Rewriting P Systems with Conditional Communication

Paolo Bottoni[1], Anna Labella[1], Carlos Martín-Vide[2], and Gheorghe Păun[3⋆]

[1] Department of Computer Science, University of Rome "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
bottoni/labella@dsi.uniroma1.it
[2] Research Group in Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
cmv@astor.urv.es
[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
gpaun@imar.ro, g_paun@hotmail.com

Abstract. A membrane system (P system) is a model of computation inspired by some basic features of the structure and behaviour of living cells. In this paper we consider systems with string-objects processed by rewriting, with the communication controlled by conditions on the contents of the strings. Symbols, substrings (in an arbitrary place, or as a prefix/suffix), or the shape of the whole string are used as permitting and as forbidding conditions when moving strings from a membrane to a neighboring membrane. Many of the obtained variants lead to new characterizations of recursively enumerable languages (as expected, these characterizations indicate a trade-off between the number of membranes and the strength of the communication conditions used). Several open problems are also formulated.

1 Introduction

Membrane computing is a branch of molecular computing which abstracts from the way the alive cells process chemical compounds (as well as energy and information) in the complex compartmental structure defined by the various membranes present inside a cell. In short, we have a *membrane structure*, in the form of a hierarchical arrangement of membranes (in principle, understood as three dimensional vesicles, but a two dimensional representation is equivalent), embedded in a *skin* membrane. Each membrane delimits in a one-to-one manner a *region*. For an elementary membrane (that is, a membrane without any membrane inside) this is the space enclosed by it, while the region of a non-elementary membrane is the space in-between the membrane and the membranes directly

⋆ Work supported by a grant of NATO Science Committee, Spain, 2000–2001, and by the Visiting professor programme of the University "La Sapienza" of Rome, Italy

W. Brauer et al. (Eds.): Formal and Natural Computing, LNCS 2300, pp. 325–353, 2002.

included in it. Each membrane is labeled; the one-to-one correspondence between membranes and regions associated with them makes it possible to refer the regions by the labels of membranes. Figure 1 illustrates these notions.

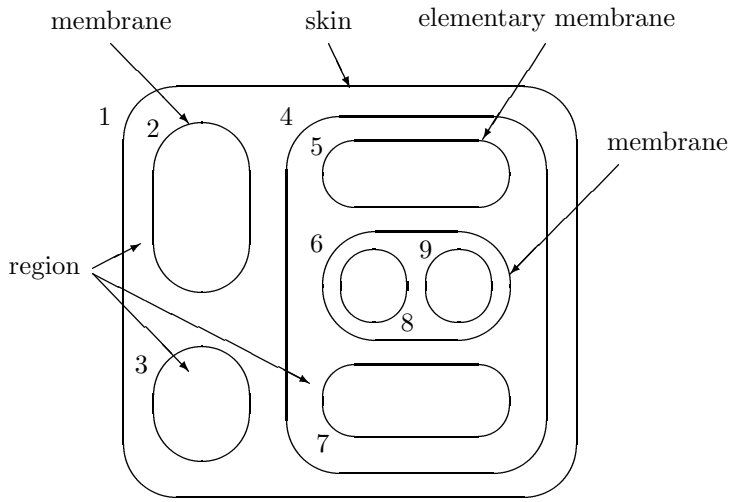

Fig. 1. A membrane structure

Each region contains *objects* and (*evolution*) *rules*. In this paper we consider the case when the objects are described by strings and the evolution rules are context-free rewriting rules. After rewriting a string, it is assumed that it is no longer present, and the result of the rewriting replaces it. (However, this is not essential for the variants of P systems we consider in this paper, but it is important for certain variants where the strings can influence each other.) In the basic variant of such a membrane system, a target indication is associated with each rule, specifying where the string obtained by a rewriting operation performed by that rule will be placed. Such indications are of the form *here* (the string remains in the same region), *out* (the string exits the region), in_j (the string is sent to the membrane with label j, providing that it is immediately inside the region where the rule is used), or, a weaker variant of the last command, *in* (the string is sent to one of the immediately inner membranes, nondeterministically choosing it).

In each step, each string which can be rewritten by a rule in its region is rewritten. (All the strings are processed in parallel, but the rewriting of each string is done sequentially: at each step only one rule per string is applied.) In this way, one gets *transitions* between the configurations of the system. A sequence of transitions is called a *computation*. The *result* of a halting computation (one which reaches a configuration where no rule can be applied) is the set of strings

sent out of the system during the computation. Thus, a rewriting P system generates a language.

Priorities among rules or possibilities to control the membrane permeability were also considered in the literature. In most cases, characterizations of recursively enumerable languages were obtained. Results of this type can be found, e.g., in [4], [17], [8], [11], [12], [13], [14], [20].

In this paper we consider 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. By combining these variants we get a large number of classes of P systems, hence of families of languages generated by these systems. As expected, many of these families equal the family of recursively enumerable languages. In many cases, such results are obtained for systems with a small number of membranes. Several cases remain to be further investigated, for instance, when checking prefixes or suffixes: we do not know whether or not we can characterize the recursively enumerable languages by systems where only prefixes or only suffixes are checked (we *conjecture* that the answer is regative); when both prefixes and suffixes are used, we have obtained a characterization of recursively enumerable languages by systems without a bound on the number of membranes (but we *conjecture* that such a characterization holds also for a reduced number of membranes, as happens for almost all classes of P systems).

We emphasize the fact that this way of communicating strings through membranes is "more realistic" than that based on target indications associated with the rewriting rules, it corresponds to the biochemical situation where the objects (some of them strings, such as DNA molecules) can pass through membranes depending on their shape (size, polarization, other properties) and not depending on the last transformation applied to them (the last rewriting, in the basic model of P systems).

2 Language Theory Prerequisites

In this section we introduce some formal language theory notions and notations which will be used in this paper; for further details we refer to [19].

For an alphabet V, by V^* we denote the set of all strings over V, including the empty one, denoted by λ; V^+ denotes the set of all non-empty strings over V, that is, $V^+ = V^* - \{\lambda\}$. 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. (For instance, $\{a, b\}^*cca^*$ is an elementary expression, while $a^* \cup ab$ is not.) We denote by $L(e)$ the language represented by a regular expression e. One can see that the language of an elementary regular expression can also be represented by a *pattern*, in the sense of [1], with each variable appearing only once (such a pattern is called *regular*)

and with a *domain* associated with each variable, in the form of a star language U^*, where U is a set of symbols. For instance, $L(e)$ for the previous expression also corresponds to the pattern $XccY$, with X interpreted by any string from $\{a, b\}^*$ and Y interpreted by any string from a^*.

By *FIN, REG, LIN, CF, CS, RE* we denote the families of finite, regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively.

In the proofs from the subsequent sections we need the notions of a *matrix grammar with appearance checking*, of *Kuroda normal form*, and of a *pure context-free 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 (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 – therefore 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. All one-letter languages in the family MAT are regular, see [10].

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 in 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 called a trap-symbol, because 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 [16] (six nonterminals, all of them used in the appearance checking mode, suffice in order to characterize RE with matrix grammars), in [9] 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 here is another result from [9]: 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.

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 *pure* context-free grammar is a construct $G = (V, S, P)$, where V is an alphabet, $S \in V$ and P is a finite set of context-free rules over V. The generated languages is defined by $L(G) = \{w \in V^* \mid S \Longrightarrow^* w \text{ with respect to } P\}$ (all generated strings are accepted).

Convention. When comparing two languages, the empty string is ignored, that is, L_1 is considered identical with L_2 as soon as $L_1 - \{\lambda\} = L_2 - \{\lambda\}$.

3 Rewriting P Systems

We introduce here only the class of P systems with string-objects processed by rewriting, in the variant we will investigate in this paper. For other classes the reader is referred to the bibliography (an up-to-date bibliography of the area can be found at the web address `http://bioinformatics.bio.disco.unimib.it/psystems`).

A membrane structure is pictorially represented by a Euler-Venn diagram (like the one in Figure 1); mathematically, it can be represented in a natural way by a tree or by a string of matching parentheses. For instance, the structure from Figure 1 is represented by the expression

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

Of course, the same membrane structure may be represented by different parenthesis expressions (neighboring membranes in the same level can be permuted).

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

$$\Pi = (V, T, \mu, M_1, \ldots, M_m, R_1, P_1, F_1, \ldots, R_m, P_m, F_m),$$

where:

1. V is the alphabet of the system;
2. $T \subseteq V$ is the *terminal alphabet*;
3. μ is a membrane structure with m membranes (injectively labeled by $1, 2, \ldots, m$);
4. $M_1, \ldots, M_m$ are finite languages over V, representing the strings initially present in the regions $1, 2, \ldots, m$ of the system;
5. $R_1, \ldots, R_m$ 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 either exit the current membrane or enter any of the directly inner membrane freely (but they cannot remain in the current membrane); we denote an empty permitting condition by $(true, X), X \in \{in, out\}$, and an empty forbidding condition by $(false, notX), X \in \{in, out\}$.
2. *symbols checking*: each P_i is a set of pairs $(a, X), X \in \{in, out\}$, for $a \in V$, and each F_i is a set of pairs $(b, notX), X \in \{in, out\}$, 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 $(u, X), X \in \{in, out\}$, for $u \in V^+$, and each F_i is a set of pairs $(v, notX), X \in \{in, out\}$, 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 $(e, X), X \in \{in, out\}$, where e is an elementary regular expression over V, and each F_i is a set of pairs $(f, notX), X \in \{in, out\}$, 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; when no upper bound on this length is imposed we replace the subscript by $*$.

A system is said to be *non-extended* if $V = T$.

The transitions in a system as above are defined in the following way. In each region, each string which can be rewritten is rewritten by a rule from

that region. The rule to be applied and the nonterminal it rewrites are nondeterministically chosen. 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 – and nondeterministically choosing the inner membrane in the case when several directly inner membranes exist. 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 do we try to communicate the result of the rewriting or the string itself if no rewriting is possible on it.

As usual, a sequence of transitions forms a computation and the result of a halting computation is the set of strings over T sent out of the system during the computation. In the case of non-extended systems, all strings sent out are accepted. A computation which never halts yields no result. A string which remains inside the system or, in the case of extended systems, which exits but contains symbols not in T does not contribute to the generated language. The language generated in this way by a system Π is denoted by $L(\Pi)$.

The family of all languages $L(\Pi)$, computed as above by extended systems Π of degree at most $m \geq 1$ and with permitting conditions of type α and forbidding conditions of type β, is denoted by $ERP_m(\alpha, \beta)$, $\alpha, \beta \in \{empty, symb, sub_*, pref_*, suff_*, patt\} \cup \{sub_k, pref_k, suff_k \mid k \geq 1\}$; when using non-extended systems we get the family $RP_m(\alpha, \beta)$. When we use both prefix and suffix checking (each condition string may 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 will indicate this by $prefsuff_k$. If the degree of the systems is not bounded, then the subscript m is replaced by $*$.

4 Proof Mechanisms

The proofs of universality which follow in the next Section are based on the simulation of well known models of universal computating devices in normal forms.

In particular, we exploit the Kuroda normal form for type 0 grammars and the binary normal form for matrix grammars with appearance checking.

Rewriting steps in these normal forms are simulated by coordinating the sequential use of context-free rules. Such a coordination is achieved by introducing new non terminals derived from those of a grammar in normal form and tagged so

that they can be rewritten only in coordination with others analogously tagged, and constraining their coordinate usage to occur only within some well-defined membranes. We anticipate here the general mechanisms that will be exploited in the next Section in which detailed proofs are presented.

In general, as will be shown in the next Section, the distinction between terminal and non terminal alphabets can be simulated by the use of an additional membrane letting only terminal symbols out. The proofs need therefore to consider only pure systems or extended systems with an empty forbidding condition. It is indeed easy to realise that forbidding has global scope, while permitting conditions can obtain a global effect only by constraining the complete form of a word, i.e. by expressing the constraint through patterns.

In particular, we will observe that patterns can constrain the simulation of Kuroda rewriting by allowing only the communication of words in which symbols derived from a rule of the form $AB \to CD$ are present in the correct sequence, i.e. $(A, r)[B, r]$, where r is a label indicating the originating rule. In this case communication is allowed towards a membrane where the original rule can be simulated in two steps.

Conversely, a forbidding condition needs only to check that symbols derived from the same rule appear in the correct order, by listing all possible incorrect occurrences of these symbols in the context of symbols related to other rules. Hence checking a substring of length two suffices.

This is not sufficient if the substring is checked in a permitting condition. A check on individual symbols is indeed needed in a forbidding condition to force strings containing derived non terminals to reach a membrane where they can be both used. In this case, the depth of the membrane system must increase for the rewriting to occur only in the innermost membrane. In general, the level of nesting is related to the decomposition of the Kuroda rules, which can occur through at most 4 levels. One can also observe that checking on single symbols is already powerful in that it forces directionality in the traversing of the membranes by preventing movements in the opposite direction. Finally, checking based on suffix/prefix strings can be used to constrain the location in which rules have to be applied.

For proofs based on the simulation of matrix grammars, special membranes are needed in order to simulate the use of rules on which appearance checking is based. Rewriting in these matrices means that the original rewriting in the matrix grammar would produce the trap symbol #. For the other rules at least two levels of nesting are needed to simulate the coupling of the rules in the binary normal form. In this case the check can be based on just one symbol, as needed for directing words with symbols deriving from a matrix to reach the correct membrane into which to simulate the matrix.

Hence a trade-off occurs between the complexity of the check and the depth of the membrane system, which remains in any case limited. The degree of branching of the membrane system is in general independent of the grammar. Hence, the results obtained hold for P systems with a fixed number of membranes. The only exception is the case of check based on prefixes and suffixes, where branch-

ing depends on the number of rules of the form $AB \to CD$ present in the original grammar (each simulated with 4 levels of nesting in independent subsystems) and on the number of symbols in the grammar (to consider that a prefix can start with any of the grammar symbols).

5 Generative Power

We start by mentioning some relations which directly follow from the definitions or can be easily proved; in all these relations α, β assume all possible values in the set $\{empty,\ symb,\ sub_*,\ pref_*,\ suff_*,\ prefsuff_*,\ patt\} \cup \{sub_k, pref_k,\ suff_k,\ prefsuff_k \mid k \geq 1\}$.

Lemma 1. (i) $ERP_*(\alpha, \beta) \subseteq RE$.
(ii) $RP_m(\alpha, \beta) \subseteq ERP_m(\alpha, \beta), m \geq 1$.
(iii) $RP_m(\alpha, \beta) \subseteq RP_{m+1}(\alpha, \beta), m \geq 1$.
(iv) $RP_m(empty, \beta) \subseteq RP_m(symb, \beta) = RP_m(sub_1, \beta) \subseteq RP_m(sub_*, \beta)$
$\subseteq RP_m(patt, \beta)$, $m \geq 1$.
(v) $RP_m(\alpha_k, \beta) = RP_m(\alpha_{k+1}, \beta), m, k \geq 1, \alpha \in \{sub,\ pref,\ suff,\ prefsuff\}$.
The relations from (iii) – (v) *are valid also in the case of families of language generated by extended systems; in all relations* (ii), (iv), (v) *the subscript m can also be* $*$.

Lemma 2. $ERP_m(\alpha, \beta) \subseteq RP_{m+1}(\alpha, \beta)$, *for all* α, β *such that* $\beta \neq empty$ *and* $m \geq 1$.

Proof. Starting from a given system Π with the total alphabet V and the terminal alphabet T, with a selection of a type (α, β) such that β allows at least the checking of symbols, we add one further membrane around the skin membrane of Π, as the skin membrane of the new system, we introduce no rewriting rule into it, but only the permitting condition $(true, out)$ and the forbidding conditions $(a, notout)$, for all $a \in V - T$. If we denote by Π' the obtained system, then we clearly have $L(\Pi) = L(\Pi')$ (no rewriting is possible in the skin membrane of Π', hence the halting computations in Π are halting also in Π', but only strings over T can leave the system). □

Corollary 1. $RP_*(\alpha, \beta) = ERP_*(\alpha, \beta)$ *for all* α, β *such that* $\beta \neq empty$.

We now pass to proving the universality results we have announced. They suggest a trade-off between the number of membranes and the permitting/forbidding conditions we use. We start by using strong conditions, and this makes possible characterizations of RE by systems with a reduced number of membranes.

Theorem 1. $RP_2(patt, empty) = RE$.

Proof. Let $G = (N, T, S, P)$ be a type-0 Chomsky grammar in the Kuroda normal form; assume that all non-context-free rules in P are labeled in a one-to-one manner. We construct the P system

$$\Pi = (V, V, [_1[_2\ ]_2]_1, \{S\}, \emptyset, R_1, P_1, F_1, R_2, P_2, F_2),$$

with the following components:

$$\begin{aligned}
V &= T \cup N \cup \{(A,r),[B,r] \mid r : AB \to CD \in P\},\\
R_1 &= \{A \to x \mid A \to x \in P\}\\
&\quad \cup \{A \to (A,r), B \to [B,r] \mid r : AB \to CD \in P\},\\
P_1 &= \{(T^*, out)\}\\
&\quad \cup \{(N \cup T)^*(A,r)[B,r](N \cup T)^*, in) \mid r : AB \to CD \in P\},\\
F_1 &= \{(false, notin), (false, notout)\},\\
R_2 &= \{(A,r) \to C,\ [B,r] \to D \mid r : AB \to CD \in P\},\\
P_2 &= \{((N \cup T)^*, out)\},\\
F_2 &= \{(false, notout)\}.
\end{aligned}$$

At any moment, only one string is present in the system; initially, this is the axiom of G.

Only terminal strings can be sent out. A string which contains at least a nonterminal of G is either rewritten in the skin membrane, or it remains forever there, hence we get no output. A string from the skin membrane can be sent to the inner membrane only if it is of the form $x(A,r)[B,r]y$, for some rule $r : AB \to CD$ and $x, y \in (N \cup T)^*$, which ensures the correct simulation in membrane 2 of this rule; the string exits membrane 2 only after replacing (A,r) with C and $[B,r]$ with D. The context-free rules of P are simulated in the skin membrane. Consequently, $L(G) = L(\Pi)$. □

When checking only substrings we need non-empty forbidding conditions, or a bigger number of membranes (Theorem 3); in turn, checking substrings of length at most two suffices.

Theorem 2. $RP_2(empty, sub_2) = RE$.

Proof. For a type-0 Chomsky grammar in the Kuroda normal form $G = (N, T, S, P)$, with the non-context-free rules in P labeled in a one-to-one manner, we construct the P system

$$\Pi = (V, T, [_1[_2\ ]_2]_1, \{XSX\}, \emptyset, R_1, P_1, F_1, R_2, P_2, F_2),$$

with the following components:

$$\begin{aligned}
V &= T \cup N \cup \{A' \mid A \in N\} \cup \{X\}\\
&\quad \cup \{(A,r),[B,r] \mid r : AB \to CD \in P\},\\
R_1 &= \{A \to x \mid A \to x \in P\}\\
&\quad \cup \{A \to (A,r), B \to [B,r] \mid r : AB \to CD \in P\}\\
&\quad \cup \{A \to A,\ A' \to A \mid A \in N\}\\
&\quad \cup \{X \to \lambda\},\\
P_1 &= \{(true, out), (true, in)\},
\end{aligned}$$

$$
\begin{aligned}
F_1 &= \{(X, notout)\} \cup \{(A, notout), (A', notout) \mid A \in N\} \\
&\quad \cup \{((A, r), notout), ([B, r], notout) \mid \text{ for all } A, B \in N \text{ and} \\
&\qquad r \text{ a non-context-free rule in } P\} \\
&\quad \cup \{(A, r)\alpha, notin), ((A, r)(E, r'), notin), (\alpha[B, r], notin), \\
&\qquad ([B, r][E, r'], notin), ([B, r](E, r'), notin) \mid \\
&\qquad \text{for all possible } A, B, E \in N, \alpha \in N \cup T \cup \{X\}, \\
&\qquad \text{and } r \text{ a non-context-free rule in } P\}, r \neq r' \\
R_2 &= \{(A, r) \to C',\ [B, r] \to D' \mid r : AB \to CD \in P\}, \\
P_2 &= \{(true, out)\}, \\
F_2 &= \{((A, r)E', notout), (E'[B, r], notout) \mid A, B, E \in N\}.
\end{aligned}
$$

The equality $L(G) = L(\Pi)$ is easy to be proved: as in the proof of Theorem 1, we can pass a string from membrane 1 to membrane 2 only if it is of the form $x(A, r)[B, r]y$, for $r : AB \to CD \in P$ and $x, y \in (N \cup T)^*$, maybe with x starting and y ending with X (the forbidding conditions prevent having further occurrences of symbols of the form $(E, p), [E, p]$ in the strings x, y; a string with only one occurrence of a symbol of the form $(E, p), [E, p]$ cannot be sent to the inner membrane); a string in $(N \cup T)^*$, maybe starting or ending with X, can be sent to membrane 2 after applying a rule $A \to A, A' \to A$, or $X \to \lambda$ in membrane 1, but it will exit immediately, unchanged; note also that only terminal strings with respect to G can be sent out of the system. □

At the price of using a larger number of membranes, we can obtain a characterization of RE by systems which check permitting substrings of length 2 and forbidding symbols.

Theorem 3. $RP_4(sub_2, symb) = RE$.

Proof. We start again from a type-0 Chomsky grammar in the Kuroda normal form $G = (N, T, S, P)$, with the non-context-free rules in P labeled in a one-to-one manner, and we construct the P system

$$\Pi = (V, T, [_1[_2[_3[_4\ ]_4]_3]_2]_1, \{S\}, \emptyset, \emptyset, \emptyset, R_1, P_1, F_1, \ldots, R_4, P_4, F_4),$$

with the following components:

$$
\begin{aligned}
V &= T \cup N \cup \{A', A'' \mid A \in N\} \cup \{\$, Z\} \\
&\quad \cup \{(A, r), [B, r] \mid r : AB \to CD \in P\}, \\
R_1 &= \{A \to x \mid A \to x \in P\} \\
&\quad \cup \{A \to (A, r) \mid r : AB \to CD \in P\} \\
&\quad \cup \{E \to E,\ E'' \to E \mid E \in N\}, \\
P_1 &= \{(\lambda, out)\} \cup \{((A, r), in) \mid r : AB \to CD \in P\}, \\
F_1 &= \{(E, notout), (E', notout) \mid E \in N\} \\
&\quad \cup \{((A, r), notout) \mid r : AB \to CD \in P\},
\end{aligned}
$$

$$
\begin{aligned}
R_2 &= \{B \to [B,r] \mid r : AB \to CD \in P\} \\
&\quad \cup \{E' \to E'',\ E'' \to Z \mid E \in N\}, \\
P_2 &= \{(E'', out) \mid E \in N\} \\
&\quad \cup \{([B,r], in) \mid r : AB \to CD \in P\}, \\
F_2 &= \{([B,r], notout) \mid r : AB \to CD \in P\} \\
&\quad \cup \{(E', notin), (E'', notin) \mid E \in N\}, \\
R_3 &= \{E' \to E \mid E \in N\}, \\
P_3 &= \{((A,r)[B,r], in) \mid r : AB \to CD \in P\} \\
&\quad \cup \{(E', out) \mid E \in N\}, \\
F_3 &= \{(\$, notin), (\$, notout)\}, \\
R_4 &= \{(A,r) \to C',\ [B,r] \to D' \mid r : AB \to CD \in P\}, \\
P_4 &= \{(C'D', out) \mid r : AB \to CD \in P\}, \\
F_4 &= \{(\$, notout)\}.
\end{aligned}
$$

The symbol \$ never appears in a string, it is only used in some forbidding conditions which should always be false, while Z is a trap-symbol, once introduced it is never removed. A string can be sent from membrane 1 to membrane 2 only after using a rule of the form $A \to (A,r)$ associated with a rule $r : AB \to CD$ from P, and, conversely after using such a rule, the string *must* go to membrane 2 (this ensures the fact that exactly one such a rule is used). Similarly, a string can be sent from membrane 2 to membrane 3 after using one rule of the form $E \to [E,p]$ (if no rule of this form can be used, then the string remains forever in membrane 2 and we get no output). Membrane 3 just checks whether or not the two rules used in membranes 1 and 2 correspond to the same non-context-free rule of P and the symbols which were rewritten are adjacent. If this is the case, then the string enters membrane 4, where the rule $r : AB \to CD$ is simulated; only when both C' and D' are introduced can the string leave membrane 4. One primed symbol is replaced in membrane 3 with its non-primed variant and the string immediately exits. In membrane 2 we cannot use a rule of the form $E \to [E,p]$, because the string will remain forever here (it cannot exit because of $[E,p]$ and cannot go to membrane 3 because of the primed symbol; even after using a rule $E' \to E''$, we cannot send the string to membranes 1 or 3). Thus, we have to use the rule $E' \to E''$ and exit. Assume that in the skin membrane we introduce again a symbol (A,r) before using the rule $E'' \to E$. The string goes to membrane 2. If we use the rule $E'' \to Z$, then the trap-symbol is introduced. If we use a rule of the form $B \to [B,r]$, then the string will remain here forever (hence, eventually, the rule $E'' \to Z$ should be used). Therefore, in the skin membrane we have to remove the primes, and only after that can we start simulating another non-context-free rule of G. The context-free rules are simulated in the skin membrane at any time before using a rule $A \to (A,r)$. Thus, any derivation of G can be simulated in Π. Clearly, only terminal strings with respect to G can be sent out, so we have the equality $L(G) = L(\Pi)$. □

By using one further membrane, as well as final selection by a terminal alphabet, we can completely avoid the checking of forbidding conditions:

Theorem 4. $RE = ERP_5(sub_2, empty)$.

Proof. We start again from a type-0 Chomsky grammar in the Kuroda normal form $G = (N, T, S, P)$, with the non-context-free rules in P labeled in a one-to-one manner, and we construct the P system

$$\Pi = (V, T, [_1[_2[_4\ ]_4]_2[_3[_5\ ]_5]_3]_1, \emptyset, \emptyset, \emptyset, \{S\}, \emptyset, R_1, P_1, F_1, \ldots, R_5, P_5, F_5),$$

with the following components:

$$\begin{aligned}
V &= T \cup N \cup \{A', A'', A''' \mid A \in N\} \cup \{f, Z\} \\
&\quad \cup \{(A, r), [B, r] \mid r : AB \to CD \in P\}, \\
R_1 &= \{f \to \lambda\} \\
&\quad \cup \{A' \to Z,\ A''' \to Z, A'' \to A \mid A \in N\}, \\
P_1 &= \{\lambda, out)\} \\
&\quad \cup \{((A, r)[B, r], in) \mid r : AB \to CD \in P\} \\
&\quad \cup \{(A', in) \mid A \in N\}, \\
R_2 &= \{B \to [B, r] \mid r : AB \to CD \in P\} \\
&\quad \cup \{E' \to E'', E''' \to Z \mid E \in N\}, \\
P_2 &= \{(f, out)\} \\
&\quad \cup \{([B, r], out) \mid r : AB \to CD \in P\}, \\
R_3 &= \{(A, r) \to C',\ [B, r] \to Z \mid r : AB \to CD \in P\}, \\
P_3 &= \{(E', in), (E'', out) \mid E \in N\}, \\
R_4 &= \{A \to (A, r) \mid r : AB \to CD \in P\} \\
&\quad \cup \{A \to x,\ A \to xf \mid A \to x \in P\} \\
&\quad \cup \{A''' \to A \mid A \in N\}, \\
P_4 &= \{(f, out)\} \cup \{((A, r), out) \mid r : AB \to CD \in P\}, \\
R_5 &= \{[B, r] \to D'' \mid r : AB \to CD \in P\} \\
&\quad \cup \{C' \to Z \mid C \in N\}, \\
P_5 &= \{(C'D'', out) \mid r : AB \to CD \in P\}.
\end{aligned}$$

All sets of forbidding conditions consist of the pairs $(false, notin)$, $(false, notout)$.

This system works as follows. We start in membrane 4 with the axiom of G. The context-free rules of G can be simulated here. If a terminal rule $A \to xf$ is used in membrane 4, then the string exits; if it not terminal and a rule $B \to [B, r]$ is used in membrane 2, then the string goes to membrane 1 and from here out of the system; but since it is not terminal, it is not accepted in the generated language. If the string is terminal, then it exits (f is erased in the skin membrane) and is introduced in $L(\Pi)$.

Assume that a string w is rewritten in membrane 4 by a rule $A \to (A, r)$ associated with a rule $r : AB \to CD \in P$. It exits; if no rule can be applied in membrane 2, then no output is produced. Assume that a rule $E \to [E, p]$ is used in membrane 2. The string is immediately sent to membrane 1. If the two tuple symbols are not associated with the same rule from P, so that no rule can be used in the skin membrane, then the string is sent out of the system, but it is not a terminal one. Assume that the string is of the form $w_1(A, r)[B, r]w_2$, for some $r : AB \to CD \in P$. No rule can be applied in the skin membrane, but the string can be sent to a lower membrane. If it arrives back in membrane 2, then it will exit either unchanged – if no rule of the form $E \to [E, p]$ can be used – or after introducing one further symbol of the form $[E, p]$. The process is repeated; eventually, the string will arrive in membrane 3 (otherwise we either continue between membranes 1 and 2 or we send the string out and it is not terminal). In membrane 3, the unique copy of the symbol (A, r) is replaced by C' and the string is sent to membrane 5. Here we also replace a symbol $[E, p]$ with H''. The string exits only if the primed symbols, $C'H''$, correspond to a rule $r : AB \to CD \in P$, otherwise the rule $C' \to Z$ introduces the trap-symbol Z. Assume that we have a substring $C'D''$ associated with a rule $r : AB \to CD \in P$. The string is sent to membrane 3. If we had at least two symbols of the form $[E, p]$, then the trap-symbol is introduced, otherwise the string is sent to the skin membrane. Consequently, we have to replace exactly two symbols A, B with $(A, r), [B, r]$, in adjacent positions, hence this corresponds to simulating the rule $r : AB \to CD \in P$.

Now, in the skin membrane (if not using the rule $C' \to Z$) we use the rule $D'' \to D$ and the string is sent to one of membranes 2 and 3. From membrane 3 the string should enter unchanged membrane 4, and here the rule $C' \to Z$ is used. Thus, we have to send the string to membrane 2. We have here two cases. If we use a rule $E \to [E, p]$, then the string is sent back to the skin membrane, where the only applicable rule is $C' \to Z$ and no terminal string will be ever obtained. If in membrane 2 we use the rule $C' \to C'''$, then the string is sent to membrane 4, where again we have two cases. If we use the rule $C''' \to C$, then we have again a string over $(N \cup T)^*$, and the process can be iterated. If, before using the rule $C''' \to C$, we use a rule $A \to (A, r)$, then the string should go to membrane 2. If we use here the rule $C''' \to Z$, then the computation will produce nothing. If we use a rule $E \to [E, p]$, then the string is sent to membrane 1, where the only applicable rule is $C''' \to Z$.

Consequently, we have to completely simulate the rule $r : AB \to CD \in P$, ending by using the rule $C'' \to C$ in membrane 4. Thus, $L(G) = L(\Pi)$. □

We do not know whether or not the number of membranes in the previous system can be decreased without decreasing the generative power. Anyway, somewhat surprisingly, one additional membrane with respect to those used in the system from the proof of Theorem 4 suffices in order to characterize RE even when checking only symbols as permitting conditions.

Theorem 5. $ERP_6(symb, empty) = RP_6(symb, symb) = RE$.

Proof. Let us consider a matrix grammar with appearance checking, $G = (N, T, S, M, F)$, in the strong binary normal form, that is with $N = N_1 \cup N_2 \cup \{S, \#\}$, with rules of the four forms mentioned in Section 2, and with $ac(G) \leq 2$. Assume that we are in the worst case, with $ac(G) = 2$, and let $B^{(1)}, B^{(2)}$ be the two symbols in N_2 for which we have rules $B^{(j)} \to \#$ in matrices of M. Let us assume that we have k matrices of the form $m_i : (X \to \alpha, A \to x), X \in N_1, \alpha \in N_1 \cup \{\lambda\}, A \in N_2$, and $x \in (N_2 \cup T)^*$, $1 \leq i \leq k$ (that is, without rules to be used in the appearance checking manner). Each matrix of the form $(X \to \lambda, A \to x), X \in N_1, A \in N_2, x \in T^*$, is replaced by $(X \to f, A \to x)$, where f is a new symbol. We continue to label the obtained matrix in the same way as the original one. The matrices of the form $(X \to Y, B^{(j)} \to \#), X, Y \in N_1$ (that is, with rules used in the appearance checking manner), are labeled by m_i, with $i \in lab_j$, for $j = 1, 2$, such that lab_1, lab_2 and $lab_0 = \{1, 2, \ldots, k\}$ are mutually disjoint sets.

We construct the extended P system (of degree 6)

$$\Pi = (V, T, \mu, M_1, \ldots, M_6, R_1, P_1, F_1, \ldots, R_6, P_6, F_6),$$

with the following components:

$$\begin{aligned}
V &= T \cup N_1 \cup N_2 \cup \{(X_i, j) \mid X \in N_1, 1 \leq i \leq k, 0 \leq j \leq k\} \\
&\quad \cup \{X_i \mid X \in N_1, i \in lab_1 \cup lab_2\} \\
&\quad \cup \{A_i, (A_i, j) \mid A \in N_2, 1 \leq i \leq k, 0 \leq j \leq k\} \\
&\quad \cup \{X', X'', X''' \mid X \in N_1\} \\
&\quad \cup \{f'', Z\}, \\
\mu &= [_1[_2[_3[_4\ ]_4]_3]_2[_5\ ]_5[_6\ ]_6]_1, \\
M_1 &= \{XA\}, \text{ for } (S \to XA) \text{ being the initial matrix of } G, \\
M_i &= \emptyset, \text{ for all } i = 2, \ldots, 6,
\end{aligned}$$

and with the following triples $(R_i, P_i, F_i), 1 \leq i \leq 6$ (the membranes with labels 2, 3, 4 will be used for simulating matrices $m_i, 1 \leq i \leq k$, while membranes with labels 5, 6 will simulate the matrices with labels in lab_1, lab_2, respectively):

R_1: $X \to X_i$, for all matrices $m_i : (X \to Y, B^{(j)} \to \#)$, $i \in lab_j, j = 1, 2$,
$A \to (A_i, 0)$, for $m_i : (X \to Y, A \to x), 1 \leq i \leq k$,
$X_i \to Y'$, for all matrices $m_i : (X \to Y, D \to \alpha)$, $i \in lab_0 \cup lab_1 \cup lab_2$,
$X'' \to X'''$, for all $X \in N_1$,
$f'' \to \lambda$;
$P_1 = \{((A_i, 0), in) \mid A \in N_2, i \in lab_0\}$
$\cup \{(X_i, in), (X', in), (X''', in) \mid X \in N_1, i \in lab_1 \cup lab_2\}$
$\cup \{(a, out) \mid a \in T\}$;
$F_1 = \{(false, notin), (false, notout)\}$;

R_2: $X_i \to Z$, for all $X \in N_1, i \in lab_1 \cup lab_2$,
$X \to (X_i, 0)$, for all $X \in N_1, 1 \leq i \leq k$,

$A_i \to x$, for all $m_i : (X \to Y, A \to x)$, $1 \leq i \leq k$,
$(A_i, j) \to Z$, for all $A \in N_2, 1 \leq j < i \leq k$,
$X'' \to Z$, for all $X \in N_1 \cup \{f\}$,
$X''' \to X$, for all $X \in N_1$;
$P_2 = \{((X_i, 0), in) \mid X \in N_1, 1 \leq i \leq k\}$
$\cup \{(X, out), (X'', out) \mid X \in N_1 \cup \{f\}\}$;
$F_2 = \{(false, notin), (false, notout)\}$;

R_3: $(X_i, j) \to (X_i, j+1)$, for all $X \in N_1, 0 \leq j < i \leq k$,
$(X_i, i) \to \alpha''$, for all $m_i : (X \to \alpha, A \to x), 1 \leq i \leq k, \alpha \in N_1 \cup \{f\}$;
$P_3 = \{((X_i, j), in) \mid X \in N_1, 1 \leq j \leq i \leq k\}$
$\cup \{(X'', out) \mid X \in N_1 \cup \{f\}\}$;
$F_3 = \{(false, notin), (false, notout)\}$;

R_4: $(A_i, j) \to (A_i, j+1)$, for all $A \in N_2, 0 \leq j < i \leq k$,
$(A_i, i) \to A_i$, for all $m_i : (X \to \alpha, A \to x), 1 \leq i \leq k, \alpha \in N_1 \cup \{f\}$,
$A_i \to Z$, for all $A \in N_2, 1 \leq i \leq k$;
$P_4 = \{((A_i, j), out) \mid A \in N_2, 1 \leq j < i \leq k\}$
$\cup \{(A_i, out) \mid A \in N_2, 1 \leq i \leq k\}$;
$F_4 = \{(false, notout)\}$;

and, for $j = 1, 2$,
$R_{4+j,1}$: $X_i \to Z$, for all $X \in N_1, i \notin lab_j$,
$(A_i, 0) \to Z$, for all $A \in N_2, i \in lab_0$,
$B^{(j)} \to Z$,
$X' \to X$, for all $X \in N_1$,
$X''' \to Z$, for all $Z \in N_1$;
$P_{4+j} = \{(a, out) \mid a \in T\}$;
$F_{4+j} = \{(false, notout)\}$;

Let us examine the work of this system.

Only strings over T are accepted in the generated language; Z is a trap-symbol, once introduced it will never be removed, hence the string will never turn to be terminal. From the skin membrane in any moment we can send out a string which contains at least one terminal symbol, but if any symbol not in T is present, then the string is not accepted in the generated language.

At any moment we have exactly one string in the system; initially, this is XA, for $(S \to XA)$ the start matrix of G. Assume that we have here a string of the general form, Xw, with $X \in N_1$ and $w \in (N_2 \cup T)^*$. We have to use a rule of the form $X \to X_i$, for $i \in lab_j, 1 \leq j \leq 3$, or of the form $A \to (A_i, 0)$, for some $A \in N_2, 1 \leq i \leq k$. In the first case we start the simulation of a matrix with appearance checking, in the second case we start the simulation of a matrix $m_i, 1 \leq i \leq k$ (without appearance checking). If no rule can be used, then the string exits, but it is not terminal.

Assume that we are in the former case. The string must be communicated to a lower level membrane, that is, one of membranes 2, 5, 6. In membranes

2 and those of 5, 6 which are different from $4 + j$ the only applicable rule is $X_i \to Z$, hence the computation will not produce a terminal string. If the string $X_i w$ arrives in the right membrane $4 + j$, that is, the membrane with $i \in lab_j$, and $B^{(j)}$ is not present in it, then no rule can be applied to it, at the next step the string exits unchanged; in this way we know that w does not contain the symbol $B^{(j)}$; if $B^{(j)}$ is present, then the symbol Z is introduced. In the skin membrane we use the rule $X_i \to Y'$ which corresponds to the matrix $m_i : (X \to Y, B^{(j)} \to \#)$ from M. The string is sent again to a lower membrane; if it arrives in membrane 2, then it will remain here forever, unchanged; from any membrane 5, 6 it exits with Y' replaced with Y. This completes the simulation of the matrix m_i. The obtained string is of the initial form, hence the process can be iterated. If in the skin membrane we do not use the rule $X_i \to Y'$, but a rule of the form $A \to (A_i, 0)$, then the string can be sent to a lower membrane; both in membrane 2 and in membranes 5, 6 there is only one applicable rule, $(A_i, 0) \to Z$, hence no terminal string will be obtained.

Assume now that we have introduced a symbol $(A_i, 0)$, corresponding to a matrix $m_i : (X \to \alpha, A \to x), 1 \leq i \leq k$, with $\alpha \in N_1 \cup \{f\}$. The string should be communicated to a lower membrane. In membranes 5, 6 the symbol Z will be introduced, hence we have to send it to membrane 2. The only applicable rule is $X \to (X_j, 0)$, for some $j \in lab_0$. The string is sent to membrane 3, where $(X_j, 0)$ is replaced with $(X_j, 1)$. The obtained string is sent to membrane 4, where $(A_i, 0)$ is replaced with $(A_i, 1)$. The string returns to membrane 3. From now on, the string will go back and forth between membranes 3 and 4, and the second component of the symbols $(A_i, s), (X_j, t)$ is alternatively increased.

Now, we distinguish three cases:

Case 1: $i < j$. This means that at some step in membrane 4 we receive from membrane 3 a string of the form $(X_j, i)w_1(A_i, i - 1)w_2$. We replace $(A_i, i - 1)$ with (A_i, i) and no communication is possible (note that $((A_i, i), out)$ is not in P_4), hence one more rewriting is necessary. We replace (A_i, i) with A_i and the string is sent out. In membrane 3 we replace (X_j, i) with $(X_j, i + 1)$ (this is possible, because $i+1 \leq j$), the string is sent back to membrane 4, where the trap-symbol is introduced (the rewriting has priority over checking the communication conditions).

Case 2: $i > j$. At some moment we produce in membrane 3 a string of the form $(X_j, j)w_1(A_i, j - 1)w_2$, which is sent to membrane 4. Here we replace $(A_i, j - 1)$ with (A_i, j), the string exits, in membrane 3 we replace (X_j, j) with $\alpha'', \alpha \in N_1 \cup \{f\}$, and the string is sent out. In membrane 2 we can apply $(A_i, j) \to Z$ or $\alpha'' \to Z$, hence again the string will never lead to a terminal one.

Case 3: $i = j$. At some moment we pass from membrane 3 to membrane 4 a string $(X_i, i)w_1(A_i, i - 1)w_2$. In membrane 4 we replace $(A_i, i - 1)$ with (A_i, i) and, because we cannot exit, we replace this latter symbol with A_i. Returned in membrane 3, we replace (X_i, i) with α'' and we send the string $\alpha'' w_1 A_i w_2$ to membrane 2. We have to continue by using the rule $A_i \to x$ and the string $\alpha'' w_1 x w_2$ is sent to the skin membrane.

We have here two subcases. If we use the rule $\alpha'' \to \alpha'''$, then the string is sent to membrane 2 again (in membranes 5, 6 the trap-symbol Z will be introduced). The only applicable rule here is $\alpha''' \to \alpha$, which completes the simulation of the matrix m_i. Moreover, we have a string of the form we have started with, hence – if $\alpha \in N_1$ – the process can be iterated. If in the skin membrane we use a rule of the form $C \to (C_l, 0)$, then a string of the form $\alpha'' z_1 (C_l, 0) z_2$ is sent to one of membranes 2, 5, 6; in all of them the trap-symbol is immediately produced.

Therefore, the only correct way (that is, leading to a terminal string) to proceed is to correctly simulate matrices from M.

At the moment when a matrix $(X \to f, A \to x)$ is simulated, the computation must stop: if a string $f'' z_1 (C_l, 0) z_2$ is sent to any membrane 2, 5, 6, then the trap-symbol is produced, hence the rule $f'' \to \lambda$ must be used in membrane 1. The string should exit (no condition allows movement to a lower membrane). If it is terminal, then it belongs to the language $L(\Pi)$, if not, then it is "lost".

If at any moment we get a string of the form $Xw, w \in T^*$, then we will never get a terminal string: we can at most simulate matrices $m_i, i \in lab_1 \cup lab_2$, but we cannot remove the symbol from N_1 present in the string. In turn, if we obtain a string without any symbol from N_1, but containing symbols from N_2, then this means that it is of the form $f''w$, hence as we have seen above, the computation cannot lead to a terminal string.

Consequently, $L(\Pi) = L(G)$, and we have the equality $EPR_6(symb, empty) = RE$.

For the non-extended case we replace all sets $F_i, 2 \leq i \leq 6$, with $\{(\$, notin), (\$, notout)\}$, where \$ is a new symbol, to be added to V, and we consider

$$F_1 = \{(\$, notin)\} \cup \{(\alpha, notout) \mid \alpha \in V - T\}.$$

No string containing symbols not in T can exit the system, but all the strings are processed as described above, hence the language is not changed. This completes the proof. □

It is an *open problem* whether or not the previous result is optimal, or the number of used membranes can be decreased.

For the case of checking separately only prefixes or only suffixes of strings we have not found a universality result, while when using both prefix and suffix checking we can characterize RE by systems with a number of membranes which is not bounded. Improving this result (we believe that the hierarchy on the number of membranes collapses also in this case) and investigating the families $RP_m(\alpha, \beta)$ with at least one of α, β in $\{pref_k, suff_k \mid k \geq 1\}$ remain as tasks for the reader.

Theorem 6. $ERP_*(prefsuff_2, empty) = RE$.

Proof. Let us consider a type-0 grammar $G = (N, T, S, P)$ in Kuroda normal form, with k non-context-free rules labeled in an injective manner, $r_i : AB \to CD, 1 \leq i \leq k$. Consider a new symbol, \$, and assume that $N \cup T \cup \{\$\} =$

$\{E_1, \ldots, E_n\}$. We construct the P system Π, of degree $4k + 2n + 7$, with the following components:

$$
\begin{aligned}
V &= T \cup N \cup \{A' \mid A \in N\} \\
&\quad \cup \{(A', r), (B, r) \mid r : AB \to CD \in P\} \\
&\quad \cup \{X, X', X'', Y, Y', Y'', Z, \$\}, \\
\mu &= [_1 [_2 [_3 [_4 [_5 \,]_5]_4 [_6 [_{E_1} [_{E'_1} \,]_{E'_1}]_{E_1} \cdots [_{E_n} [_{E'_n} \,]_{E'_n}]_{E_n}]_6 \\
&\qquad [_7 [_{r_1} [_{r'_1} [_{r''_1} [_{r'''_1} \,]_{r'''_1}]_{r''_1}]_{r'_1}]_{r_1} \cdots [_{r_k} [_{r'_k} [_{r''_k} [_{r'''_k} \,]_{r'''_k}]_{r''_k}]_{r'_k}]_{r_k}]_7]_3]_2]_1,
\end{aligned}
$$

all sets M_i of axioms are empty, excepting M_5, which contains the unique string $X''\$SY$, and with the following sets of rules and associated strings to be checked as permitting conditions (all forbidding condition sets are of the form $\{(false, notin), (false, notout)\}$):

R_1: $Y'' \to \lambda$;
$P_1 = \{(a, out) \mid a \in T\}$;

R_2: $Y' \to Y''$,
$X \to \lambda$,
$\$ \to \lambda$,
$Y \to Z$;
$P_2 = \{(Y'', out)\}$;

R_3: $\alpha \to \alpha'$, for all $\alpha \in N \cup T \cup \{\$\}$,
$X' \to X''$,
$Y \to Y'$,
$B \to (B, r)$, for all $r : AB \to CD \in P$;
$P_3 = \{(\alpha' Y, in), (\alpha Y, out) \mid \alpha \in N \cup T \cup \{\$\}\}$
$\cup \{(X'', in), (\$Y', out)\} \cup \{((B, r)Y, in) \mid r : AB \to CD \in P\}$;

R_4: $= \{\alpha' \to Z \mid \alpha \in N \cup T \cup \{\$\}\}$
$\cup \{(B, r) \to Z \mid r : AB \to CD \in P\}$;
P_4: $= \{(X, out), (X'', out)\}$;

R_5: $A \to x$, if such a rule is in P,
$X'' \to X$,
$\alpha' \to Z$, for all $\alpha \in N \cup T \cup \{\$\}$,
$(B, r) \to Z$, for all $r : AB \to CD \in P$;
$P_5 = \{(X, out)\}$;

R_6 contains no rule;
$P_6 = \{(X, in), (X', out)\}$;

R_7 contains no rule;

$P_7 = \{(X, in), (X', out)\}$,

for each $i = 1, 2, \ldots, n$, we have:

R_{E_i}: $X' \to X'$,
$E_i' \to \lambda$,
$\alpha' \to Z$, for all $\alpha \in N \cup T \cup \{\$\}$ such that $\alpha \neq E_i$,
$(B, r) \to Z$, for all $r : AB \to CD \in P$;

$P_{E_i} = \{(Y, in), (X', out)\}$;

$R_{E_i'}$: $X \to X' E_i$;

$P_{E_i'} = \{(X', out)\}$;

and for each $i = 1, 2, \ldots, k$ we have:

R_{r_i}: $\alpha' \to Z$, for all $\alpha \in N \cup T \cup \{\$\}$,
$(B, r_i) \to \lambda$,
$(F, r_j) \to Z$, for all $1 \leq j \leq k, j \neq i, F \in N$,
$X' \to X'$,
$(A', r_i) \to Z$;

$P_{r_i} = \{(Y, in), (X', out)\}$;

$R_{r_i'}$: $A \to (A', r_i)$,
$X' \to X'$;

$P_{r_i'} = \{((A', r_i)Y, in), (X', out)\} \cup \{(\alpha Y, out) \mid \alpha \in N \cup T \cup \{\$\}\}$;

$R_{r_i''}$: $(A', r_i) \to \lambda$,
$X' \to X'$;

$P_{r_i''} = \{(Y, in), (X', out)\}$;

$R_{r_i'''}$: $X \to X'CD$;

$P_{r_i'''} = \{(X', out)\}$.

For the reader's convenience, Figure 2 presents the shape of the membrane structure of the system Π.

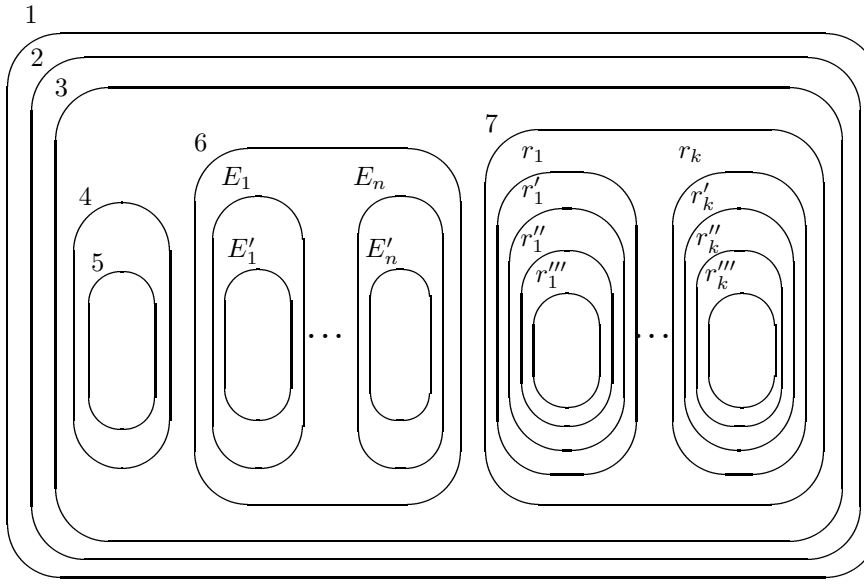

Fig. 2. The membrane structure of the system Π from the proof of Theorem 6

The idea of this construction is to simulate the non-context-free rules from P in the right end of the strings of Π, and, to this aim, the sentential forms of G are circularly permuted in Π; the symbol \$ indicates the actual beginning of strings from G: if $Xw_1\$w_2Y$ is a sentential form of Π, then w_2w_1 is a sentential form of G. Z is a trap-symbol, once introduced, it cannot be removed, hence the string will never turn to be terminal.

We start from the string $X''\$SY$, initially present in membrane 5.

In membrane 5 we can simulate any context-free rule from P and the string, if not starting with X, will remain in the same region. After using the rule $X'' \to X$, it has to exit. From membrane 4, it immediately goes to membrane 3.

If in membrane 3 we use the rule $Y \to Y'$, then the string will go to membrane 2 only if it ends with $\$Y'$, which means that it is in the same permutation as in G. This is the way to produce a terminal string: the auxiliary symbol $X, \$$ are erased, Y' is replaced by Y'', the string is sent to the skin membrane, where Y'' is erased. If the string which is sent out of the system is terminal, then it is added to the language $L(\Pi)$, if not, then it is "lost".

Assume that in membrane 3 we have a string XwY and we use a rule $\alpha \to \alpha'$, for some $\alpha \in N \cup T \cup \{\$\}$. This will start the procedure of circularly permuting the string with one symbol: α is removed from the right end of the string and added in the left end of the string. Note that the nonterminals, the terminals, and the symbol \$ are treated in the same manner. If the primed symbol is the rightmost one, then the condition to send the string to a lower membrane is fulfilled, otherwise the string is sent to membrane 2, because a condition $(\beta Y, out)$

is fulfilled, for some $\beta \in N \cup T \cup \{\$\}$. In membrane 2 we have to eventually use the rule $Y \to Z$ and no terminal string will be obtained. Thus, we have to prime the rightmost symbol of the string w, and the obtained string is sent to a lower level membrane. If it arrives in membrane 4, then the symbol Z is introduced by the rule $\alpha' \to Z$, if the string arrives in one of membranes 6 and 7, then it will be sent to a lower level membrane. In all membranes different from $[_\alpha\ ]_\alpha$ the trap-symbol is introduced. In membrane $[_\alpha\ ]_\alpha$ the symbol α' is removed and the string is sent to the inner membrane $[_{\alpha'}\ ]_{\alpha'}$, where the same symbol α is introduced by the symbol X in the leftmost position; at the same time, X is replaced by X', which makes possible sending the string up to membrane 3. We have here several possibilities.

If we use the rule $Y \to Y'$ and the string is sent out (the special symbol \$ was adjacent to Y'), then we cannot remove X', hence the string will not lead to a terminal string. If we use a rule $\beta \to \beta'$, then either the string goes to membrane 2 and Z is introduced, or it goes to a lower membrane; in membrane 4 one introduces the trap-symbol by the rule $\beta' \to Z$, from membranes 6 and 7 we have to exit immediately, hence the process is repeated. Similar results are obtained if we use a rule $B \to (B, r)$, for some $r : AB \to CD \in P$. Thus, we have to use the rule $X' \to X''$, which implies that the string is sent to a lower level membrane. This should be membrane 4, otherwise the string remains forever in one of membranes 6 and 7. From membrane 4 the string is sent to membrane 5, where we can simulate context-free rules from P and eventually we have to use again the rule $X'' \to X$ and send to membrane 3 a string XzY (with z obtained by circularly permutingg the string w with a symbol, maybe also by using some context-free rules from P). The process can be iterated.

A similar procedure ensures the simulation of rules $r : AB \to CD$, in the membranes "protected" by membrane 7: a symbol B is replaced by (B, r) in membrane 3; if this is not done in the rightmost position, then the string has to exit and in membrane 2 one introduces the trap-symbol; if the string is of the form $Xw(B, r)Y$, then it goes to one of membranes 4, 6, or 7; in membrane 4 one introducees the trap-symbol, from membrane 6 the string is sent to a lower membrane and will introduce the trap-symbol. From membrane 7, the string is sent to a lower level membrane. If this is not the one associated with the rule r, then again the trap-symbol is introduced, otherwise in membrane $[_r\ ]_r$ one removes (B, r) and one goes to the lower membrane. In membrane $[_{r'}\ ]_{r'}$ one also replaces A by (A', r). If this is not done in the rightmost position, then the string has to exit, and in membrane $[_r\ ]_r$ one introduces the trap-symbol. If A was the rightmost symbol, then the string goes one membrane below, where also (A', r) is erased. The string enters the lowest membrane associated with the rule r, where the rule $X \to X'CD$ completes the simulation of the rule. Because of X', the string can now exit all membranes associated with the rule r, and returns to membrane 3. Again the process can be iterated.

Consequently, we can correctly simulate all rules from P, and all strings which can be sent out of the system and are terminal precisely correspond to

strings generated by terminal derivation of G. That is, $\mathrm{L}(G) = L(\Pi)$, which concludes the proof. □

6 The Remaining Families

"Below" the families considered in Theorems 1 – 5 there remain several families whose size is not precisely known. We will present here some results in this respect, especially about the families of languages generated by systems with only one membrane, but a systematic study of these families remains as a topic for further research.

Theorem 7. $RP_1(empty, symb) = CF$.

Proof. For a context-free grammar $G = (N, T, S, P)$ we consider the system $\Pi = (N \cup T, N \cup T, [_1\ ]_1, \{S\}, P, \{(true, out)\}, \{(A, notout) \mid A \in N\})$, and we obviously have $L(G) = L(\Pi)$, that is, $CF \subseteq RP_1(empty, symb)$.

Conversely, consider a system $\Pi = (V, T, [_1\ ]_1, M_1, R_1, P_1, F_1)$ with $P_1 = \{(true, out)\}$ and with $F_1 = \{(b_i, notout) \mid 1 \leq i \leq k\}$, for some $k \geq 1$, $b_i \in V, 1 \leq i \leq k$. Denote by $dom(R_1)$ the set $\{a \in V \mid a \to z \in R_1\}$. We construct the pure context-free grammar $G = (U, S, P)$ with

$$U = V \cup \{a' \mid a \in V\} \cup \{S, c\},$$

where S, c are new symbols, and with the following rules ($g(w)$ denotes the string obtained by priming all symbols from $w \in V^*$):

$$\begin{aligned} P = \{&S \to g(w)c \mid w \in M_1, alph(w) \cap dom(R_1) = \emptyset, \\ &\text{and } alph(w) \cap \{b_1, \ldots, b_k\} = \emptyset, \\ &\text{or } z \Longrightarrow w \text{ by a rule from } R_1, (alph(z) \cup alph(w)) \cap \{b_1, \ldots, b_k\} = \emptyset\} \\ &\cup \{S \to w \mid w \in M_1, alph(w) \cap dom(R_1) \neq \emptyset\} \\ &\cup \{a \to z \mid a \to z \in R_1\} \\ &\cup \{b_i \to g(z)c \mid b_i \to z \in R_1, 1 \leq i \leq k\}. \end{aligned}$$

Consider also the morphism h defined by $h(a) = h(a') = a, a \in V$, and $h(c) = \lambda$. We have the equality

$$L(\Pi) = h(L(G) \cap (V \cup V')^* c (V \cup V')^*) \cap \{w \in V^* \mid alph(w) \cap \{b_1, \ldots, b_k\} = \emptyset\} \cap T^*,$$

where $V' = \{a' \mid a \in V\}$. Indeed, the intersection with the regular language $(V \cup V')^* c (V \cup V')^*$ selects from the language $L(G)$ only those strings which contain exactly one occurrence of c, that is, they are either strings from M_1 which cannot be rewritten (but can be sent out), or they are obtained by a derivation step when a rule from R_1 is used. In the latter case, we either have rewritten an axiom w_1, obtaining a string w_2 such that both w_1 and w_2 contain no symbol $b_i, 1 \leq i \leq k$ (hence the derivation can have only one step), or a symbol $b_i, 1 \leq i \leq k$, appears either in w_1 or in w_2, and then a rule $b_i \to g(z)c$ is used during the

derivation. Moreover, the intersection with $\{w \in V^* \mid alph(w) \cap \{b_1, \ldots, b_k\} = \emptyset\}$ ensures the fact that if a string of the form $x_1 g(z) c x_2$ is obtained by using such a rule, then we have $alph(x_1 z x_2) \cap \{b_1, \ldots, b_k\} = \emptyset$, hence it can be sent out. If further rules of the form $a \to y$ (without introducing the symbol c) are used for rewriting a string $x_1 g(z) c x_2$, then the string remains of the same form, and such rules were also possible to be used before using the rule $b_i \to g(z)c$ (the primed symbols prevent to use rules for rewriting the symbols from z), hence this does not lead to strings not in $L(\Pi)$. If a rule of the form $b_i \to g(y)c$ is used, then the obtained string is not in the intersection, because it contains two occurrences of c. The fact that the symbol c is introduced by a rule which removes a symbol $b_i, 1 \leq i \leq k$, ensures the fact that we finish the derivation in the moment when removing the last symbol from the forbidding set, hence the string is introduced in the language of $L(G)$ in the same moment when the corresponding string is sent out of the system Π.

The morphism h erases the primes and the symbol c, hence we return to a string from V^*. The intersection with T^* ensures the selection of terminal strings only.

All these operations preserve context-freeness, hence $L(\Pi) \in CF$. □

Theorem 8. $ERP_1(empty, empty) = RP_1(empty, empty) = FIN$.

Proof. Let $\Pi = (V, T, [\,]_1, M_1, R_1, \{(true, out)\}, \{(false, notout)\})$ be a system with empty communication conditions. After each rewriting the string must exit the system, hence all computations have at most one step, which means that $L(\Pi)$ is finite. Conversely, we can take any finite language as M_1 and no rule in R_1. In this way, we just send out the strings from M_1, hence all finite languages are in $RP_1(empty, empty)$. □

Lemma 1. $RP_1(symb, empty) - LIN \neq \emptyset$.

Proof. The system

$$\Pi = (\{a, b, c, d\}, \{a, b, c, d\}, [_1\,]_1, \{dd\},$$
$$\{d \to adb, d \to acb\}, \{(c, out)\}, \{(false, notout)\}),$$

generates the non-linear language

$$L(\Pi) = \{a^n c b^n a^m d b^m,\ a^m d b^m a^n c b^n \mid n \geq 1, m \geq 0\}.$$

(We can send out a string only if it contains the symbol c, hence immediately after using the rule $d \to acb$; the rule $d \to adb$ can be used any number $m \geq 0$ of times.) □

Lemma 2. $ERP_1(symb, empty) \subseteq CF$.

Proof. The proof is similar (but simpler) to that of the inclusion $RP_1(empty, symb) \subseteq CF$ from the proof of Theorem 7, so we only present the construction

needed (the notations for $dom(R_1), g, h, V'$ are the same as in the proof of Theorem 7). Let $\Pi = (V, T, [_1\ ]_1, M_1, R_1, P_1, F_1)$ be a system with $P_1 = \{(b_i, out) \mid 1 \leq i \leq k\}$, for some $k \geq 1$, $b_i \in V, 1 \leq i \leq k$, and with $F_1 = \{(false, notout)\}$. We construct the pure context-free grammar $G = (U, S, P)$ with

$$U = V \cup \{a' \mid a \in V\} \cup \{S, c\},$$

where S, c are new symbols, and with the following rules:

$$\begin{aligned} P = \{&S \to g(w)c \mid w \in M_1, alph(w) \cap dom(R_1) = \emptyset, \\ &\text{and } alph(w) \cap \{b_1, \ldots, b_k\} \neq \emptyset\} \\ &\cup \{S \to w \mid w \in M_1, alph(w) \cap dom(R_1) = \emptyset\} \\ &\cup \{a \to z \mid a \to z \in R_1, alph(z) \cap \{b_1, \ldots, b_k\} = \emptyset\} \\ &\cup \{a \to g(z)c \mid a \to z \in R_1, alph(z) \cap \{b_1, \ldots, b_k\} \neq \emptyset\}. \end{aligned}$$

We have the equality

$$L(\Pi) = h(L(G) \cap (V \cup V')^* c (V \cup V')^*) \cap T^*,$$

consequently $L(\Pi) \in CF$. □

This inclusion is proper. Actually, the following stronger result holds (which somehow completes the study of the families $ERP_1(symb, empty), RP_1(symb, empty)$):

Lemma 3. *All one-letter languages in $ERP_1(sub, empty)$ are finite.*

Proof. Consider a regular language $L \subseteq a^*$ and let $\Pi = (V, \{a\}, [_1\ ]_1, M_1, R_1, P_1, \{(false, notout)\})$ be a system such that $L(\Pi) = L$. At least one condition from P_1 should be of the form (a^k, out) for some $k \geq 0$. Let k_0 be the smallest k with this property. All strings obtained by using a rule from R_1 and containing a substring a^j with $j \geq k_0$ is sent out of the system. Let $k_1 = \max\{|x| \mid \alpha \to x \in R_1\}$, $k_2 = \max\{|w| \mid w \in M_1\}$, and denote $t = \max\{k_0, k_1, k_2\}$.

No string of the form a^m with $m > 3t$ can be generated by the system Π. Indeed, in order to produce such a string we need a rewriting $w \Longrightarrow a^m$. Because $m > 3t$, we must have $|w| > 2t$, hence $w \notin M_1$. This means that in its turn, also w was obtained by a rewriting. However, because $t \geq k_0$, it follows that $a^{k_0} \in Sub(w)$. This means that w should be sent out immediately after obtaining it, hence the step $w \Longrightarrow a^m$ cannot be performed. This contradiction closes the proof. □

Theorem 9. *The families $ERP_1(symb, empty), RP_1(symb, empty)$ are incomparable with REG, LIN, and strictly included in CF.*

However, *REG* is "almost included" into $RP_1(symb, empty)$:

Theorem 10. *For each regular language $L \subseteq T^*$ and $c \notin T$, the language $L\{c\}$ is in $RP_1(symb, empty)$.*

Proof. For a regular grammar $G = (N, T, S, P)$ and $c \notin T$ we consider the system $\Pi = (\{N \cup T \cup \{c\}, N \cup T \cup \{c\}, [_1\ ]_1, \{S\}, \{A \to aB \mid A \to aB \in P\} \cup \{A \to ac \mid A \to a \in P\}, \{(c, out)\}, \{(false, notout)\})$. A string can be sent out only when c is present, which means that a derivation in G was completed, hence $L(\Pi) = L(G)\{c\}$. □

If we pass to systems with (at least) two membranes, then much more complex languages can be produced, even when using communication conditions of a weak type.

Theorem 11. $RP_2(empty, empty) - CF \neq \emptyset$.

Proof. The system

$$\Pi = (\{a, b, c, d_1, d_2\}, \{a, b, c, d_1, d_2\}, [_1[_2\ ]_2]_1, \{d_1d_2\}, \emptyset, R_1, P_1, F_1, R_2, P_2, F_2),$$
$$R_1 = \{d_1 \to ad_1b\}, P_1 = \{(true, in), (true, out)\},$$
$$F_1 = \{(false, notin), (false, notout)\}, R_2 = \{d_2 \to cd_2\},$$
$$P_2 = \{(true, out)\}, F_2 = \{(false, notout)\},$$

generates the non-context-free language

$$L(\Pi) = \{a^{n+1}d_1b^{n+1}c^nd_2 \mid n \geq 0\}.$$

Indeed, after a number of steps when the current string is moved between the skin membrane and the inner membrane (in such a step all symbols a, b, c increase by one the number of occurrences), the string can be sent out, which means that one further copy of a and b are produced. □

Theorem 12. $RP_2(empty, symb) - MAT \neq \emptyset$.

Proof. The system

$$\Pi = (\{a, b\}, \{a, b\}, [_1[_2\ ]_2]_1, \{a\}, \emptyset, R_1, P_1, F_1, R_2, P_2, F_2),$$
$$R_1 = \{a \to bb\}, P_1 = \{(true, in), (true, out)\}, F_1 = \{(a, notin), (a, notout)\},$$
$$R_2 = \{b \to a\}, P_2 = \{(true, out)\}, F_2 = \{(b, notout)\},$$

generates the language

$$L(\Pi) = \{b^{2^n} \mid n \geq 0\}.$$

Assume that we have a string a^m in the skin membrane; initially, $m = 1$. We have to use the rule $a \to bb$ for all copies of a before communicating the obtained string. Thus, we have to obtain the string b^{2m}, which is either sent out of the system or to the inner membrane. In membrane 2 we have to use the rule $b \to a$ for all copies of b, otherwise the string cannot be communicated. Thus, we return to the skin membrane the string a^{2m} and the process is iterated. □

7 Final Remarks; Topics for Further Research

We have considered here a variant of rewriting P systems where communication is controlled by the contents of the strings, not by the rules used for obtaining these strings. Specifically, permitting and forbidding conditions were defined, depending on the symbols or the substrings of a given string, or depending on the shape of the string (whether or not it is of a given pattern). Several new characterizations of recursively enumerable languages were obtained, but the power of many classes of systems (especially with a small number of membranes) has remained to be clarified.

This approach can be seen as a counterpart of the approach in [3], where the use of rules (processing multisets, not strings as here) is controlled in a similar way, by the contents of a given membrane.

On the other hand, this work proceeds in the line of research on constraining application of context-free rules by the shape of the string to which a production is applied or into which an application results.

In [2] this was achieved in a setting where only one string was present and only a level was considered. The basic model proposd there was equivalent to what we could define as $ERP_1(patt, empty)$. This work shows how communication allows the complexity of the check on the produced words to be reduced, by distributing it over several membranes.

We have considered here only the rewriting case, but the same idea can be explored for all types of string-processing operations, in particular, for *splicing*. Actually, many other *research topics* remain to be investigated. We have already mentioned the need to further examine the families considered in Section 5, to improve the bounds in Theorems 3, 4, 5, and to find a characterization of RE as in Theorem 6 with a bounded number of membranes.

Here are three further research topics.

Remember that we have said that when a string fulfills both *in* and *out* conditions, it will go to one direction, nondeterministically chosen. An attractive case would be to *replicate* the string and send copies of it both out of the current membrane and to a lower level membrane. In this way we can produce additional strings, which is in general useful for solving NP-complete problems in polynomial (often, linear) time, by creating an exponential space and trading space for time (see [18] and references therein; in particular, this is the case of [12]).

In rewriting P systems we process in a parallel way different strings, present in the same membrane or in different membranes, but each string is rewritten in a sequential manner. The case of parallel rewriting, like in L systems, was considered in [11], with the following way of defining the communication: one counts how many rules from those applied to a string indicate to send the obtained string *out*, how many *in*, and how many *here*, and we send the string to the place which was indicated a larget number of times. This looks rather artificial; in the conditional case, the conditions are checked on the string obtained by rewriting, hence this difficulty does not appear, we can rewrite the string in any manner we like.

Finally, we can use patterns not only for defining the communication targets, but also for modifying the strings, thus replacing rewriting by another type of operation. This way of "growing" strings was already explored in the so-called *pattern grammars*, [7], or in other types of language generating mechanisms based on patterns, see [15]. The idea is simple: start with finite sets of terminal strings and of patterns in each membrane, interpret the variables from patterns by means of the available strings, and then evaluate the communication conditions; the terminals from a region can be variables in another region and conversely. In this way, all the work of the system would be based on using patterns.

References

1. D. Angluin, Finding Patterns Common to a Set of Strings, *J. Comput. System Sci.*, 21 (1980), 46–62.
2. P. Bottoni, A. Labella, P. Mussio, Gh. Păun, Pattern Control on Derivation in Context-Free Rewriting, *J. Automata, Languages, Combinatorics*, 3, 1, (1998), 3–28.
3. P. Bottoni, C. Martin-Vide, Gh. Păun, G. Rozenberg, Membrane Systems with Promoters/Inhibitors, submitted, 2000.
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. J. Dassow, Gh. Păun, A. Salomaa, Grammars Based on Patterns, *Intern. J. Found. Comput. Sci.*, 4, 1 (1993), 1–14.
8. R. Freund, C. Martin-Vide, Gh. Păun, Computing with Membranes: Three More Collapsing Hierarchies, submitted, 2000.
9. R. Freund, Gh. Păun, On the Number of Non-terminals in Graph-Controlled, Programmed, and Matrix Grammars, submitted, 2000.
10. D. Hauschild, M. Jantzen, Petri Nets Algorithms in the Theory of Matrix Grammars, *Acta Informatica*, 31 (1994), 719–728.
11. 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.
12. S.N. Krishna, R. Rama, P Systems with Replicated Rewriting, *J. Automata, Languages, Conbinatorics*, to appear.
13. 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.
14. C. Martin-Vide, Gh. Păun, Computing with Membranes. One More Collapsing Hierarchy, *Bulletin of the EATCS*, 72 (2000).
15. V. Mitrana, Patterns and Languages. An Overview, *Grammars*, 2, 2 (1999), 149–173.
16. Gh. Păun, Six Nonterminals Are Enough for Generating Each r.e. Language by a Matrix Grammar, *Intern. J. Computer Math.*, 15 (1984), 23–37.

17. 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).
18. 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.
19. G. Rozenberg, A. Salomaa, Eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.
20. 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.