P Systems on Graphs of Restricted Forms[1]

Gheorghe PĂUN[a], Yasubumi SAKAKIBARA[b], Takashi YOKOMORI[c]

[a]Institute of Mathematics of the Romanian Academy
PO Box 1 – 764, 70700 Bucureşti, Romania
gpaun@imar.ro

[b]Department of Information Sciences, Tokyo Denki University
Ishizaka, Hatoyama-machi, Hiki-gun
Saitama 350-0394, Japan
yasu@j.dendai.ac.jp

[c]Department of Mathematics, School of Education
Waseda University, 1-6-1 Nishi-waseda, Shinjuku-ku
Tokyo 169-8050, Japan
yokomori@mn.waseda.ac.jp

Abstract

The P systems were recently introduced as distributed parallel computing models of a biochemical type. Multisets of objects are placed in a hierarchical structure of membranes and they evolve according to given rules, which are applied in a synchronous manner: at each step, all objects which can evolve, from all membranes, must evolve. The membrane structure can be described by a tree. We consider here restrictions on the form of this tree. We find that representations of permutation closures of recursively enumerable languages can be obtained by P systems with trees of a restricted form, a star or a line, providing that we use two bi-stable catalysts. If the number of catalysts is not bounded, then even systems with only one component suffice. We also consider P systems working on graphs which are not trees. A representation of permutation closures of recursively enumerable languages is obtained also for asymmetric graphs of a rather restricted form – a ring. A variant of the membrane structure is considered in this latter case in the form of a planar map (the regions of a membrane structure are now "countries" on the map, separated by "borders". In all cases, we use flip-flop catalysts, no priority, membranes (or borders) of a variable thickness, but a completely non-deterministic communication (the objects are only directed *up* or *down*, without any indication on the target membrane).

KEYWORDS: P systems, Molecular computing, Recursively enumerable languages, Matrix grammars

[1]Research supported by "Research for Future" Program no. JSPS-RFTF 96I00101, from the Japan Society for the Promotion of Science.

1 Introduction

The P systems are a class of distributed parallel computing devices of a biochemical type (so, they belong to the vivid area of Molecular Computing) which were recently introduced in [5]; an early survey can be found in [6].

In short, in the basic model one considers a *membrane structure* consisting of several cell-membranes which are hierarchically embedded in a main membrane, called the *skin* membrane. The membranes delimit *regions*, where we place *objects*, elements of a finite set (an alphabet; that is why we use as almost synonymous the terms *object* and *symbol*). The objects evolve according to given *evolution rules*, which are associated with the regions. An object can evolve independently of the other objects in the same region of the membrane structure, or in cooperation with other objects. In particular, we can consider *catalysts*, objects which evolve only together with other objects, but are not modified by the evolution (they just assist other objects to evolve). The evolution rules are given in the form of multisets transition rules, can be the subject of a given priority relation, and in their right hand members contain symbols $a_{here}, a_{out}, a_{in_j}$, where a is an object. The meaning is that one occurrence of the symbol a is produced and remains in the same region, is sent out of the respective membrane, or is sent to membrane j (which should be reachable from the region where the rule is applied), respectively. The membranes can be *dissolved*. When such an action takes place, all the objects of the dissolved membrane remain free in the membrane placed immediately outside, but the evolution rules of the dissolved membrane are lost. The skin membrane is never dissolved.

The application of evolution rules is done in a maximally parallel manner: at each step, all objects which can evolve should evolve.

Starting from an initial configuration and using the evolution rules, we get a *computation*. We consider a computation completed when it halts, no further rule can be applied. The multiplicity of objects present in a designated membrane in a halting configuration is the result of the computation. Thus, in this way we compute vectors of natural numbers.

Many variants are considered in [2], [5], [7], [9], [12]. In most of these cases, one gets computationally universal devices: all recursively enumerable sets of vectors of natural numbers can be computed. When membrane division is allowed, NP-complete problems can be solved in linear time; see [8], [4], [10].

Here we consider a combination of these variants, in general, looking for a "realistic" model.

First, we do not consider any priority relation among rules.

Second, like in [7], we use membranes of a variable thickness; initially, they are of thickness one; by an operation δ we can decrease the thickness and by an operation τ we can increase the thickness. Only membranes of thickness one and two are used; τ does not increase the thickness of a membrane of thickness two; when both δ and τ are used at the same time, in the same membrane, the membrane thickness is not changed. A membrane which reaches thickness zero is considered as dissolved, one of thickness one can be used for communicating symbols through it, but one of thickness two is opaque, no symbol can pass through it (the using of τ can be interpreted as the inhibition of all channels by which the membrane allows communication of objects).

Third, we use catalysts which can change their states, among two given states, as already considered in [12] (we say that they are bi-stable catalysts). This feature is powerful, because it can be used as a "short term memory", storing some useful information for one time unit. Thus, we are interested in restricting the number of used catalysts.

Fourth, as in [9], with a computation we associate an external output: the symbols which leave the system are collected in the order they leave the system and a string is formed with them; if two or more symbols leave the system at the same time, then any ordering of them is taken into consideration. In this way, a language is associated with a system.

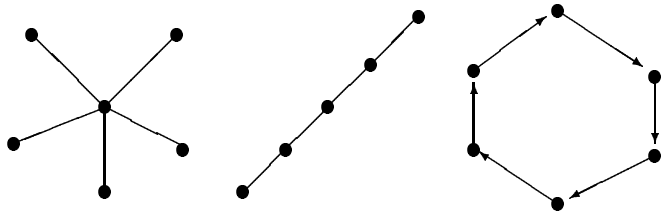

Figure 1. A star tree, a linear tree, a ring.

In [7] one communicates symbols by using the target indications *here, out*, and "electrical charges" instead of indications of the form in_j: each membrane has associated a sign $+$ or $-$ and also objects marked with these signs are introduced; a symbol marked with $+$ will go to any adjacent membrane marked with $-$, non-deterministically choosing the target membrane, while objects marked with $-$ will go to any adjacent membrane marked with $+$.

Here we use a still weaker variant, where only the indication to go up (indicated by *out*) or down (indicated by *in*) is given, without any further restriction on the target membrane in the case of sending an object down; this is like having all membranes marked with the same sign, say $+$, and all objects which have to be moved down marked with the opposite sign, $-$.

A membrane structure can be described by a tree (this has already been pointed out in [9]). We investigate here the problem of the influence of the shape of this tree on the power of P systems. This is a natural problem both from a mathematical point of view and from a practical point of view: it is possible that certain simple shapes of the membrane structure (of the underlying tree) are easier to implement than general shapes. Such simple shapes were already found for systems of the general type (using priority and target indications). Namely, representations of the permutation closure of the recursively enumerable languages are found by P systems with only a few membranes [5], while, for a variant for which such a result is not known, a normal form theorem was directly given in [13]: any system of a certain type (with priority, without catalysts, with δ) can be simulated by systems of depth two, that is, with a star tree.

We prove here three similar normal form results, for representations of permutations of recursively enumerable languages. Star trees, linear trees, as well as ring graphs suffice even when only two catalysts are used; in the last case, the membrane structure is allowed

to be of a different type, namely as a map (corresponding to a connex planar graph).

The shape of these three types of graphs, the simplest ones we can consider, is illustrated in Figure 1.

When the number of catalysts is not restricted, systems with only one component already characterize the permutation closures of recursively enumerable languages.

It is worth emphasizing that although we present here *representations* of the permutation closure of recursively enumerable languages, we can get *characterizations* of the Parikh sets of such languages (by Turing-Church thesis or by a direct proof), or, if we prefer an equivalent formulation, we can get *characterizations* of recursively enumerable relations of natural numbers.

2 Some Formal Language Theory Prerequisites

We only give some notations and notions; for further elements of formal language theory used in the sequel we refer to [14].

For an alphabet V we denote by V^* the set of all strings of elements in V; the empty string is denoted by λ. Any subset of V^* is called a language over V.

For a language L, we denote by $p(L)$ the permutation closure of L. For a family FL of languages we denote by pFL the family $\{p(L) \mid L \in FL\}$, of permutation closures of languages in FL. By RE we denote the family of recursively enumerable languages.

In all the proofs of the results from the subsequent sections we use the notion of a matrix grammar.

A *matrix grammar with appearance checking* is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (we say that N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If $F = \emptyset$, then the grammar is said to be without appearance checking (and F is no longer mentioned).

We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$. The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When we use only grammars without appearance checking, then the obtained family is denoted by MAT.

It is known that $MAT \subset MAT_{ac} = RE$ and that each one-letter language in the family MAT is regular, [3]. Further details about matrix grammars can be found in [1] and in [14].

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, at the last step of a derivation.

According to Lemma 1.3.7 in [1], for each matrix grammar (with appearance checking) there is an equivalent matrix grammar (with appearance checking) in the binary normal form.

3 P Systems

We do not give here a completely formal definition of a P system; the reader is referred to [5] and [6] for details.

A *membrane structure* is a construct consisting of several *membranes* placed in a unique "skin" membrane; we identify a membrane structure with a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane. Graphically, a membrane structure is represented by a Venn diagram.

Each membrane identifies a *region*, delimited by it and the membranes immediately inside it (if any). A membrane without any other membrane inside it is said to be *elementary*.

If in the regions delimited by the membranes we place multisets of objects from a specified finite set V, as well as evolution rules for these objects, then we obtain a *P system*. We introduce here the variant we start with: without priority, with catalysts, with polarized membranes of a variable thickness.

Such a system of degree $m, m \geq 1$, is a construct

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

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 (labeled, for reference only, with $1, 2, \ldots, m$); each membrane is marked with one of the signs $+, -, 0$ (these markers are written as superscripts of the right square bracket representing the membrane, e.g., $[_1[_2\,]_2^+]_1^0$);

(v) $w_i, 1 \leq i \leq m$, are strings over V associated with the regions $1, 2, \ldots, m$ of μ; they represent multisets of objects present in the regions of μ (the multiplicity of a symbol in a region is given by the number of occurrences of this symbol in the string corresponding to that region);

(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.
The evolution rules are of the forms $a \to v$ or $ca \to cv$, where a is an object from $V - C$ and $v = v'$ or $v = v'\delta$ or $v = v'\tau$, where v' is a string over

$$(V - C) \cup \{a^\alpha \mid a \in V - C, \alpha \in \{+, -\}\} \cup \{a_{out} \mid a \in V - C\},$$

and δ, τ are special symbols not in V.

The membrane structure μ and the multisets represented by $w_1, \ldots, w_n$ constitute the *initial configuration* of the system. In the initial configuration, all membranes are considered *of thickness* 1. We can pass from a configuration to another one by using the evolution rules. This is done in parallel: all objects, from all membranes, which can be the subject of local evolution rules, should evolve simultaneously.

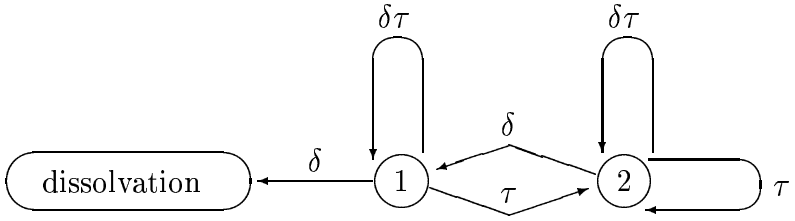

Figure 2. The effect of actions δ, τ.

The use of a rule $ca \to cv$ (similarly for $a \to v$) in a region with a multiset represented by a string w means to remove from w one copy of a and one of c, providing that such copies exist, then to follow the prescriptions of v: if an object appears in v unmarked with one of $+, -, out$, then it remains in the same region; if we have a_{out} and the membrane has thickness 1, then this copy of the object a will be moved in the region placed immediately outside; if we have a^+ (or a^-), then this copy of a is introduced in one of the membranes marked with $-$ (respectively $+$), adjacent to the region of the rule $ca \to cv$ (if no such a membrane exists, then the rule cannot be applied), and of thickness one; if the special symbol δ appears in v and the membrane where we work has thickness 1, then this membrane is dissolved; in this way, all the objects in this region become elements of the region placed immediately outside, while the rules of the dissolved membrane are removed.

The symbols δ, τ change the thickness of the membranes where they are produced as suggested by Figure 2.

That is, δ decreases the thickness and τ increases it; when both these symbols are introduced in the same region (of course, by different rules), the corresponding membrane preserves its thickness. The thickness of a membrane of thickness 2 is not further increased; once dissolved, a membrane is no longer recreated.

The communication of objects has priority over the actions δ, τ, in the sense that if at the same step one both sends a symbol through a membrane and one changes the thickness of that membrane, then one first transmits the symbol and after that one changes the thickness.

The rules are applied in parallel (the time is marked in discrete steps, by a unique clock for the whole system), an object introduced by a rule cannot evolve at the same step by means of another rule.

Note that the catalysts can pass from a region to another one only by membrane dissolving actions.

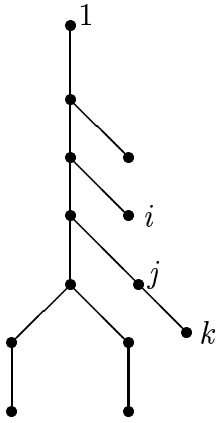

Figure 3. The shape of the tree in the proof of Theorem 1.

A sequence of transitions between configurations of a given P system Π, starting from the initial configuration, 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 consists of the objects from T ejected from the skin membrane, in the order they are ejected. Using these objects, we form a string. When several objects are ejected at the same time, any permutation of them is considered. In this way, a string or a set of strings is associated with each computation, that is, a language is associated with the system. (Note that the objects which remain in the system, as well as the objects which exit the system but are not elements of T are ignored.)

We denote by $L(\Pi)$ the language computed by Π in the way described above and by $LP_m^{\pm}(Cat, \delta, \tau)$ the family of languages generated by P systems as above, of degree at most m, using catalysts and both actions indicated by δ, τ; when one of the features $\alpha \in \{Cat, \delta, \tau\}$ is not used, we write $n\alpha$ instead of α. If systems of an arbitrary degree are

used, then the subscript m is removed. (The superscript $\pm$ is used in order to distinguish the present variant, with polarized membranes, from those in [5], [9], [2], where the objects are communicated by indicating the label of the target membrane.)

In [7] it is proved that $LP^{\pm}(Cat, \delta, \tau)$ equals the family of one letter recursively enumerable languages. By slightly modifying the proof, we get the following result:

Theorem 1. $pRE \subseteq LP^{\pm}(Cat, \delta, \tau)$.

It has been pointed out in [9] that a P system can also be considered as a device using a tree as the underlying structure: each node is associated with a membrane (the skin membrane is the root, the elementary membranes are leaves); moving an object from a region to another one means to move it from a node to a neighboring node (to its parent in the case of *out* and to one of its children in the case of *in*); dissolving a membrane means to remove the corresponding node and to link its children to its parent; the result is read in the root, as the sequence of symbols sent out.

Although this representation of a P system is mathematically attractive, it loses the biochemical intuition of the model. Thus, whenever we shall use it, we shall try to "implement" the tree/the graph also in a planar, intuitive, manner (for instance, as a map, in the case of planar graphs).

The tree describing the membrane structure of the system involved in the proof of Theorem 1 from [7] is of the form as indicated in Figure 3. The proof starts from a matrix grammar with appearance checking in the binary normal form. The number of nodes depends on the number of matrices in this grammar; branches of type i are associated with matrices used in the non-appearance checking manner and branches of type j–k are associated with matrices containing rules used in the appearance checking manner.

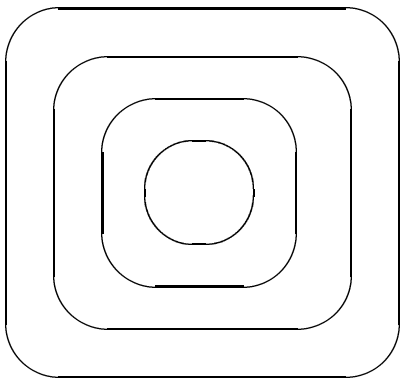

Figure 4. A membrane structure described by a linear tree.

Our investigations here start from the question whether or not we can obtain the same result, but using a tree of a simpler form, for instance, a linear one, without any branch[2]. Note that the membrane structure described by such a tree is of the form in Figure 4.

We do not have an answer to this question for P systems as above, but we will obtain

[2]In a different context, this question was also formulated to one of us by John McCaskill, during the Molecular Computing Workshop held in Leiden, in February 1999.

such a normal form result providing that a slightly more powerful type of catalysts is used, namely allowing them to have a "short memory": each catalyst c is allowed to have two states, c and $\bar{c}$; the rules involving catalysts always switch among these states. Thus, the allowed rules are of the forms: $ca \to \bar{c}v$, and $\bar{c}a \to cv$. Following [12], we say that we use *flip-flop* catalysts. We indicate the use of such bi-stable catalysts by writing $2Cat$ instead of Cat. It is clear that $LP^{\pm}(Cat, \delta, \tau) \subseteq LP^{\pm}(2Cat, \delta, \tau)$ (each rule $ca \to cv$ is replaced by the rules $ca \to \bar{c}v, \bar{c}a \to cv$, so that the role of c is alternately played by c and $c\bar{c}$). If the number of catalysts is bounded by a constant r, then the corresponding families of languages are denoted by $LP_m^{\pm}(2Cat_r, \delta, \tau)$, possibly without the subscript m.

We also look for an "orthogonal normal form", where the tree is of a minimal depth and of a maximal width, a star. This corresponds to P systems of depth two; a normal form theorem of this form was proved in [13] for the case of using priorities and δ, but without catalysts and τ, and, very important, using target indications of the form in_j for objects communication. In the same framework as before – with bi-stable catalysts and δ, τ operations – we prove here a similar theorem.

There is here an important point. In systems of the type described in Figure 4 there is no need to give any target indication, always we have exactly one lower membrane (excepting the central membrane) and one upper membrane (excepting the skin membrane). Thus, when we want to communicate, we can only indicate *in* and *out*, without any other information. (Specifically, we write a_{in}, a_{out} if we want to send a into any lower level membrane or out the current membrane, respectively.) What about using such loose indications in systems with a non-linear graph and allowing to the symbols to go to any membrane accessible in the indicated direction? (Of course, always an object can pass through one membrane only.) Surprisingly enough, in our setup this does not decrease the power; furthermore, a representation of pRE is obtained by systems using a membrane structure of depth two, hence with a star graph. Note that in the case of a star, the ambiguity of an indication of the form a_{in} introduced by a rule in the skin membrane is maximal, all second level membranes can be the target of the symbol a.

When dealing with only *in*/*out* indications, no sign $+, -$ will be associated with objects and membranes; the corresponding families are denoted by $LP^{i/o}(2Cat, \alpha, \beta), \alpha \in \{\delta, n\delta\}, \beta \in \{\tau, n\tau\}$.

4 Three Normal Form Theorems

Let us first prove the fact that the bi-stable catalysts are very poverful (if their number is not restricted).

Theorem 2. $pRE \subseteq LP_1^{i/o}(2Cat, n\delta, n\tau)$.

Proof. We use the equality $MAT_{ac} = RE$. Let us consider a matrix grammar with appearance checking, $G = (N, T, S, M, F)$, in the binary normal form, that is, with $N = N_1 \cup N_2 \cup \{S, \#\}$ and matrices of the four types specified in Section 2. Assume that the two-rules matrices of M are labeled in a one-to-one manner by $m_1, \ldots, m_n$; that is, we assume that we have n matrices of this type.

We construct the P system (of degree one)

$$\Pi = (V, T, C, \mu, w_1, R_1),$$

with

$$\begin{aligned} V &= N_1 \cup N_2 \cup T \cup C \cup \{b, d, e, e', e'', f, \dagger\} \\ &\cup \{e_i \mid 1 \leq i \leq n\} \cup \{X' \mid X \in N_1\}, \\ C &= \{c_i, \bar{c}_i \mid 0 \leq i \leq n\}, \\ \mu &= [_1\,]_1, \\ w_1 &= XAbec_0c_1c_2 \ldots c_n, \text{ for } (S \to XA) \text{ the initial matrix of } G, \end{aligned}$$

and R_1 containing the following rules:

1. For each matrix m_i of the form $(X_i \to z_i, A_i \to x_i)$, with $X_i \in N_1, z_i \in N_1 \cup \{\lambda\}$, $A_i \in N_2$, and $x_i \in (N_2 \cup T)^*$, we consider the rules:

   $X_i \to X_i'$,
   $c_i X_i' \to \bar{c}_i$,
   $\bar{c}_i A_i \to c_i e_i$,
   $e_i \to z_i x_i$,
   $\bar{c}_i b \to c_i \dagger$.

2. For each matrix m_i of the form $(X_i \to Y_i, A_i \to \#)$, with $X_i, Y_i \in N_1, A_i \in N_2$, we consider the rules:

   $X_i \to X_i'$,
   $c_i X_i' \to \bar{c}_i f$,
   $\bar{c}_i A_i \to c_i \dagger$,
   $\bar{c}_i d \to c_i Y_i$.

3. Moreover, we consider the following rules:

   $f \to d$,
   $a \to a_{out}$, for all $a \in T$,
   $c_0 e \to \bar{c}_0 e'$,
   $\bar{c}_0 B \to c_0 B$, for all $B \in N_2$.
   $c_0 e' \to \bar{c}_0 e''$,
   $\bar{c}_0 e'' \to c_0 e$,
   $e'' \to \dagger$.

The system works as follows.

At each moment, at most one symbol from N_1 is present, therefore at most one catalyst $c_i, 1 \leq i \leq n$, can be used. At the first step, the symbol from N_1 is primed.

If we apply a rule $c_i X_i' \to \bar{c}_i$, associated with a matrix $m_i : (X_i \to z_i, A_i \to x_i)$, hence not used in the appearance checking mode, then at the next step we have to use the rule

$\bar{c}_i A_i \to c_i e_i$, otherwise the trap-object $\dagger$ is introduced and the computation will never stop. At the fourth step, the symbols of $Y_i x_i$ are introduced. Thus, the matrix m_i is correctly simulated (both its rules are simulated). The operation takes four steps.

If we apply a rule $c_i X_i' \to \bar{c}_i f$, associated with a matrix $m_i : (X_i \to Y_i, A_i \to \#)$, hence used in the appearance checking mode, then at the next step we either apply the rule $\bar{c}_i A_i \to c_i \dagger$, if A_i is present, or the catalyst $\bar{c}_i$ is not modified, if A_i is not present. In the first case, the computation never halts. The symbol f is immediately transformed in d; thus, at the third step, if $\bar{c}_i$ is still present, then the rule $\bar{c}_i d \to c_i Y_i$ can be used. In the former case the computation will never stop, in the latter case the matrix is correctly simulated. The simulation of a matrix used in the appearance checking manner also takes four steps.

The process can be iterated. Note that in both cases the symbol Y_i is available only after completing the simulation of a matrix and that no symbol from N_2 can be processed if we do not first process a symbol from N_1.

The terminal symbols can be sent out of the system at any moment, so any permutation of a string can be obtained. As long as any symbol $B \in N_2$ is present, the computation can continue. This is ensured by the rules which involve the catalyst c_0: in cycles of four steps, at step two, a symbol $B \in N_2$ is used by the rule $\bar{c}_0 B \to c_0 B$. Note that the simulation of matrices of any type, with or without appearance checking, uses symbols from N_2 at the third step, hence the simulation is not confused by the use of rules in group 3. Moreover, at step four we have to use the rule $\bar{c}_0 e'' \to c_0 e$ (instead of $\bar{c}_0 B \to c_0 B$ for some $B \in N_2$), otherwise the trap-object is introduced by the rule $e'' \to \dagger$. Therefore, on the one hand, we have to send out all terminal symbols, on the other hand, a computation stops only if it coresponds to a terminal derivation in G.

Consequently, $p(L(G)) = L(\Pi)$. □

In the previous construction, the number of catalysts can be arbitrarily large, it depends on the number of matrices in the starting grammar. If we bound the number of catalysts we use, then we obtain a similar result, but without having a bound on the number of membranes. Thus, a trade-off between the number of membranes and the number of catalysts seems to hold.

Theorem 3. *$pRE \subseteq LP^{i/o}(2Cat_2, \delta, \tau)$; moreover, P systems with a linear underlying tree suffice.*

Proof. As in the previous proof, we consider a matrix grammar $G = (N, T, S, M, F)$ in the binary normal form, with n matrices $m_1, \ldots, m_n$.

We construct the P system (of degree $n + 1$)

$$\Pi = (V, T, C, \mu, w_0, w_1, \ldots, w_n, R_0, R_1, \ldots, R_n)$$

(the skin membrane is labeled with 0) with the following components:

$$\begin{aligned} V = {} & N_1 \cup N_2 \cup T \cup C \cup \{d_1, d_2, d_1', d_2', d_1'', d_2'', e, \dagger\} \\ & \cup \{\alpha', \alpha'', \alpha''', \bar{\alpha} \mid \alpha \in N_1 \cup N_2\} \\ & \cup \{\alpha^{iv} \mid \alpha \in N_1 \cup N_2 \cup T\} \end{aligned}$$

$$\cup \{(\alpha, i) \mid \alpha \in N_1 \cup N_2, 1 \leq i \leq n\}$$
$$\cup \{f_i \mid 1 \leq i \leq n\},$$
$$C = \{c_1, c_2, \bar{c}_1, \bar{c}_2\},$$
$$\mu = [_0[_1[_2 \cdots [_{n-1}[_n \,]_n]_{n-1} \cdots]_2]_1]_0,$$
$$w_0 = XA, \text{ for } (S \to XA) \text{ being the initial matrix of } G,$$
$$w_i = c_1 c_2 f_i, \; 1 \leq i \leq n,$$

and with the sets $R_0, R_1, \ldots, R_n$ constructed as follows.

1. The set R_0 contains the following rules:

   1. $\alpha \to (\alpha, i)_{in}$, for $\alpha \in N_1 \cup N_2, 1 \leq i \leq n$
      (send down nonterminals, at random addresses indicated by the second component of (α, i)),
   2. $a \to a$,
      $a \to a_{out}$, for $a \in T$
      (send out terminals, at any time after having them in the skin membrane),
   3. $f_i \to \dagger$, for all $1 \leq i \leq n$,
      $\dagger \to \dagger$
      (trap-rules; once introduced, the symbol $\dagger$ can evolve for ever).

2. Each set $R_i, 1 \leq i \leq n$, contains the following rules (slight differences for the cases $i = 1$ and $i = n$ will be mentioned below):

   1. $(\alpha, j) \to (\alpha, j)_{in}$, for $\alpha \in N_1 \cup N_2, i < j \leq n$
      (send down nonterminals, to the addresses specified in the skin membrane),
   2. $(\alpha, i) \to \bar{\alpha}$
      (when a symbol sent to membrane i reaches this membrane, its barred version is introduced),
   3. $\bar{\alpha} \to \alpha\delta$, for $\alpha \in N_2$,
      $\bar{\beta} \to \beta\tau$, for $\beta \in N_1$
      (these rules check whether or not all nonterminals are present in the same place; this is a very important point of the construction – see below complete explanations),
   4. $f_j \to \dagger$, for all $i < j \leq n$,
      $\dagger \to \dagger$
      (if any symbol f_j with $j > i$ reaches membrane i, then the computation never stops),
   5. $\alpha^{iv} \to \alpha^{iv}_{out}$, for $\alpha \in N_1 \cup N_2 \cup T$
      (the symbols marked with iv are sent to the upper membrane).

3. In the set R_1, instead of rules of type 5 above we introduce the rules

4′. $\alpha^{iv} \to \alpha_{out}$, for $\alpha \in N_1 \cup N_2 \cup T$
(the symbols reaches the skin membrane without any marking).

In the set R_n no rule of type 1 above is introduced.

4. If the matrix m_i is of the form $(X \to z, A \to x)$, for some $X \in N_1, A \in N_2$, $x \in (N_2 \cup T)^*$, and $z \in N_1 \cup \{\lambda\}$ (we cover at the same time both the case of nonterminal and of terminal matrices which are not used in the appearance checking mode), then in R_i we also introduce the following rules:

   1. $\alpha \to \alpha'$, for $\alpha \in N_1 \cup N_2$,
   $c_1 X \to \bar{c}_1 h(z) d_1' e$,
   $c_2 A \to \bar{c}_2 h(x) d_2' e$,
   where h is the morphism defined by

$$h(\alpha) = \begin{cases} \alpha', & \text{if } \alpha \in N_1 \cup N_2, \\ \alpha_{out}^{iv}, & \text{if } \alpha \in T \end{cases}$$

   (we simulate the two rules of matrix m_i),

   2. $\alpha' \to \alpha''$, for $\alpha \in N_1 \cup N_2$,
   $\bar{c}_1 d_1' \to c_1 d_1''$,
   $\bar{c}_2 d_1' \to c_2 d_2''$,
   $c_1 e \to \bar{c}_1 \dagger$,
   $c_2 e \to \bar{c}_2 \dagger$,
   $c_2 d_1' \to \bar{c}_2 \dagger$,
   $c_1 d_2' \to \bar{c}_1 \dagger$
   (we check whether or not both rules were simulated; if only one of them was simulated, then the trap-symbol $\dagger$ is introduced and the computation never ends; see more complete explanations below),

   3. $\alpha'' \to \alpha'''$, for $\alpha \in N_1 \cup N_2$,
   $c_1 d_1'' \to \bar{c}_1$,
   $c_2 d_2'' \to \bar{c}_2$
   (the auxiliary symbols d_1, d_2, in their primed versions, are removed),

   4. $\alpha''' \to \alpha_{out}^{iv}$, for $\alpha \in N_1 \cup N_2$,
   $\bar{c}_1 e \to c_1$,
   $\bar{c}_2 e \to c_2$
   (also the two copies of the auxiliary symbol e are removed; the catalysts return to their non-barred state).

5. If the matrix m_i is of the form $(X \to Y, A \to \#)$, for $X, Y \in N_1, A \in N_2$ (hence with the second rule used in the appearance checking manner), then we introduce in R_i the following rules:

   1. $X \to Y_{out}^{iv}$,
   $Z \to \dagger$, for all $Z \in N_1 - \{X\}$,

2. $A \to \dagger$,

3. $\alpha \to \alpha_{out}^{iv}$, for all $\alpha \in N_2 - \{A\}$
   (all nonterminals go out, except A; if A is present, then the computation will never finish).

We claim that $p(L(G)) = L(\Pi)$, which would conclude the proof.

Let us examine the work of the system Π. As already sketched above, the skin membrane sends randomly the current nonterminal symbols to the inner membranes, by attaching to them target integers: each α is replaced with (α, i), for some $1 \leq i \leq n$. The membranes $1, 2, \ldots, n$ simulate the corresponding matrices $m_1, m_2, \ldots, m_n$.

The symbols (α, i) are sent down until reaching the membrane i. This means exactly i steps (counting also the step when the rules $\alpha \to (\alpha, i)_{in}$ were used). When (α, i) reaches membrane i, we introduce the barred variant of α. At the next step, $\bar{\alpha}$ is replaced by α, by rules in group 3. It is important to note that all symbols sent to membrane i arrive in this membrane at the same time.

Our aim is to simulate in Π derivations of G. Initially, we have in the skin membrane the symbols XA. If the simulation is correct, then always we will have in our system only one occurrence of a symbol from N_1. Assume that we start from such a situation, that is, from a multiset in membrane 0 which contains exactly one occurrence of a symbol from N_1.

Suppose that a nonterminal $A \in N_2$ has reached a membrane i (in the barred form), but the currently available symbol $X \in N_1$ is not present in the same membrane. Then, the rule $\bar{A} \to A\delta$ is used in membrane i, without also using a rule of the type $\bar{\beta} \to \beta\tau$, for some $\beta \in N_1$. In this way, the membrane is dissolved, the symbol f_i is left free in the upper membrane (or a membrane placed at a higher level, in the case when several membranes are dissolved at the same time). In the upper membrane (or any superior one) we can use the rule $f_i \to \dagger$ and the computation is never finished.

Consequently, all the nonterminals from N_2 present in the system must be together with the unique symbol from N_1. This means that *all the nonterminals are together*. Although the skin membrane uses the rules $\alpha \to (\alpha, i)_{in}$ randomly, the computation will continue in a correct way only when all symbols gets the same "address" i. This is a crucial observation for the good functioning of our system (and this trick will be used also in the proofs below), for instance, in the case of simulating the matrices with appearance checking rules.

Let us now look how the matrices of G are simulated.

Consider first the case of a matrix m_i of the form $(X \to Y, A \to \#)$, hence with the second rule used in the appearance checking manner. Assume that all nonterminals have reached membrane i. We can change X with Y_{out}^{iv} and continue correctly only if no occurrence of A is present, otherwise the trap-symbol $\dagger$ is introduced. If a symbol $Z \in N_1$ different from X is present (this means that membrane 0 has incorrectly guessed the membrane where the nonterminals are sent), then again the trap-symbol $\dagger$ is introduced. Thus, we either correctly simulate the matrix, or the computation will never end. Note here the important role of the maximal parallelism: if a rule *can* be applied to an available symbol, then it *must* be used.

Assume now that we are in a membrane i associated with a matrix $m_i = (X \to Y, A \to x)$. If we use only rules of the forms $\alpha \to \alpha', \alpha' \to \alpha'', \alpha'' \to \alpha''', \alpha''' \to \alpha^{iv}_{out}$, then we can return all the symbols unmodified to the skin membrane. In particular, we can simulate only one rule of the matrix, using only one rule from the pair $c_1X \to \bar{c}_1Y'd'_1e$, $c_2A \to \bar{c}_2h(x)d'_2e$. Assume that the first rule is used, the second not; the other case is symmetric. This means that in membrane i we have the symbols $\bar{c}_1, c_2, d'_1, e$. The symbol e can now be paired either with $\bar{c}_1$ or with c_2, that is one of the rules $\bar{c}_1e \to c_1, c_2e \to \bar{c}_2\dagger$ must be used. The second rule introduces the trap-symbol $\dagger$, hence the computation will never stop. If we use the first rule, then also the rule $c_2d'_1 \to \bar{c}_2\dagger$ must be used, hence again the trap-symbol $\dagger$ is introduced. Therefore, both rules $c_1X \to \bar{c}_1Y'd'_1e$, $c_2A \to \bar{c}_2h(x)d'_2e$ must be used. The catalysts (barred or not) will remove the auxiliary symbols d_1, d_2, e and will return to their non-barred forms. This is done in four steps and exactly at the same time all the nonterminals present in the membrane will be sent to the upper membrane, in the form α^{iv}. Such symbols will be push-up until reaching again the skin membrane.

The simulation of a terminal matrix $(X \to \lambda, A \to x)$ is done exactly in the same way.

Note that any terminal symbol is immediately sent up. In the skin membrane, each terminal can wait as long as we want (by using rules $a \to a$) or it can be sent out of the system (by rules $a \to a_{out}$). In this way, all permutations of a string in $L(G)$ can be obtained.

After simulating a terminal matrix, we remove the symbol from N_1, therefore if any symbol from N_2 is still present in the system, then the computation will never halt (such nonterminals will dissolve membranes, hence symbols f_i will be released in upper membranes). Thus, a computation in Π halts only if the corresponding derivation in G is a terminal one.

It is now clear that each derivation in G can be simulated in Π in such a way that any permutation of the terminal string generated by this derivation can be produced at the end of the computation and, conversely, all computations in Π which end in a correct way correspond to terminal derivations in G. □

The depth of the system in the previous proof (the height of the asociated tree) depends on the number of matrices in the grammar we start with. Let us now look for a graph of a minimal depth. A counterpart of the previous theorem can be obtained (even using no target indication when sending symbols from the skin membrane to the lower level membranes; in this moment, this is somewhat expected, taking into account the way of keeping together all the nonterminal symbols, by using the actions δ, τ).

Theorem 4. *Each language $p(L), L \in RE$, can be generated by a P system of type $(2Cat_2, \delta, \tau)$, communicating by using in/out indications only, and with a membrane structure of depth two.*

Proof. The proof is similar to that of Theorem 3, but because of the importance of this result, for the sake of completeness, we give the core construction with almost full details.

Start again from a matrix grammar with appearance checking, $G = (N, T, S, M, F)$, in the binary normal form, with n matrices, $m_1, \ldots, m_n$.

We construct the P system (of degree $n+1$)

$$\Pi = (V, T, C, \mu, w_0, w_1, \ldots, w_n, R_0, R_1, \ldots, R_n)$$

(the skin membrane is labeled with 0) with the following components:

$$\begin{aligned}
V &= N_1 \cup N_2 \cup T \cup C \cup \{d_1, d_2, d_1', d_2', d_1'', d_2'', e, f, \dagger\} \\
&\quad \cup \{\alpha', \alpha'', \alpha''', \bar{\alpha} \mid \alpha \in N_1 \cup N_2\} \\
C &= \{c_1, c_2, \bar{c}_1, \bar{c}_2\}, \\
\mu &= [_0[_1\,]_1 \ldots [_n\,]_n]_0, \\
w_0 &= XA, \text{ for } (S \to XA) \text{ being the initial matrix of } G, \\
w_i &= c_1 c_2 f, \ 1 \leq i \leq n,
\end{aligned}$$

and with the sets $R_0, R_1, \ldots, R_n$ constructed as follows.

1. R_0 contains the following rules:

   1. $\alpha \to \bar{\alpha}_{in}$, for $\alpha \in N_1 \cup N_2$
      (send down nonterminals, to any membrane $1, 2, \ldots, n$),
   2. $a \to a$,
      $a \to a_{out}$, for $a \in T$
      (send out terminals, at any time after having them in the skin membrane),
   3. $f \to \dagger$,
      $\dagger \to \dagger$
      (trap-rules).

2. Each set $R_i, 1 \leq i \leq n$, contains the following rules:

   1. $\bar{\alpha} \to \alpha\delta$, for $\alpha \in N_2$,
      $\bar{\beta} \to \beta\tau$, for $\beta \in N_1$
      (check whether or not all nonterminals are present in the same place),
   2. $\dagger \to \dagger$.

3. If the matrix m_i is of the form $(X \to z, A \to x)$, for some $X \in N_1, A \in N_2$, $x \in (N_2 \cup T)^*$, and $z \in N_1 \cup \{\lambda\}$, then in R_i we introduce the rules of types 4.1 – 4.3 from the previous proof, as well as the following rules:

   $\alpha''' \to \alpha_{out}$, for $\alpha \in N_1 \cup N_2$,
   $\bar{c}_1 e \to c_1$,
   $\bar{c}_2 e \to c_2$
   (also the auxiliary symbol e is removed; the catalysts return to their non-barred state).

4. If the matrix m_i is of the form $(X \to Y, A \to \#)$, for $X, Y \in N_1, A \in N_2$ (hence with the second rule used in the appearance checking manner), then we introduce to R_i the following rules:

1. $X \to Y_{out}$,
   $Z \to \dagger$, for all $Z \in N_1 - \{X\}$,
2. $A \to \dagger$,
3. $\alpha \to \alpha_{out}$, for all $\alpha \in N_2 - \{A\}$
   (all nonterminals go out, except A; if A is present, then the computation will never finish).

In the same way as in the proof of Theorem 3, we have the equality $p(L(G)) = L(\Pi)$. It is clear that the membrane structure of Π is of the desired type. □

5 P Systems on Asymmetric Graphs

The observation that a membrane structure corresponds to a tree and that we can compute directly on a tree suggests the following general idea: consider computing devices similar to P systems, using an arbitrary graph as an underlying structure. Some of the operations specific to P systems (using evolution rules in parallel, maybe subject to some priority relations, moving symbols from a node to another one, reading the result in a node or outside the graph) can be easily extended to such a general case, but others should be carefully defined. This is the case with the dissolving operation, δ, and with its dual, τ.

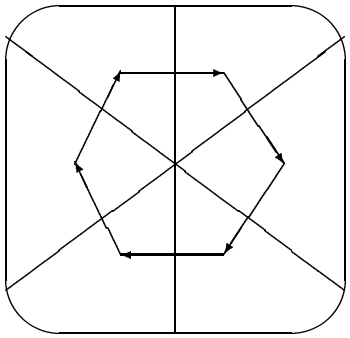

Figure 5 A possible support for a computation.

The idea of using graphs different from trees is not completely unrealistic from a (bio)chemical point of view. For instance, we can imagine a "membrane structure" as that in Figure 5, where six regions are delimited by six "walls". A computation based on multiset processing can be done also in such a framework (described by a graph in the form of a ring).

In general, any type of a planar map (hence planar graph) can be considered as an underlying structure for a P system, but again we look for structures which are as simple as possible. Here is an immediate question: what about the support of the type in Figure 5, with a ring as a graph?

There appears here an interesting point. In trees, each edge is supposed to be used for passing objects in both directions; in most cases, a symbol which reaches a node must

also leave the node. In the case of a ring graph (in general, for arbitrary graphs) we may ask whether or not we can pass symbols through walls only in one direction. This corresponds to using asymmetric graphs: for each vertices i, j, at most one of $(i, j), (j, i)$ is an edge.

In the case of a cycle-ring, the communication must go in one direction only, that is, each wall is supposed to be one-way. Thus, the only possibility to send a symbol a from a region to another one can be simply indicated by a_{go}; as usual, a_{out} means sending a outside the system (this is necessary in order to collect the result of a computation).

We can define the actions δ, τ also for asymmetric graphs, in the following way. In general, dissolving a membrane means to move all its objects to the upper membrane and remove its rules; the membrane itself disappears. In the case of planar maps it is not clear how a room can dissappear (how the neighboring rooms must connect each other after removing a room). Thus, we keep here only the first action mentioned above: when δ is introduced by a rule in a given room and we do not introduce at the same time in that room also the symbol τ, then all the objects in that room will be communicated, randomly, to any of the neighboring rooms to which a communication can take place (the action is equivalent to the fact that all symbols a in that room are transformed into a_{go} and they move through the walls which allow communication). If in some room we introduce both δ and τ at the same step, then no symbol leaves the room because of δ, but, if there are rules which introduce symbols a_{out} of a_{go}, then these symbols leave the room. If τ is introduced in a room without introducing at the same time also δ, then nothing happens. Thus, actions δ, τ do not act on the walls of the rooms, but on the objects in that room.

Of course, the case when arbitrarily many flip-flop catalysts are used is trivial: in view of Theorem 2, one room is enough. When we bound the number of catalysts, we can obtain one more representation of pRE similar to those in Theorems 3 and 4:

Theorem 5. *Each language in pRE can be generated by a P system using two bi-stable catalysts, actions δ, τ, and an underlying graph in the form of a cycle-ring.*

Proof. (Sketch) We proceed as in the proof of the Theorem 3. Starting from a matrix grammar with appearance checking in the binary normal form, with n matrices, $m_1, \ldots, m_n$, we construct a map like that in Figure 5, with a room associated with each matrix (labeled with the numbers $1, 2, \ldots, n$) and a further room, labeled by 0, placed between rooms n and 1.

Each room $i = 1, 2, \ldots, n$ has associated a symbol f_i, placed initially there, together with the two bi-stable catalysts c_1, c_2. In room 0 we initially place the objects XA for $(S \to XA)$ the S-matrix of G.

From room 0 we send all nonterminal symbols α of G to room 1, in the form (α, i), where $1 \leq i \leq n$ (to his aim, we use rules $\alpha \to (\alpha, i)_{go}$). In each room i we consider rules which send the symbols (α, j) with $j > i$ to the next room, that is, rules of the form $(\alpha, j) \to (\alpha, j)_{go}$. For (α, i), we introduce the rules $(\alpha, i) \to \bar{\alpha}$. As in the proof of Theorem 3 we now check whether or not all nonterminals are in the same room (that is, whether or not in room 0 we have associated the same address k to all symbols (α, k)). In the opposite case, the symbol δ is introduced and the symbol f_i is sent to the next room.

In each room, we provide rules $f_j \to \dagger$ for all j smaller than the label of the room.

The simulation of matrices of G are performed in the same way as in the proof of Theorem 3 (using the two bi-stable catalysts and the auxiliary symbols $d_1, d_2, d'_1, d'_2, d''_1, d''_2, e$).

For all symbols α^{iv} produced in each room when simulating a matrix, we introduce the symbols α^{iv}_{go}; moreover, each room – excepting room number n – contains rules $\alpha^{iv} \to \alpha^{iv}_{go}$. In room n, these rules are replaced by $\alpha^{iv} \to \alpha_{go}$. In this way, all symbols return (at the same time) to room 0. The process can be iterated.

As usual, the terminal symbols are collected in room 0, where they can wait any number of steps (by using rules $a \to a$) or can be sent out of the system. After using a terminal matrix, hence after removing the unique occurrence of a symbol from N_1, if any nonterminal is still present, then the computation never stops: no further matrix can be simulated, the nonterminals have to travel through the rooms of the map until reaching a room where they entail the use of action δ and the trap-symbol $\dagger$ will be produced.

Thus, our system can generate all permutations of all strings in the language generated by the grammar we start with. The formal details are left to the reader. □

It is of interest to note that without using the actions δ, τ and using only two bi-stable catalysts we still can generate a large family of languages:

Theorem 6. *Each language in the family pMAT can be generated by a P system using two bi-stable catalysts, no action δ, τ, and having a cycle-ring as the underlying graph.*

Proof. (Sketch) We repeat, with certain modifications, the construction in the proof of Theorem 5, starting from a matrix grammar G without appearance checking in the binary normal form. With each matrix m_i of the starting grammar G we associate a room (labeled with i) in a support like that in Figure 5 (hence a node in the cycle); one more room is considered, where initially we place the symbols XA. From this room we send all nonterminals to room 1 in their form from G, that is, by rules of the form $\alpha \to \alpha_{go}$. In all rooms we proceed in the same way. All nonterminals α are replaced by α', then by α'' and α''' and then sent to the next room, by commands α_{go} (this takes four time units). In any room, we can also simulate the associated matrix. This is done exactly as in the previous proof (using the two flip-flop catalysts c_1, c_2, the auxiliary symbols d_1, d_2, e, etc). This also takes four time units, that is, the symbols are travelling always together. (Thus, there is no need to control the fact that they are together, by means of actions δ, τ, that is why these actions are no longer necessary. Actually, it is not necessary that the symbols are together, because we do not have to simulate rules applied in the appearance checking manner.) Consequently, the symbols move circularly over the cycle and they are changed by simulating matrices of G. When a terminal matrix is simulated, the symbol $X \in N_1$ is removed. If any nonterminal $A \in N_2$ remains, it will either circulate for ever, or a rule $A \to x$ from a matrix $(X \to \alpha, A \to x)$ will be applied to it and the trap-symbol $\dagger$ is introduced. In both cases, the computation will never stop. The terminal symbols are kept in room 0 any number of steps (by rules of the form $a \to a$) and sent out at any time. Thus, any permutation of a string in $L(G)$ can be produced.

The formal details of this proof are left to the reader. □

We do not know whether or not this result can be improved or whether or not P systems as above can generate languages not in $pMAT$. (The only way we see for proving

such a result is to find a P system which generates a one-letter non-regular language and to make use of the fact that each one-letter matrix language is regular, [3], but we do not have found such a system.)

6 Final Remarks

We have given here two normal forms for P systems of the type considered in [7] (no priority, catalysts in a bounded number, actions δ, τ, membranes of a variable thickness, with communication based on polarity), but with the catalysts as in [12], changing their states among two possibilities, in a flip-flop manner, and with a weaker way of indicating targets of symbols to be communicated (only indications *in* and *out*). Linear graphs and star graphs are shown to suffice as underlying structures. When the number of catalysts is not bounded, systems of degree one suffice. Then, we have proposed a generalization of P systems, to graphs which are not necessarily trees. Only asymmetric graphs were considered. A representation of permutation closures of recursively enumerable languages was found also for the case of P systems with the underlying structure being a ring.

Thus, we may say that we have obtained the same powerful result – representations of pRE – for the three simplest classes of graphs: linear trees, star trees, rings.

In the theory of P systems, we already have several features, each of which already considered in several variants: we can use or not priorities; we can use or not catalysts, while the catalysts can be either stable (they never change their state) or bi-stable (they can switch between two states); we can use or not action δ; we can use or not action τ; the communication can be done by indicating the label of the target membrane (in the form in_i), using polarity (signs $+, -$ associated with symbols and with membranes), or imposing no restriction on the target (that is, using commands of the form a_{in}, and a will go to any accessible lower region); finally, we can have synchronized systems or unsynchronized systems. In total, by combining these variants, we get $2 \times 3 \times 2 \times 2 \times 3 \times 2 = 144$ types of systems. Several of them were shown to be computationally complete, able to compute the permutation closure of RE (when the result is read in an inner membrane, as the number of objects present there in a halting configuration, we can compute in this way recursively enumerable sets of natural numbers or even recursively enumerable relations of natural numbers, see [5]). A more systematic study of the 144 classes of P systems mentioned above deserves to be carried out. (The main missing tools are in this moment counterexamples and necessary conditions useful for finding counterexamples.)

References

[1] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[2] J. Dassow, Gh. Păun, On the power of membrane computing, *J. Univ. Computer Sci.*, 5, 2 (1999), 33–49 (www.iicm.edu/jucs).

[3] D. Hauschild, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, *Acta Informatica*, 31 (1994), 719–728.

[4] 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).

[5] Gh. Păun, Computing with membranes, *J. Computer and System Sciences*, in press (see also *TUCS Research Report* No. 208, November 1998, Turku Centre for Computer Science, www.tucs.fi).

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

[7] Gh. Păun, Computing with membranes. A variant: P systems with polarized membranes, *Intern. J. Foundations of Computer Science*, 11, 1 (2000) (see also *CDMTCS Report* No. 098, March 1999, Computer Science Department, Auckland Univ., www.cs.auckland.ac.nz/CDMTCS).

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

[9] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, to appear (see also *TUCS Research Report* No. 218, December 1998, Turku Centre for Computer Science, www.tucs.fi).

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

[11] 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.

[12] Gh. Păun, S. Yu, On synchronization in P systems, *Fundamenta Informaticae*, 38, 4 (1999), 397–410.

[13] I. Petre, A normal form for P systems, *Bulletin of the EATCS*, 67 (1999), 165–172.

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