Computing with Membranes (P Systems): Universality Results

Carlos Martín-Vide[1] and Gheorghe Păun[2]

[1] Research Group in Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
cmv@astor.urv.es
[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
gpaun@imar.ro, g_paun@hotmail.com

Abstract. This is a survey of universality results in the area of Membrane Computing (P systems), at the level of December 2000[1]. We consider both P systems with symbol-objects and with string-objects; in the latter case, we consider systems based on rewriting, splicing, as well as rewriting together with other operations (replication, crossingover), with sets or with multisets of strings. Besides recalling characterizations of recursively enumerable languages and of recursively enumerable sets of vectors of natural numbers, we also briefly discuss the techniques used in the proofs of such results. Several open problems are also formulated.

1 Introduction; The Basic Idea

The P systems (initially, in [28], they were called *super-cell* systems) were introduced as a possible answer to the question whether or not the frequent statements (see, e.g., [3], [19]) that the processes which take place in a living cell are "computations", that "the alive cells are computers", are just metaphors, or a formal computing device can be abstracted from the cell functioning. As we will see below, the answer turned out to be affirmative.

Three are the fundamental features of alive cells which are basic to P systems: (1) the complex compartmentation by means of a **membrane structure**, where (2) **multisets** of chemical compounds evolve according to prescribed (3) **rules**.

A *membrane structure* is a hierarchical arrangement of membranes, all of them placed in a main membrane, called the *skin* membrane. This one delimits the system from its environment. The membranes should be understood as three-dimensional vesicles, but a suggestive pictorial representation is by means of planar Euler-Venn diagrams (see Figure 1). Each membrane precisely identifies a *region*, the space between it and all the directly inner membranes, if any exists. A membrane without any membrane inside is said to be *elementary*.

[1] An up-to-date bibliography of the area can be found at the web address http://bioinformatics.bio.disco.unimib.it/psystems

M. Margenstern and Y. Rogozhin (Eds.): MCU 2001, LNCS 2055, pp. 82–101, 2001.

In the regions of a membrane structure we place sets or *multisets* of *objects*. A multiset is a usual set with multiplicities associated with its elements, in the form of natural numbers; the meaning is that each object can appear in a number of identical copies in a given region. For the beginning, the objects are supposed to be symbols from a given alphabet (we will work with finitely many types of objects, that is, with multisets over a finite support-set), but later we will also consider string-objects.

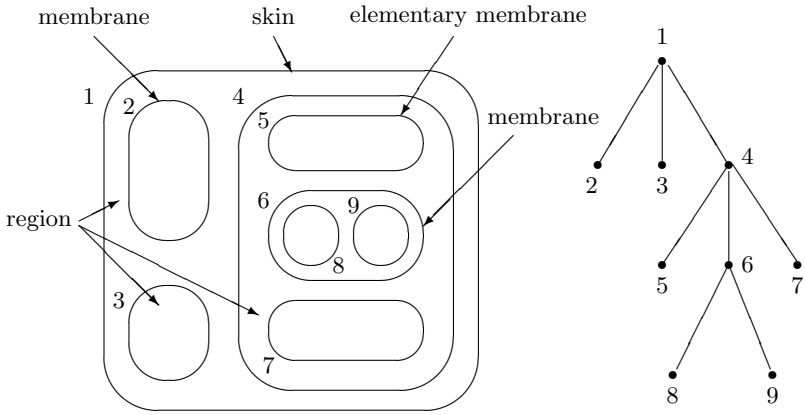

Fig. 1. A membrane structure and its associated tree

The objects evolve by means of given *rules*, which are associated with the regions (the intuition is that each region has specific chemical reaction conditions, hence the rules from a region cannot necessarily act also elsewhere). These rules specify both object transformation and object transfer from a region to another one. The passing of an object through a membrane is called *communication*.

Here is a typical rule:

$$cabb \to caad_{out}d_{in_3},$$

with the following meaning: one copy of the *catalyst* c (note that it is reproduced after the "reaction") together with one copy of object a and two copies of object b react together and produce one copy of c, two of a, and two copies of object d; one of these latter objects is sent out of the region where the rule is applied, while the second copy is sent to the adjacently inner membrane with the label 3, if such a membrane exists; the objects c, a, a remain in the same membrane (it is supposed that they have associated the communication command *here*, but we do not explicitly write this indication); if there is no membrane with label 3 directly inside the membrane where the rule is to be applied, then the rule cannot be applied. By a command *out*, an object can be also sent out of the skin membrane, hence it leaves the system and never comes back.

Therefore, the rules perform a multiset processing; in the previous case, the multiset represented by *cabb* is subtracted from the multiset of objects in a given region, objects *caa* are added to the multiset in that region, while copies of d are added to the multisets in the upper and lower regions.

The rules are used in a *nondeterministic maximally parallel manner*: the objects to evolve and the rules to be applied to them are chosen in a nondeterministic manner, but after assigning objects to rules no further rule should be applicable to the remaining objects. Sometimes, a priority relation among rules is considered, hence the rules to be used and the objects to be processed are selected in such a way that only rules which have a maximal priority among the applicable rules are used.

Other features can be considered, such as the possibility to control the membrane thickness/permeability, but we will introduce them later.

The membrane structure together with the multisets of objects and the sets of evolution rules present in its regions constitute a *P system*. The membrane structure and the objects define a *configuration* of a given P system. By using the rules as suggested above, we can define *transitions* among configurations. A sequence of transitions is called a *computation*. We accept as successful computations only the *halting* ones, those which reach a configuration where no further rule can be applied.

With a successful computation we can associate a *result*, for instance, by counting the multiplicity of objects which have left the system during the computation. More precisely, we can use a P system for solving three types of tasks: as a *generative* device (start from an initial configuration and collect all vectors of natural numbers describing the multiplicities of objects which have left the system during all successful computations), as a *computing* device (start with some input placed in an initial configuration and read the output at the end of a successful computation, by considering the objects which have left the system), and as a *decidability* device (introduce a problem in an initial configuration and wait for the answer in a specified number of steps). Here we deal only with the first case. Many classes of P systems turn out to be computationally universal, able to generate exactly what Turing machines can recursively enumerate.

2 A More Formal Definition of a P System

A membrane structure can be mathematically represented by a tree, in the natural way, or by a string of matching parentheses. The tree of the structure in Figure 1 is given in the same figure, while the parenthetic representation of that structure is the following:

$$[_1\,[_2\,]_2\,[_3\,]_3\,[_4\,[_5\,]_5\,[_6\,[_8\,]_8\,[_9\,]_9\,]_6\,[_7\,]_7\,]_4\,]_1.$$

The tree representation makes possible considering various parameters, such as the *depth* of the membrane structure, and also suggests considering membrane structures of particular types (described by linear trees, star trees, etc).

A multiset over an alphabet $V = \{a_1, \ldots, a_n\}$ is a mapping μ from V to $\mathbf{N}$, the set of natural numbers, and it can be represented by any string $w \in V^*$ such that $\Psi_V(w) = (\mu(a_1), \ldots, \mu(a_n))$, where Ψ_V is the Parikh mapping associated with V. Operations with multisets are defined in the natural manner.

With these simple prerequisites, we can define a P system (of *degree* $m \geq 1$) as a construct

$$\Pi = (V, T, C, \mu, w_1, \ldots, w_m, (R_1, \rho_1), \ldots, (R_m, \rho_m)),$$

where:

(i) V is an alphabet; its elements are called *objects*;
(ii) $T \subseteq V$ (the *output* alphabet);
(iii) $C \subseteq V, C \cap T = \emptyset$ (*catalysts*);
(iv) μ is a membrane structure consisting of m membranes, with the membranes and the regions labeled in a one-to-one manner with elements of a given set H; in this section we use the labels $1, 2, \ldots, m$;
(v) $w_i, 1 \leq i \leq m$, are strings representing multisets over V associated with the regions $1, 2, \ldots, m$ of μ;
(vi) $R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* over V associated with the regions $1, 2, \ldots, m$ of μ; ρ_i is a partial order relation over R_i, $1 \leq i \leq m$, 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 $\{a_{here},\ a_{out},\ a_{in_j} \mid a \in V, 1 \leq j \leq m\}$, and δ is a special symbol not in V. The length of u is called *the radius* of the rule $u \to v$.

When presenting the evolution rules, the indication "here" is in general omitted.

If Π contains rules of radius greater than one, then we say that Π is a system *with cooperation*. Otherwise, it is a *non-cooperative* system. A particular class of cooperative systems is that of *catalytic* systems: the only rules of a radius greater than one are of the form $ca \to cv$, where $c \in C, a \in V - C$, and v contains no catalyst; moreover, no other evolution rules contain catalysts (there is no rule of the form $c \to v$ or $a \to v_1 c v_2$, for $c \in C$).

The $(m + 1)$-tuple $(\mu, w_1, \ldots, w_m)$ constitutes the *initial configuration* of Π. In general, any sequence $(\mu', w'_{i_1}, \ldots, w'_{i_k})$, with μ' a membrane structure obtained by removing from μ all membranes different from $i_1, \ldots, i_k$ (of course, the skin membrane is not removed), with w'_j strings over V, $1 \leq j \leq k$, and $\{i_1, \ldots, i_k\} \subseteq \{1, 2, \ldots, m\}$, is called a *configuration* of Π.

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

For two configurations $C_1 = (\mu', w'_{i_1}, \ldots, w'_{i_k})$, $C_2 = (\mu'', w''_{j_1}, \ldots, w''_{j_l})$, of Π we write $C_1 \Longrightarrow C_2$, 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 $R_{i_1}, \ldots, R_{i_k}$ in the following manner.

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 at least as large as specified by u, appear in 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 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 $u \to v$.

The result of using the rule is determined by v. If an object appears in v in the form a_{here}, then it will remain in the same region i_t. If an object appears in v in the form a_{out}, then a will exit membrane i_t and will become an element of the region which surrounds membrane i_t. In this way, it is possible that an object leaves the system: if it goes outside the skin of the system, then it never comes back. If an object appears in the form a_{in_q}, then a will be added to the multiset from membrane q, providing that the rule $u \to v$ was used in the region adjacent to membrane q. If a_{in_q} appears in v and 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 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 from membrane i_t is added (in the sense of multisets union) to the multiset associated with the region which was directly external to membrane i_t. We do not allow the dissolving of the skin membrane, because this means that the whole "cell" is lost, we do no longer have a correct configuration of the system.

All these operations are performed in parallel, for all possible applicable rules $u \to v$, for all occurrences of multisets u in the regions associated with the rules, for all regions at the same time. No contradiction appears because of multiple membrane dissolving, or because 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 when it is dissolved, thus a will remain in the region surrounding membrane i; that is, from the point of view of a, the effect of a_{in_i} in the region outside membrane i and δ in membrane i is a_{here}.

A sequence of transitions between configurations of a given P system Π is called a *computation* with respect to Π. A computation is *successful* if and only if it halts, that is, there is no rule applicable to the objects present in the last configuration. The result of a successful computation is $\Psi_T(w)$, where w describes the multiset of objects from T which have been sent out of the system during the

computation. The set of such vectors $\Psi_T(w)$ is denoted by $Ps(\Pi)$ (from "Parikh set") and we say that it is *generated* by Π.

3 Universality Results for Systems with Symbol-Objects

In this section we recall some results about the generative power of some variants of P systems working with symbol-objects. In many cases, characterizations of recursively enumerable sets of vectors of natural numbers (their family is denoted by $PsRE$) are obtained.

3.1 Further Features Used in P Systems

Before giving these results, we will specify some further ingredients which can be used in a P system. They are in general introduced with the aim of obtaining more "realistic" systems. For instance, instead of the powerful command in_j, which indicates the target of the destination membrane, we can consider weaker communication commands. The weakest one is to add no label to *in*: if an object a_{in} is introduced in some region of a system, then a will go to any of the adjacent lower membranes, nondeterministically chosen; if no inner membrane exists, then a rule which introduces a_{in} cannot be used.

An intermediate possibility is to associate both with objects and membranes *electrical charges*, indicated by $+, -, 0$ (positive, negative, neutral). The charges of membranes are given in the initial configuration and are not changed during computations, the charge of objects are given by the evolution rules, in the form $a \to b^+ d^-$. A charged object will immediately go into one of the directly lower membranes of the opposite polarization, nondeterministically chosen, the neutral objects remain in the same region or will exit it, according to the commands *here, out* associated with them.

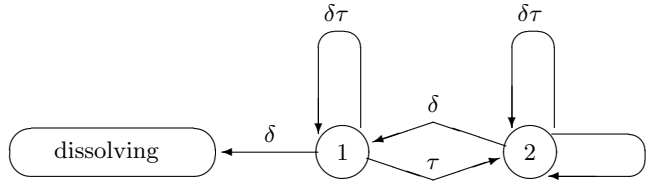

Fig. 2. The effect of actions δ, τ

Moreover, besides the action δ we can also consider an opposite action, denoted by τ, in order to control the membrane thickness (hence permeability). This is done as follows. Initially, all membranes are considered of thickness 1. If a rule in a membrane of thickness 1 introduces the symbol τ, then the membrane becomes of thickness 2. A membrane of thickness 2 does not become thicker by

using further rules which introduce the symbol τ, but no object can enter or exit it. If a rule which introduces the symbol δ is used in a membrane of thickness 1, then the membrane is dissolved; if the membrane had thickness 2, then it returns to thickness 1. If at the same step one uses rules which introduce both δ and τ in the same membrane, then the membrane does not change its thickness. These actions of the symbols δ, τ are illustrated in Figure 2.

No object can be communicated through a membrane of thickness two, hence rules which introduce commands *out, in*, requesting such communications, cannot be used. However, the communication has priority over changing the thickness: if at the same step an object should be communicated and a rule introduces the action τ, then the object is communicated and "afterthat" the membrane changes the thickness.

Also a variant of catalysts can be considered, with a "short term memory". Such catalysts (we call them *bi-stable*) have two states each, c and $\overline{c}$, and they can enter rules of the forms $ca \to \overline{c}v, \overline{c}a \to cv$ (always changing from c to $\overline{c}$ and back).

3.2 The Power of the Previous Systems

Consider now some notations. The family of sets of vectors of natural numbers $Ps(\Pi)$ generated by P systems with priority, catalysts, and the actions δ, τ, and of degree at most $m \geq 1$, using target indications of the form *here, out, in*, is denoted by $NP_m(Pri, Cat, i/o, \delta, \tau)$; when one of the features $\alpha \in \{Pri,\ Cat,\ \delta, \tau\}$ is not present, we replace it with $n\alpha$. We also write $2Cat$ instead of Cat when using bi-stable catalysts instead of usual catalysts.

Proofs of the following results can be found in [28], [12], [36]:

Theorem 1. $PsRE = NP_2(Pri, Cat, i/o, n\delta, n\tau) = NP_4(nPri, Cat, i/o, \delta, \tau) = NP_1(nPri, 2Cat,\ i/o, n\delta, n\tau)$.

It is an *open problem* whether or not systems of the type $(nPri, Cat, i/o, \delta, \tau)$ can characterize $PsRE$ when using less than four membranes.

3.3 Conditional Use of Rules

Starting from the observation that in the biochemistry of the cell certain reactions are enhanced/supressed by certain chemical compounds, such as enzymes, catalysts, hormons, in [1] one considers P systems with the rules applicability controlled by the contents of each region by means of certain *promoters* and *inhibitors*; these promoters/inhibitors are given as multisets of objects associated with given sets of rules. A rule from such a set (as usual, placed into a region) can be used in its region only if *all* the promoting objects are present, respectively, only if *not all* the inhibiting objects are present in that region.

Specifically, a *P system* (of degree $m \geq 1$), *with promoters* is a construct

$$\Pi = (V, \mu, w_1, \ldots, w_m, (R_{1,1}, p_{1,1}), \ldots, (R_{1,k_1}, p_{1,k_1}), \ldots, (R_{m,1}, p_{m,1}), \ldots, (R_{m,k_m}, p_{m,k_m})),$$

where all components are as usual in a P system (note that we use neither a terminal subset of V, nor catalysts) and $R_{i,1}, \ldots, R_{i,k_i}, k_i \geq 1$, are finite sets of *rules* present in region i of the system, while $p_{i,j}$ are strings over V, called *promoters*, $1 \leq i \leq m, 1 \leq j \leq k_i$. The rules are of the form $a \to v$, for $a \in V$, $v \in \{b_{tar} \mid b \in V, tar \in \{here, out, in\}\}^*$.

The rules are used as usual in P systems, in the maximally parallel manner, but *the rules from a set $R_{i,j}$ can be used only if the multiset represented by $p_{i,j}$ is present in region i*, for all $1 \leq i \leq m, 1 \leq j \leq k_i$.

An identical definition holds for systems with forbidding conditions, where the rules from $R_{i,j}$ are used only if the associated multiset $p_{i,j}$ is not included in the multiset of objects present in the region.

The maximum of $k_i, 1 \leq i \leq m$, is called the *promoter diversity* (resp., *inhibitor diversity*) of the system. The family of all sets $Ps(\Pi)$, computed as above by systems Π of degree at most $m \geq 1$ and with the promoter diversity not exceeding $k \geq 1$, is denoted by $PsP_m(i/o, prom_k)$. When using inhibitors, we replace $prom_k$ by $inhib_k$. We stress the fact that we do not use catalysts, priorities, or actions δ, τ, while the communication commands are of the form *here, out, in*. The following results are proved in [1]; they show a clear trade-off between the number of membranes and the promoter diversity of the used systems.

Theorem 2. $PsRE = PsP_6(i/o, prom_2) = PsP_4(i/o, prom_3) = PsP_3(i/o, prom_4) = PsP_1(i/o, prom_7) = PsP_3(i/o, inhib_6)$.

It is an *open problem* whether or not these results can be improved (for instance, using at the same time less membranes and a smaller promoter/inhibitor diversity).

4 P Systems with String-Objects

As we have mentioned in the Introduction, is also possible (this was considered already in [28]) to work with objects described by strings. The evolution rules should then be string processing rules, such as rewriting and splicing rules. As a result of a computation we can either consider the language of all strings computed by a system, or the number of strings produced by a system and sent out during a halting computation. In the first case one works with usual sets of strings (languages), not with multisets (each string is supposed to appear in exactly one copy), while in the latter case we have to work with multisets. We start by considering the set case.

4.1 Rewriting P Systems

We consider here the case of using string-objects processed by rewriting. Always we use only context-free rules, having associated target indications. Thus, the rules of our systems are of the form $(X \to v; tar)$, where $X \to v$ is a context-free rule over a given alphabet and $tar \in \{here, out, in\} \cup \{in_j \mid 1 \leq j \leq m\}$,

with the obvious meaning: the string produced by using this rule will go to the membrane indicated by *tar* (j is the label of a membrane). As above, we can also use priority relations among rules as well as the actions δ, τ, and then the rules are written in the form $(X \to x\alpha; tar)$, with $\alpha \in \{\delta, \tau\}$.

Formally, a *rewriting P system* is a construct

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

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

The language generated by Π is denoted by $L(\Pi)$ and it is defined as follows: we start from the initial configuration of the system and proceed iteratively, by transition steps performed by using the rules in parallel, to all strings which can be rewritten, obeying the priority relations; at each step, each string which can be rewritten must be rewritten, but this is done in a sequential manner, that is, only one rule is applied to each string; the actions δ, τ have the usual meaning; when the computation halts, we collect the terminal strings sent out of the system during the computation. We stress the fact that each string is processed by one rule only, the parallelism refers here to processing simultaneously all available strings by all applicable rules.

We denote by $RP_m(Pri, i/o, \delta, \tau)$ the family of languages generated by rewriting P systems of degree at most $m \geq 1$, using priorities, target indications of the form *here, out, in*, and actions δ, τ; as usual, we use *nPri*, $n\delta, n\tau$ when appropriate.

The following result is proved in [28] for the case of three membranes; the improvement to two membranes was given independently in [17] and [25].

Theorem 3. $RE = RP_2(Pri, i/o, n\delta, n\tau)$.

The powerful feature of using a priority relation can be avoided at the price of using membranes with a variable thickness. This was proved first in [39], [41], without a bound on the number of membranes, then the result has been improved in [12]:

Theorem 4. $RE = RP_4(nPri, i/o, \delta, \tau)$.

It is not known whether or not this result is optimal.

4.2 Splicing P Systems

The strings in a P system can also be processed by using the *splicing* operation introduced in [14] as a formal model of the DNA recombination under the influence of restriction enzymes and ligases (see a comprehensive information about splicing in [32]).

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

$$(x, y) \vdash_r (w, z) \text{ iff } x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,\ w = x_1u_1u_4y_2,\ z = y_1u_3u_2x_2,$$
$$\text{for some } x_1, x_2, y_1, y_2 \in V^*.$$

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

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

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

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

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

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

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

We denote by $SP_m(i/o)$ the family of languages $L(\Pi)$ generated by splicing P systems as above, of degree at most $m \geq 1$.

In [35] it was proved that $SP_3(i/o) = RE$; the result has been improved in [26] as follows.

Theorem 5. $RE = SP_2(i/o)$.

4.3 P Systems with Rewriting and Replication

Recently, in [18] a variant of rewriting P systems was considered, where rules of the form $X \to (u_1, tar_1)||\ldots||(u_n, tar_n)$ are used. When applying this rule to

a string, n strings are obtained, by replacing one occurrence of the symbol X by $u_1, \ldots, u_n$ and replicating the remaining part of the string; the n strings are sent to the membranes indicated by the targets $tar_1, \ldots, tar_n$, respectively. The definition of a replication rewriting system and of its generated language is as usual in P systems area.

The family of all languages $L(\Pi)$, computed as above by systems Π of degree at most $m \geq 1$ is denoted by $RRP_m(i/o)$.

The following result is from [20], where one solves the problem formulated as open in [18] whether or not the hierarchy on the number of membranes gives an infinite hierarchy (as expected, the answer is negative).

Theorem 6. $RE = RRP_6(i/o)$.

4.4 Rewriting P Systems with Conditional Communication

In this subsection, we recall the universality results from [2], where one considers a variant of rewriting P systems where the communication of strings is not controlled by the evolution rules, but it depends on the contents of the strings themselves. This is achieved by considering certain types of *permitting* and *forbidding* conditions, based on the symbols or the substrings (arbitrary, or prefixes/suffixes) which appear in a given string, or on the shape of the string.

First, let us mention that the set of symbols appearing in a string $x \in V^*$ is denoted by $alph(x)$ and the set of substrings of x is denoted by $Sub(x)$. A regular expression is said to be *elementary* if it has the star height at most one and uses the union at most for symbols in the alphabet.

A *rewriting P system* (of degree $m \geq 1$) with conditional communication is a construct

$$\Pi = (V, T, \mu, L_1, \ldots, L_m, R_1, P_1, F_1, \ldots, R_m, P_m, F_m),$$

where the components $V, T, \mu, L_1, \ldots, L_m$ are as usual in rewriting P systems, $R_1, \ldots, R_m$ are finite sets of context-free *rules* over V present in region i of the system, P_i are *permitting conditions* and F_i are *forbidding conditions* associated with region $i, 1 \leq i \leq m$.

The conditions can be of the following forms:

1. *empty*: no restriction is imposed on strings, they exit the current membrane or enter any of the directly inner membrane freely.
2. *symbols checking*: each P_i is a set of pairs $(a, \alpha), \alpha \in \{in, out\}$, for $a \in V$, and each F_i is a set of pairs $(b, not\alpha), \alpha \in \{in, out\}$, for $b \in V$; a string w can go to a lower membrane only if there is a pair $(a, in) \in P_i$ with $a \in alph(w)$, and for each $(b, notin) \in F_i$ we have $b \notin alph(w)$; similarly, for sending the string w out of membrane i it is necessary to have $a \in alph(w)$ for at least one pair $(a, out) \in P_i$ and $b \notin alph(w)$ for all $(b, notout) \in F_i$.
3. *substrings checking*: each P_i is a set of pairs $(u, \alpha), \alpha \in \{in, out\}$, for $u \in V^+$, and each F_i is a set of pairs $(v, not\alpha), \alpha \in \{in, out\}$, for $v \in V^+$; a string w can go to a lower membrane only if there is a pair $(u, in) \in P_i$ with

$u \in Sub(w)$, and for each $(v, notin) \in F_i$ we have $v \notin Sub(w)$; similarly, for sending the string w out of membrane i it is necessary to have $u \in Sub(w)$ for at least one pair $(u, out) \in P_i$ and $v \notin Sub(w)$ for all $(v, notout) \in F_i$.

4. *prefix/suffix checking*: exactly as in the case of substrings checking, with the checked string being a prefix or a suffix of the string to be communicated.
5. *shape checking*: each P_i is a set of pairs $(e, \alpha), \alpha \in \{in, out\}$, where e is an elementary regular expression over V, and each F_i is a set of pairs $(f, not\alpha), \alpha \in \{in, out\}$, where f is an elementary regular expression over V; a string w can go to a lower membrane only if there is a pair $(e, in) \in P_i$ with $w \in L(e)$, and for each pair $(f, notin) \in F_i$ we have $w \notin L(f)$; similarly, for sending the string w out of membrane i it is necessary to have $w \in L(e)$ for at least one pair $(e, out) \in P_i$ and $w \notin L(f)$ for all $(f, notout) \in F_i$.

We say that we have conditions of the types *empty, symb, sub$_k$, pref$_k$, suff$_k$, patt*, respectively, where k is the length of the longest string in all P_i, F_i.

The transitions in a system as above are defined in the usual way. In each region, each string which can be rewritten is rewritten by a rule from that region. Each string obtained in this way is checked against the conditions P_i, F_i from that region. If it fulfills the requested conditions, then it will be immediately sent out of the membrane or to an inner membrane, if any exists; if it fulfills both *in* and *out* conditions, then it is sent either out of the membrane or to a lower membrane, nondeterministically choosing the direction. If a string does not fulfill any condition, or it fulfills only *in* conditions and there is no inner membrane, then the string remains in the same region. A string which is rewritten and a string which is sent to another membrane is "consumed", we do not have a copy of it at the next step in the same membrane. If a string cannot be rewritten, then it is directly checked against the communication conditions, and, as above, it leaves the membrane (or remains inside forever) depending on the result of this checking.

That is, the rewriting has priority over communication: we first try to rewrite a string and only after that we try to communicate the result of the rewriting or the string itself if no rewriting is possible on it.

The family of all languages $L(\Pi)$, computed as above by systems Π of degree at most $m \geq 1$, with permitting conditions of type α, and forbidding conditions of type β, is denoted by $RP_m(\alpha, \beta)$, $\alpha, \beta \in \{$*empty, symb, patt*$\} \cup \{sub_k, pref_k,$ *suff*$_k \mid k \geq 1\}$. When we will use both prefix and suffix checking (each condition string can be checked both as a prefix or as a suffix, that is, we do not separately give sets of prefixes and sets of suffixes), then we indicate this by *prefsuff*$_k$. If the degree of the systems is not bounded, then the subscript m is replaced by $*$.

Proofs of the following results can be found in [2]. Again, a clear trade-off between the number of membranes and the power of the communication conditions is found. We do not know whether or not these results are optimal (for instance, as the number of used membranes); in particular, we do not have a proof of the fact that a reduced number of membranes suffice also when checking prefixes and suffixes, but we *conjecture* that such a result holds true.

Theorem 7. $RE = RP_2(patt, empty) = RP_2(empty, sub_2) = RP_4(sub_2, symb) = RP_6(symb, empty) = RP_*(prefsuff_2, empty)$.

4.5 P Systems with Leftmost Rewriting

Following [7], we now consider a restriction in the use of rules of a rewriting P system, of a language-theoretic nature: any string is rewritten in the leftmost position which can be rewritten by a rule from its region. That is, we examine the symbols of the string, step by step, from left to right; the first one which can be rewritten by a rule from the region of the string is rewritten. If there are several rules with the same left hand symbol, any one is chosen.

We denote by $L_{left}(\Pi)$ the language generated by a system Π in this way and by $RP_m(left), m \geq 1$, we denote the family of all such languages, generated by systems with at most m membranes. In view of the previous results, the following theorem is expected (but whether or not it is optimal in the number of membranes remains as an *open problem*):

Theorem 8. $RE = RP_6(left)$.

4.6 P Systems with Worm-Objects

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

The following four operations were considered in [5] (they are slight variants of the operations used in [38]): *rewriting* (called *point mutation* in [5] and [38]), *replication* (as in Subsection 4.3, but always producing only two strings), *splitting* (if $a \in V$ and $u_1, u_2 \in V^+$, then $r : a \to u_1|u_2$ is called a *splitting rule* and we define the operation $x_1 a x_2 \Longrightarrow_r (x_1 u_1, u_2 x_2)$), *recombination/crossovering* (for a string $z \in V^+$ we define the operation $(x_1 z x_2, y_1 z y_2) \Longrightarrow_z (x_1 z y_2, y_1 z x_2)$).

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

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

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

where:

- V is an alphabet;
- μ is a membrane structure of degree m (with the membranes labeled by $1, 2, \ldots, m$);

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

The transitions are defined as usual in P systems area, according to the following specific rules: A string which enters an operation is "consumed" by that operation, its multiplicity is decreased by one. The multiplicity of strings produced by an operation is accordingly increased. A string is processed by only one operation. For instance, we cannot apply two mutation rules, or a mutation rule and a replication one, to the same string. The strings resulting from an operation are communicated to the region specified by the target indications associated with the used rule. (Note that when we have two resulting strings, two targets are associated with the rule.)

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

In [5] it is proved that each recursively enumerable set of natural numbers (their family is denoted by nRE) can be computed by a P system as above; the result is improved in [23], where it is shown that the hierarchy on the number of membranes collapses:

Theorem 9. $nRE = NWP_6(tar)$.

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

Note the resemblance of P systems with worm objects with P systems with rewriting and replication (also Theorems 6 and 9 are similar), but also the essential difference between them: in Subsection 4.3 we have used sets of strings, while here we have worked with multisets (and we have generated sets of vectors of natural numbers). Following the same strategy as in the case of using both rewriting and replication, we can consider also other combinations of operations from those used in the case of worm-objects. The case of rewriting and crossovering was investigated in [22]. The work of such a P system is exactly as the work of a P system with worm-objects, only the way of defining the result of a computation is different: we consider the language of all strings which leave the system during the halting computations.

Let us denote by $RXP_m(i/o), m \geq 1$, the family of languages generated by such systems with at most m membranes, using as communication commands the indications *here, out, in* (but not priorities and actions δ, τ). Somewhat expected, we get one further characterization of recursively enumerable languages (the proof can be found in [22]).

Theorem 10. $RE = RXP_5(i/o)$.

It is an *open problem* whether or not the bound 5 is optimal.

5 P Systems with Active Membranes

Let us now consider P systems where the membranes themselves are involved in rules. Such systems were introduced in [30] with also the possibility of dividing membranes, and their universaity was proved for the general case. However, in [25] it was proven that membrane division is not necessary. Here we will consider this restricted case. (However, membrane division is crucial in solving NP-complete problems in polynomial – even linear – time, by making use of an exponential working space created by membrane division, but we do not deal here with this aspect. A survey of results of this type can be found in [31].)

A *P system with active membranes, in the restricted form*, is a construct

$$\Pi = (V, T, H, \mu, w_1, \ldots, w_m, R),$$

where:

(i) $m \geq 1$;
(ii) V is the alphabet of the system;
(iii) $T \subseteq V$ (terminal alphabet);
(iv) H is a finite set of *labels* for membranes;
(v) μ is a *membrane structure*, consisting of m membranes labeled with elements of H and having a neutral charge (all membranes are marked with 0);
(vi) $w_1, \ldots, w_m$ are strings over V, describing the *multisets of objects* placed in the m regions of μ;
(vii) R is a finite set of *rules*, of the following forms:

(a) $[_h a \to v]_h^\alpha$, for $h \in H,\ a \in V,\ v \in V^*,\ \alpha \in \{+, -, 0\}$ (object evolution rules),
(b) $a[_h]_h^\alpha \to [_h b]_h^\beta$, where $a, b \in V,\ h \in H,\ \alpha, \beta \in \{+, -, 0\}$ (an object is introduced in membrane h),
(c) $[_h a]_h^\alpha \to [_h]_h^\beta b$, for $h \in H,\ \alpha, \beta \in \{+, -, 0\},\ a, b \in V$ (an object is sent out of membrane h).

Also rules for dissolving and for dividing a membrane are considered in [30] (and in other subsequent papers), but we do not use such rules here.

The rules are used as customary in a P system, in a maximally parallel manner: in each time unit, all objects which can evolve, have to evolve. Each

copy of an object and each copy of a membrane can be used by only one rule, with the exception of rules of types (a), where we count only the involved object, not also the membrane. That is, if we have several objects a in a membrane i and a rule $[_i a \to v]_i^\alpha$, then we use this rule for all copies of a, irrespective how many they are; we do not consider that the membrane was used – note that its electrical charge is not changed. However, if we have a rule $[_i a]_i^\alpha \to [_i\,]_i^\beta b$, then this counts as using the membrane, no other rule of types (b) and (c) which involves the same membrane can be used at the same time.

As any other membrane, the skin membrane can be "electrically charged". During a computation, objects can leave the skin membrane (using rules of type (c)).

We denote by $N(\Pi)$ the set of all vectors of natural numbers computed as above by a P system Π. The family of all such sets of vectors, computed by systems with at most $m \geq 1$ membranes, is denoted by $PsAP_m$; when the number of membranes is not restricted, we replace the subscript m by $*$.

As announced above, in [25] it is proved that $PsRE = PsAP_*$ and the problem is formulated whether or not the hierarchy on the number of membranes collapses at a reasonable level. In [12] it is proved that four membranes suffice (it is an *open problem* whether or not this result is optimal).

Theorem 11. $PsRE = PsAP_4$.

6 Techniques Used in the Proofs of Universality Results

The most used tool (almost always used when dealing with P systems with symbol-objects) for proving universality results is the characterization of recursively enumerable languages by means of *matrix grammars with appearance checking* in the binary normal form. 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 $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either (1) $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or (2) $w_i = w_{i+1}$, 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.)

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When $F = \emptyset$ (hence we do not use the appearance checking feature), the generated family is denoted by MAT.

It is known that $CF \subset MAT \subset MAT_{ac} = RE$, the inclusions being proper.

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

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

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

According to [6], for each matrix grammar there is an equivalent matrix grammar in the binary normal form.

For an arbitrary matrix grammar $G = (N, T, S, M, F)$, let us denote by $ac(G)$ the cardinality of the set $\{A \in N \mid A \to \alpha \in F\}$. ¿From the construction in the proof of Lemma 1.3.7 in [6] one can see that if we start from a matrix grammar G and we get the grammar G' in the binary normal form, then $ac(G') = ac(G)$.

Improving the result from [27] (six nonterminals, all of them used in the appearance checking mode, suffice in order to characterize RE with matrix grammars), in [13] it was proved that four nonterminals are sufficient in order to characterize RE by matrix grammars and out of them only three are used in appearance checking rules. Of interest in the P area is another result from [13]: if the total number of nonterminals is not restricted, then each recursively enumerable language can be generated by a matrix grammar G such that $ac(G) \leq 2$.

Consequently, to the properties of a grammar G in the binary normal form we can add the fact that $ac(G) \leq 2$. We will say that this is *the strong binary normal form* for matrix grammars.

Starting from such a grammar G, in many cases a P system is constructed with some membranes simulating the matrices from G which do not contain rules to be used in the appearance checking mode, and some membranes associated with the symbols which are checked in the matrices which contain rules used in the appearance checking mode (that is why the number of nonterminals A which appear in rules of the form $A \to \#$ is so important). This leads to a reduced number of membranes, as seen in the theorems from the previous sections.

Other very useful tools used mainly in the proofs dealing with string-objects are the *normal forms* of Chomsky grammars known to generate all recursively enumerable languages. The most important are the *Kuroda normal form* and the *Geffert normal form*.

A type-0 grammar $G = (N, T, S, P)$ is said to be in the *Kuroda normal form* if the rules from P are of one of the following two forms: $A \to x, AB \to CD$, for $A, B, C, D \in N$ and $x \in (N \cup T)^*$ (that is, besides context-free rules we have only rules which replace two nonterminals by two nonterminals).

A type-0 grammar $G = (N, T, S, P)$ is said to be in the *Geffert normal form* if $N = \{S, A, B, C\}$, and the rules from P are of one of the following two forms:

$S \to xSy$, with $x, y \in (T \cup \{A, B, C\})^*$, and $ABC \to \lambda$ (that is, besides context-free rules we have only one non-context-free rule, the erasing one $ABC \to \lambda$, with A, B, C being the only nonterminals of G different from S).

At a more technical level, the proofs of universality start by a matrix grammar or a grammar in Kuroda or Gefffert normal form, and construct a P system which simulates it; the control of the correct behavior of the system is ensured by various tricks, directly related to the type of the system we construct. In most cases, because only halting computations are accepted, a "wrong" choice of rules or of communication of objects is prevented by introducing trap-symbols which are able to evolve forever; in the string case, if a terminal alphabet is used, then one introduces trap-symbols which just prevent the string to become terminal. Another useful trick is to control the communication possibilities; if a rule introduces an object which must be communicated to a lower membrane, then the communication should be possible (for instance, the thickness of the lower membranes is not increased). A nice way to control the appearance of a symbol to be checked by a rule of a matrix grammar is provided by replication: a copy of the string is sent to a membrane where nothing happens except that the computation goes forever in the case when the specified symbol appears in the string; if this is not the case, then the other copy of the string will continue a "correct" development. When we have to work on strings in the leftmost/rightmost manner, then the rotate-and-simulate technique from the splicing area is useful: we simulate the rules of the starting grammar in the ends of the strings generated by a P system, and the strings are circularly permuted in order to make possible such a simulation in any place of a sentential form of the grammar.

Still more precise proof techniques (for instance, for synchronizing the work of different membranes) can be found in the literature, but we do not recall them here.

7 Final Remarks

We have considered most of the variants of P systems with symbol-objects and with string-objects and we have recalled characterizations of the family of recursively enumerable sets of vectors of natural numbers or of the family of recursively enumerable languages. Many other variants, several of them leading to similar results, can be found in the literature. We only mention here the generalized P systems considered in [8], [10], [11], the carrier-based systems (where no object evolves, but only passes through membranes) [24], the P systems with valuations [21] (their power is not know yet), the systems also able to create membranes [15], which lead to a characterization of Parikh sets of ET0L languages, and the systems also taking into account the energy created/consumed by each evolution rule [34].

Note. The work of the second author was supported by a grant of NATO Science Committee, Spain, 2000–2001.

References

1. P. Bottoni, C. Martin-Vide, Gh. Păun, G. Rozenberg, Membrane systems with promoters/inhibitors, submitted, 2000.
2. P. Bottoni, A. Labella, C. Martin-Vide, Gh. Păun, Rewriting P systems with conditional communication, submitted, 2000.
3. D. Bray, Protein molecules as computational elements in living cells, *Nature*, 376 (1995), 307–312
4. C. Calude, Gh. Păun, *Computing with Cells and Atoms*, Taylor and Francis, London, 2000.
5. J. Castellanos, A. Rodriguez-Paton, Gh. Păun, Computing with membranes: P systems with worm-objects, *IEEE 7th. Intern. Conf. on String Processing and Information Retrieval, SPIRE 2000*, La Coruna, Spain, 64–74.
6. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
7. Cl. Ferretti, G. Mauri, Gh. Păun, Cl. Zandron, Further remarks on P systems with string-objects, submitted, 2000.
8. R. Freund, Generalized P systems, *Fundamentals of Computation Theory, FCT'99*, Iaşi, 1999 (G. Ciobanu, Gh. Păun, eds.), *LNCS* 1684, Springer, 1999, 281–292.
9. R. Freund, Generalized P systems with splicing and cutting/recombination, *Workshop on Formal Languages*, FCT'99, Iaşi, 1999, *Grammars*, 2, 3 (1999), 189–199.
10. R. Freund, Sequential P systems, *Pre-proc. Workshop on Multiset Processing*, Curtea de Argeş, Romania, 2000, and *Theorietag 2000; Workshop on New Computing Paradigms* (R. Freund, ed.), TU University Vienna, 2000, 177–183.
11. R. Freund, F. Freund, Molecular computing with generalized homogeneous P systems, *Proc. Conf. DNA6* (A. Condon, G. Rozenberg, eds.), Leiden, 2000, 113–125.
12. R. Freund, C. Martin-Vide, Gh. Păun, Computing with membranes: Three more collapsing hierarchies, submitted, 2000.
13. R. Freund, Gh. Păun, On the number of non-terminals in graph-controlled, programmed, and matrix grammars, submitted, 2000.
14. T. Head, Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology*, 49 (1987), 737–759.
15. M. Ito, C. Martin-Vide, Gh. Păun, A characterization of Parikh sets of ET0L languages in terms of P systems, submitted, 2000.
16. S. N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP-complete problems, *Romanian J. of Information Science and Technology*, 2, 4 (1999), 357–367.
17. S. N. Krishna, R. Rama, On the power of P systems with sequential and parallel rewriting, *Intern. J. Computer Math.*, 77, 1-2 (2000), 1–14.
18. S. N. Krishna, R. Rama, P systems with replicated rewriting, *J. Automata, Languages, Combinatorics*, to appear.
19. W. R. Loewenstein, *The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life*, Oxford Univ. Press, New York, 1999.
20. V. Manca, C. Martin-Vide, Gh. Păun, On the power of P systems with replicated rewriting, *J. Automata, Languages, Combinatorics*, to appear.
21. C. Martin-Vide, V. Mitrana, P systems with valuations, *Proc. Second Conf. Unconventional Models of Computing* (I. Antoniou, C. S. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 154–166.

22. C. Martin-Vide, Gh. Păun, String objects in P systems, *Proc. of Algebraic Systems, Formal Languages and Computations Workshop*, Kyoto, 2000, RIMS Kokyuroku, Kyoto Univ., 2000, 161–169.
23. C. Martin-Vide, Gh. Păun, Computing with membranes. One more collapsing hierarchy, *Bulletin of the EATCS*, 72 (2000), 183–187.
24. C. Martin-Vide, Gh. Păun, G. Rozenberg, Membrane systems with carriers, *Theoretical Compuer Sci.*, to appear.
25. A. Păun, On P systems with membrane division, *Proc. Second Conf. Unconventional Models of Computing* (I. Antoniou, C. S. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 187–201.
26. A. Păun, M. Păun, On the membrane computing based on splicing, in vol. *Words, Languages, Grammars* (C. Martin-Vide, V. Mitrana, eds.), Kluwer, Dordrecht, 2000, 409–422.
27. Gh. Păun, Six nonterminals are enough for generating each r.e. language by a matrix grammar, *Intern. J. Computer Math.*, 15 (1984), 23–37.
28. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (see also *Turku Center for Computer Science-TUCS Report* No 208, 1998, www.tucs.fi).
29. Gh. Păun, Computing with membranes – A variant: P systems with polarized membranes, *Intern. J. of Foundations of Computer Science*, 11, 1 (2000), 167–182.
30. Gh. Păun, P systems with active membranes: Attacking NP-complete problems, *J. Automata, Languages and Combinatorics*, 6, 1 (2001).
31. Gh. Păun, Computing with membranes; Attacking NP-complete problems, *Proc. Second Conf. Unconventional Models of Computing* (I. Antoniou, C. S. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 94–115.
32. Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Berlin, 1998.
33. Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, 41, 3 (2000), 259–266,
34. Gh. Păun, Y. Suzuki, H. Tanaka, P Systems with energy accounting, *Intern. J. Computer Math.*, to appear.
35. Gh. Păun, T. Yokomori, Membrane computing based on splicing, *Preliminary Proc. of Fifth Intern. Meeting on DNA Based Computers* (E. Winfree, D. Gifford, eds.), MIT, June 1999, 213–227.
36. Gh. Păun, S. Yu, On synchronization in P systems, *Fundamenta Informaticae*, 38, 4 (1999), 397–410.
37. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.
38. M. Sipper, Studying Artificial Life using a simple, general cellular model, *Artificial Life Journal*, 2, 1 (1995), 1–35.
39. Cl. Zandron, G. Mauri, Cl. Ferretti, Universality and normal forms on membrane systems, *Proc. Intern. Workshop Grammar Systems 2000* (R. Freund, A. Kelemenova, eds.), Bad Ischl, Austria, July 2000, 61–74.
40. Cl. Zandron, Cl. Ferretti, G. Mauri, Solving NP-complete problems using P systems with active membranes, *Proc. Second Conf. Unconventional Models of Computing* (I. Antoniou, C. S. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 289–301.
41. Cl. Zandron, Cl. Ferretti, G. Mauri, Using membrane features in P systems, *Pre-proc. Workshop on Multiset Processing*, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 2000, 296–320.