processed forever by using the rule $X\#Z\$\#$. The only way to continue in such a way that the computation will be successfully finished is to use the rule $\#\, Y\$Z\#bY$; the obtained string $Xa^{j+2i}bY$ is returned to membrane 1 and the process is iterated. In this way, the number of occurrences of a is doubled at each passing of the string through membrane 3.

When in membrane 1 we have a string Xba^jY, then we can also use the rule $c\#Z\$Xb\#$, the marker X is replaced with c and the string is sent to membrane 3. If we apply here the rule $\#\, Y\$Z\#bY$, then the string ca^ibY arrives in membrane 1 and it will be processed forever by using the rule $c\#Z\$c\#$. Thus, in order to halt, in membrane 3 we have to use the rule $\#\, Y\$Z\#c$; that is we get the string ca^ic. No further rule can be applied.

In conclusion, we obtain the language $L(\Pi) = \{ca^{2^n}c \mid n \geqslant 1\}$, which is not in the family MAT: each one-letter language in MAT is regular, see [14], and $L(\Pi)$ can be mapped to the nonregular language $\{a^{2^n} \mid n \geqslant 1\}$ by a morphism; MAT is closed under morphisms, see [11]. ∎

LEMMA 5 (The computational completeness lemma for splicing P systems). $RE \subseteq SP_4(nPri)$.

Proof. Let $G=(N, T, S, P)$ be a type-0 Chomsky grammar. Assume that $N \cup T = \{D_1, ..., D_n\}$ and take a further symbol, B, also denoted by D_{n+1}.

We construct the following splicing P system (of degree 4 and without priorities):

$$\Pi = V, \mu, L_1, L_2, L_3, L_4, (R_1, \varnothing), (R_2, \varnothing), (R_3, \varnothing), (R_4, \varnothing), 4),$$

$$V = \{N \cup T \cup \{Z, B\}\} \cup \{X_i, Y_i \mid 0 \leqslant i \leqslant n+1\},$$

$$\mu = [_1[_2]_2[_3]_3[_4]_4]_1,$$

$$L_1 = L_3 = L_4 = \varnothing,$$

$$L_2 = \{XSBY\},$$

$$R_1 = \{(X_iD_i\#Z\$X\#, X_iD_iZ)in_2 \mid 1 \leqslant i \leqslant n+1\}$$
$$\cup \{(X_{i-1}\#Z\$X_i\#, X_{i-1}Z)in_2 \mid 2 \leqslant i \leqslant n+1\}$$
$$\cup \{(X\#Z\$X_1\#, XZ)in_3, (\#\, BY\$Z\#BY', ZBY')in_2,$$
$$(\#\, Y''\$Z\#, Z)in_4, (\#\dagger\$Z\#\dagger, Z\dagger)\},$$

$$R_2 = \{(\#uY\$Z\#vY, ZvY) \mid u \to v \text{ is a rule from } P\}$$
$$\cup \{(\#D_iY\$Z\#Y_i, ZY_i)out, (\#Y_i\$Z\#Y_{i-1}, ZY_{i-1})out \mid 1 \leqslant i \leqslant n+1\}$$
$$\cup \{(\#D_iY\$Z\#Y'_i, ZY'_i)out \mid D_i \in T \cup \{B\}, 1 \leqslant i \leqslant n+1\}$$
$$\cup \{(\#Y'_i\$Z\#Y'_{i-1}, ZY_{i-1})out) \mid 1 \leqslant i \leqslant n+1\}$$
$$\cup \{(\#Y_0\$Z\#\dagger, Z\dagger)out\},$$
$$R_3 = \{(\#Y_0\$Z\#Y, ZY)out, (\#Y'_0\$Z\#Y', ZY')out, (\#BY'_0\$Z\#Y'', ZY'')out\}$$
$$\cup \{(\#Y_i\$Z\#\dagger, Z\dagger)out, (\#Y'_i\$Z\#\dagger, Z\dagger)out \mid 1 \leqslant i \leqslant n+1\},$$
$$R_4 = \{(\#Z\$X\#, Z)\}.$$

The system works as follows. There is only one string in the initial configuration (namely, in membrane 2), $XBSY$, which introduces the axiom of G, together with the new symbol B, and the end markers X, Y. Therefore, always we will have only one string in the system.

Assume that we have a string of the form XwY in membrane 2. If we apply a splicing rule $\#uY\$Z\#vY$, then we simulate the use of a rule from P at the end of the string, $Xw'uY \Rightarrow Xw'vY$, and this corresponds to $w'u \Rightarrow w'v$ (modulo an occurrence of B in the string w') in G. The string remains in membrane 2.

If we perform a splicing

$$(Xw' \mid D_iY, Z \mid Y_i) \vdash Xw'Y_i,$$

for some $i = 1, 2, ..., n+1$, then the string exits the membrane. In the skin membrane, we have only the possibility to use a splicing rule of the form $X_jD_j\#Z\$X\#$. In this way, we get a string $X_jD_jw'Y_i$, which is again passed to membrane 2. Here, we have to decrease by one the subscript of the right end marker. The obtained string, $X_jD_jw'Y_{i-1}$, is sent to the skin membrane, where the subscript of the left end marker is decreased by one; the obtained string is sent to membrane 2 and the process is repeated. When in membrane 2 we get $X_kD_jw'Y_0$ and this string is again passed to the skin membrane; if $k = 1$, then the rule $X\#Z\$X_1\#$ is used here and the resulting string is passed to membrane 3. Only strings with the subscript of Y equal to zero can be processed in membrane 3 without introducing the trap-symbol $\dagger$ in the currently produced string (once introduced, this symbol will be processed forever in the skin membrane by the rule $\#\dagger\$Z\#\dagger$; hence the computation can never be finished. Note that the string is of the form $Xw\dagger$; hence no other rule but $\#\dagger\$Z\#\dagger$ can be used in the skin membrane. Using this rule, we do not change the string: $(Xw \mid \dagger, Z \mid \dagger) \vdash Xw\dagger$). If the string $X_kD_jw'Y_0$ is passed from membrane 2 to the skin membrane and $k \geqslant 2$, then the rule $X_{k-1}\#Z\$X_k\#$ is used and the string $X_{k-1}D_jw'Y_0$, with $k-1 \geqslant 1$, is sent to membrane 2. The only rule in membrane 2 which can be applied is $\#Y_0\$Z\#\dagger$, which introduces the trap-object $\dagger$. The computation is never terminated.

If a string $X_1D_jw'Y_k$ with $k \geqslant 1$ is produced in membrane 2 and sent to membrane 1, here we can only apply the rule $X\#Z\$X_1\#$ and the string $XD_jw'Y_k$

is sent to membrane 3. No rule but $\# Y_i \$ Z \# \dagger$ is here applicable; hence the computation never terminates.

Consequently, in order to finish correctly the computation, the subscripts of the end markers have to reach the value zero at the wine time; that is, $i = j$. This means that the symbol D_i which was cut from the right end of the string has been reproduced in the left end of the string. Note that the symbol B can be moved from an end of the string to the other one like any symbol from $N \cup T$.

In this way, the string is circularly permuted, making possible the simulation of rules of G in any position. In each moment, there is exactly one occurrence of the symbol B, indicating the beginning of the sentential form of G simulated by our system: if in Π we have generated the string Xw_1Bw_2Y, then the string w_2w_1 is a sentential form of G, and conversely.

When the rule $\# BY \$ Z \# BY'$ is used in the skin membrane, the resulting string of the form $XwBY'$ is sent to membrane 2; no simulation of a rule in P is possible from now on, but only circular permutations. Such circular permutations can be performed as above, using the primed right end markers Y_i' instead of Y_i, $0 \leqslant i \leqslant n+1$. However, it is important to note that only symbols which are terminal with respect to G can be moved. In any moment when in membrane 3 we have a string of the form $XwBY_0'$ (received from the skin membrane), we can choose to replace BY_0' by Y''. The fact that B is in the right end of the string tells us that w is a sentential form of G (in a nonpermuted form). Moreover, because B is again in the right end, this implies that at least one circular permutation of the string wB has been done since Y' has been introduced; that is, the string w is terminal. The obtained string, Xw, is sent to membrane 4, where the left marker is removed. No rule can now be applied and the computation stops with the output w. Because we know that this string can be generated by G, we have the equality $L(G) = L(\Pi)$. ▌

We do not know whether or not the degree of the system in this lemma can be further decreased.

If we provide a splicing P system Π with a terminal alphabet T and we define the language generated as consisting of all strings over T collected in the output membrane at the end of halting computations, then systems of degree three can characterize the recursively enumerable languages: in the proof of Lemma 5, membrane 4 is no longer necessary, while membrane 3 can be considered the output one (a rule able to remove X_1B in the skin membrane and a rule for removing Y_0 in the output membrane should be considered). We can formulate this observation also in the following form:

COROLLARY 1. *Each recursively enumerable language $L \subseteq T^*$ can be written in the form $L = L' \cap T^*$, where $L' \in SP_3(nPri)$.*

10. FINAL REMARKS

We have introduced a new computability model, called a P system, based on the evolution of objects in a membrane structure. It uses the architecture of a chemical

abstract machine in the sense of [4], with the rules inspired from the Γ-systems of [3], provided with extra features concerning the paths of objects through membranes and the possibility of dissolving membranes. The objects can be single symbols or strings of symbols; in the latter case, the evolution is defined in terms of string transformations. We have considered here rewriting and splicing as underlying operations with strings. In all three cases, basic (transition) P systems, rewriting P systems, and splicing P systems, we obtain computational completeness and characterizations of recursively enumerable sets of natural numbers (of relations, too) and of recursively enumerable languages by systems of a rather simple form.

From the proofs of these results we can draw an important observation: the computational completeness is obtained without making use of the synchronized evolution of objects and membranes. Synchronization, in the sense of a universal clock, is assumed in the definition of transitions in P systems, but it does not appear in the proofs of the three computational completeness lemmas: in the case of transition P systems we have only one working membrane and in the case of rewriting and splicing P systems we always have only one string in the system; hence the synchronization has no object. (We can restate this by saying that a separate clock for each membrane suffices.)

On the other hand, in the proofs we have simulated the sequential computations done by various types of (matrix) grammars in the framework of P systems, thus losing the central attractive feature of these systems, the parallelism. These proofs are a way to clarify the power of P systems, but not to implement algorithms by starting from a sequential representation of them (e.g., a Turing machine). Direct constructions of P systems carrying out given tasks should be found.

Many open problems and research topics were formulated and still many more questions naturally arise in this framework. We only mention some of them. Of definite interest is to consider *deterministic* P systems, having in each moment at most one possible transition. This might be important in the case that such devices would be implemented in "reliable" media, which behave deterministically. Of course, in biochemical-like media, where a huge parallelism is possible, the nondeterminism could be useful, because by using it we can simulate the deterministic parallelism (with a high probability, working nondeterministically on a large number of "processors" we can get the result of a parallel exploration of the search space).

We have considered above that when a membrane is dissolved, only the objects survive and the rules of the former membrane are lost. This, of course, can be changed. Moreover, in the same way as the objects evolve, we may assume that also the rules evolve; for instance, we can allow also rules to move from one membrane to another. Still further, the membranes can also evolve, not only by disappearing under the influence of certain object evolution rules, but also other modes. Creating new membranes can be done either by usual action rules (instead of a symbol δ, consider a symbol τ_X, with the meaning "create a new membrane, labeled with X") or by membrane evolution rules (duplication, separation in two distinct membranes, etc.) Some technical problems appear here concerning the contents of the new membranes and the objects and the rules to be put into them (certain inheritance principles should be considered). A small jungle of variants can be produced in this way.

We finish by stressing again the importance of parallelism in P systems, appearing at two levels in transition and splicing P systems and, possibly, in three levels in rewriting P systems: we can also use the rules in parallel in the sense of Lindenmayer systems (each symbol of a string which can be rewritten should be rewritten; then, *all* strings are rewritten simultaneously, in *all* membranes of the system). The influence of parallelism on the time complexity of computations in P systems is a question of a basic interest. It is highly conceivable that when rules for producing new membranes are provided, by creating an exponential number of membranes, an essential speed-up can be obtained (perhaps even polynomial time computations, done in parallel, of solutions to exponential problems).

An important problem, not mentioned in the formal framework above, concerns the possibility of implementing P systems, either in electronic media or in biological media. A related, double, problem is (1) too find specific computing problems which can be solved in this way, and (2) to find natural processes (for instance, biological) which can be considered as counterparts of membrane structures we used here or ones at least similar to the operations used in our P systems (for instance, moving objects through membranes in a well-specified manner, dissolving membranes—producing "holes" in there, etc.). The answer to such questions can direct the theoretical studies to the most promising and practically relevant direction.

ACKNOWLEDGMENTS

This work was supported by the Academy of Finland, Project 137358. Useful discussions with J. Dassow, H. Fernau, M. Holzer, A. Mateescu, I. Petre, K. Reinhardt, G. Rozenberg, and A. Salomaa are gratefully acknowledged. Many thanks are due to an anonymous referee for many useful remarks on an earlier version of this paper. A preliminary version of this paper was placed on the Internet in November 1998 as "TUCS Technical Report No 208" (www.tucs.fi). Several continuations are now (June 1999) available, some of which answer questions suggested above. For the reader's convenience, all known titles are mentioned in the References. We emphasize some of these titles: Refs. [18, 35] deal with implementations of various types of P systems on usual computers, Ref. [20] considers the case when communication is controlled by electrical polarization (but a further operation of inhibiting all communications through a membrane is also used) Ref. [21] introduces the operation of membrane division (SAT, the satisfiability of propositional formulas in the conjunctive normal form, one of the most notorious NP-complete problems, is then solved in linear parallel time. As expected, the result is obtained of the expense of using an exponential space; such a space-time trade-off is known—and accepted—in natural computing, e.g., in DNA computing), Refs. [31, 24] deal with normal forms of the membrane structures, Ref. [32] discusses a connection between P systems and the ambient calculus of Refs. [6, 5], while Ref. [13] extends the notion of a P system to arrays and graphs.

REFERENCES

1. L. M. Adleman, On constructing a molecular computer, *in* "DNA Based Computers" (R. J. Lipton and E. B. Baum, Eds.), pp. 1–22, Proc. of a DIMACS Workshop, Princeton, 1995, Amer. Math. Soc., Providence, RI, 1996" pp. 1–22,.
2. M. Amos, "DNA Computing," Ph.D. thesis, Dept. of Computer Sci., Univ. of Warwick, 1997.
3. J. P. Banâtre, A. Coutant, and D. Le Metayer, A parallel machine for multiset transformations and its programming style, *Future Gener. Comput. Systems* **4** (1988), 133–144.
4. G. Berry and G. Boudol, The chemical abstract machine, *Theoret. Comput. Sci.* **96** (1992), 217–248.

5. L. Cardelli, Abstractions for mobile computation, *in* "Secure Internet Programming" (J. Vitek and Ch. Jensen, Eds.), Lecture Notes in Computer Science, Vol. 1603, Springer-Verlag, Berlin, 1999.
6. L. Cardelli and A. Gordon, Mobile ambients, *in* "Proceedings of FoSSaCS'98" (M. Nivat,, Ed.), Lecture Notes in Computer Science, Vol. 1378, pp. 140–155, Springer-Verlag, Berlin, 1998.
7. E. Csuhaj-Varju, J. Dassow, J. Kelemen, and Ch. Păun, "Grammar Systems. A Grammatical Approach to Distribution and Cooperation," Gordon and Breach, London, 1994.
8. E. Csuhaj-Varju, J. Kelemen, and Ch. Păun, Grammar systems with WAVE-like communication, *Comput. Artificial Intelligence* **15** (1996), 419–436.
9. E. Csuhaj-Varju and A. Salomaa, Networks of parallel language processors, *in* "New Trends in Formal Languages. Control, Cooperation, Combinatorics" (Gh. Păun and A. Salomaa, Eds.), Lecture Notes in Computer Science, Vol. 1218, pp. 299–318, Springer-Verlag, Berlin, 1997.
10. E. Csuhaj-Varju and G. Vaszil, On the computational completeness of context-free PC grammar systems, *Theoret. Comput. Sci.* **215** (1999), 348–358.
11. J. Dassow and Gh. Păun, "Regulated Rewriting in Formal Language Theory," Springer-Verlag, Berlin, 1989.
12. J. Dassow and Gh. Păun, On the power of membrane computing, *J. Universal Comput. Sci.* **5**, 2 (1999), 33–49 [www.iicm.edu/jucs].
13. F. Freund, Generalized P systems, *in* "Proc. of FCT'99" (G. Ciobanu and Gh. Păun, Eds.), Lecture Notes in Computer Science, Vol. 1684, Springer-Verlag, Berlin, 1999.
14. D. Hauschildt and M. Jantzen, Petri nets algorithm in the theory of matrix grammars, *Acta Inform.* **31** (1994), 719–728.
15. T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology* **49** (1987), 737–759.
16. L. Ilie and A. Salomaa, 2-testability and relabeling procedure everything, *J. Comput. System Sci.* **56** (1998), 253–262.
17. R. J. Lipton, Speeding up computations via molecular biology, *in* "DNA Based Computers" (R. J. Lipton and E. B. Baum, Eds.), Proc. of a DIMACS Workshop, Princeton, 1995, pp. 67–74, Amer. Math. Soc., Providence, RI, 1996.
18. M. Maliţa, Membrane computing in Prolog, manuscript, 1999.
19. V. Manca, String rewriting and metabolism: A logical perspective, *in* "Computing with Bio-Molecules. Theory and Experiments" (Gh. Păun, Eds.), pp. 36–60, Springer-Verlag, Berlin/New York, 1998.
20. Gh. Păun, Computing with membranes. A variant, submitted, 1999. [Auckland University, CDMTCS Report 098, 1999 (www.cs.auckland.ac.nz/CDMTCS)]
21. Ch. Păun, P systems with active membranes: Attacking NP complete problems, *J. Automat. Languages Combin.*, to appear. [Auckland University, CDMTCS Report 102, 1999 (www.cs.auckland.ac.nz/CDMTCS)]
22. Gh. Păun, G. Rozenberg, and A. Salomaa, "DNA Computing. New Computing Paradigms," Springer-Verlag, Berlin/New York, 1998.
22. Gh. Păun, G. Rozenberg, and A. Salomaa, Membrane computing with external output, submitted, 1999. [Turku Center for Computer Science, TUCS Report 218, 1988 (www.tucs.fi)]
23. Gh. Păun, Y. Sakakibara, and T. Yokomori, P systems on graphs of restricted forms, submitted, 1999.
25. Gh. Păun and A. Salomaa, (Eds.), "Grammatical Models of Multi-Agent Systems," Gordon and Breach, London, 1998.
26. Gh. Păun and L. Sântean, Parallel communicating grammar systems; the regular case, *Ann. Univ. Buc., Matem.-Inform. Series* **38**, 2 (1989), 55–63.
27. Gh. Păun and G. Thierrin, Multiset processing by means of systems of finite state transducers, *in* "Workshop on Implementing Automata WIA99, Potsdam, August 1999." [Auckland University, CDMTCS Report 101, 1999 (www.cs.auckland.ac.nz/CDMTCS)]

28. Gh. Păun and T. Yokomori, Membrane computing based on splicing, *in* "Fifth Inter. Worshop on DNA Based Computers, DNA5, MIT, 1999."

29. Ch. Păun and T. Yokomori, Simulating H systems by P systems, *J. Universal Comput. Sci.* **5** (1999), to appear. [www.iicm.edu/jucs]

30. Ch. Păun and S. Yu, On synchronization in P systems, *Fund. Inform.* **37** (1999). [University of Western Ontario Report TR 539, 1999 (www.csd.uwo.ca/faculty/syu/TR539.html)]..

31. I. Petre, A normal form for P systems, *Bull. EATCS* **67** (1999), 165–172.

32. I. Petre and L. Petre, Mobile ambients and P systems, *in* "Workshop on Formal Languages, FCT'99, Iaşi, Romania, 1999."

33. G. Rozenberg and A. Salomaa, "The Mathematical Theory of L Systems," Academic Press, New York, 1980.

34. G. Rozenberg and A. Salomaa (eds.), "Handbook of Formal Languages," Springer-Verlag, Berlin/New York, 1997.

35. Y. Suzuki, and H. Tanak, On a LISP implementation of a class of P systems, submitted 1999.