P systems with active membranes and without polarizations — **Source link** ↗

Rudolf Freund, Andrei Păun

Institutions: Vienna University of Technology, Louisiana Tech University

Published on: 01 Sep 2005 - Soft Computing

Topics: Membrane computing, Recursively enumerable language and Membrane

Related papers:

- Computing with Membranes

- The power of communication: P systems with symport/antiport

- Computation: Finite and Infinite Machines

- Tissue P systems

- On membrane computing based on splicing

P Systems with Active Membranes and Without Polarizations

Rudolf FREUND

Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9, A-1040 Wien, Austria
E-mail: `rudi@emcc.at`

Andrei PĂUN

Department of Computer Science
Louisiana Tech University, Ruston
PO Box 10348, Louisiana, LA-71272 USA
E-mail: `apaun@latech.edu`

Abstract. P systems with active membranes but without using electrical charges (polarizations) are shown to be complete for generating recursively enumerable string languages when working on string objects and using only rules with membrane transitions as well as rules with membrane dissolving and elementary membrane division, but also when using various other kinds of rules, even including a new type of rules allowing for membrane generation. Especially, allowing for changing membrane labels turns out to be a very powerful control feature.

1 Introduction

In [8] membrane systems (then called P systems) were introduced as bio-inspired computing devices that work in a parallel and distributed way (see [11] for a comprehensive overview and [7] for current developments in the area). P systems with active membranes were introduced (e.g., see [11]; for variants solving NP complete problems see, e.g., [5], [10]) with rules for:

(a) rewriting multisets;
(b) introducing objects into membranes;
(c) sending objects out of membranes;
(d) dissolving membranes;
(e) dividing elementary membranes;
(f) dividing non-elementary membranes.

All these rules were associated with membranes not only having a specific label, but also having assigned an electrical charge (also called polarization), which could be $+$, 0, and $-$. The rules of the forms (b) to (c) involving membranes could change the polarization of the involved membrane(s), but never changed the label(s) of the membranes.

Throughout this paper we investigate some special variants of P systems with active membranes with rules especially including division of elementary membranes, but without changing

polarizations, which in fact means that we simply may forget the polarizations of membranes. Moreover, we shall also introduce membrane generation rules. In contrast to the original model, where P systems with active membranes were working on symbol objects, in this paper we consider P systems with active membranes working on string objects. Even without using the feature of changing membrane labels computational completeness can already be achieved with a quite small number of active membranes for different variants of P systems with active membranes using various kinds of rules. Allowing for changing membrane labels, the number of membranes can be reduced significantly, sometimes even to one, when using only part of the possible forms of rules.

2   Preliminaries

Before proceeding to a formal description of matrix grammars, we fix some basic notations first. For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation; the *empty string* is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. Any subset of V^+ is called a λ-*free (string) language.* Moreover, by $\mathbf{N}$ we denote the set of positive integers (or natural numbers). The family of λ-free recursively enumerable languages is denoted by RE, the family of sets of λ-free recursively enumerable languages over a one-letter alphabet by NRE (in fact, this family corresponds to the set of recursively enumerable sets of natural numbers). For more notions as well as basic results from the theory of formal languages, the reader is referred to [1], [4], [6], and [13].

We now recall the Z-binary normal form for matrix grammars, which will be used in the following proofs. Consider a matrix grammar with appearance checking $G = (N, T, S, M, F)$, where N and T are the sets of terminal and non-terminal symbols, respectively, S is the start symbol, M is the set of matrices, and F is the set of productions that can be used in the appearance checking mode. We say that G is in the *Z-binary normal form* if $N = N_1 \cup N_2 \cup \{S, Z, \#\}$, with these three sets being 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 $X, Y \in N_1$, $A \in N_2$, $x \in (N_2 \cup T)^*$, $|x| \leq 2$,

3. $(X \to Y, A \to \#)$, with $X \in N_1, Y \in N_1 \cup \{Z\}$, $A \in N_2$,

4. $(Z \to \lambda)$.

Moreover, there is only one matrix of type 1, F consists exactly of all rules $A \to \#$ appearing in matrices of type 3, and, if a sentential form generated by G contains the symbol Z, then it is of the form Zw, for some $w \in (T \cup \{\#\})^*$ (i.e., the appearance of Z makes sure that, except for Z, all symbols are either terminal or the trap symbol $\#$); the (unique) matrix of type 4 is used only once, in the last step of a derivation. Finally, at most three non-terminal symbols are used in the appearance checking mode. The following proposition is a consequence of the results elaborated in [2] and [3]:

Proposition 2.1 *For each language $L \in RE$ there is a matrix grammar with appearance checking G in the Z-binary normal form such that $L = L(G)$.*

In several of the following proofs we will start from matrix grammars in the Z-binary normal form. In order not to repeat the same notations, we consider the following representation

(notations and labelling) for such grammars: A matrix grammar with appearance checking in the Z-binary normal form is always given as $G = (N, T, S, M, F)$, with $N = N_1 \cup N_2 \cup \{S, Z, \#\}$, and with $n + 2$ matrices in M, injectively labelled with $m_0, m_1, \ldots, m_n, m_{n+1}$; the matrix (of type 1) $m_0 : (S \rightarrow X_{init} A_{init})$ is the initial one, with X_{init} a given symbol from N_1 and A_{init} a given symbol from N_2; the next k matrices (of type 2) contain only rules without appearance checking, $m_i : (X \rightarrow Y, A \rightarrow x)$, $1 \leq i \leq k$, where $X, Y \in N_1$, and $A \in N_2$, $x \in (N_2 \cup T)^*$, $|x| \leq 2$; the next $n - k$ matrices (of type 3) have rules to be applied in the appearance checking mode, $m_i : (X \rightarrow Y, A \rightarrow \#)$, $k + 1 \leq i \leq n$, with $X \in N_1$, $Y \in N_1 \cup \{Z\}$, $A \in N_2$. Finally, $m_{n+1} : (Z \rightarrow \lambda)$ is the unique terminal matrix (of type 4). Thus, when we will say "a matrix grammar in the Z-binary normal form, *with the standard notations/representation*" we will mean the notations and the conventions given here.

3   P Systems with Active Membranes and String Objects

A *P system with active membranes and string objects* (for sake of simplicity, we often will only use the notion P system) is a construct

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

where: $m \geq 1$; V is an alphabet (the *total alphabet* of the system); $T \subseteq V$ (the *terminal* alphabet); H is a finite set of *labels* for membranes; μ is a *membrane structure*, consisting of m membranes, labelled with elements of H; $w_1, \ldots, w_m$ are finite multisets of words over V (describing the *initial objects in the membranes* placed in the m regions of μ); R is a finite set of *rules*, of the following forms:

(sa) $[A]_h \longrightarrow [v]_{h'}$, where $A \in V$, $v \in V^*$, $h, h' \in H$ (a letter A is rewritten into v in a membrane labelled with h thereby changing its label to h');

(sb) $A[\ ]_h \longrightarrow [v]_{h'}$, where $A \in V$, $v \in V^*$, $h, h' \in H$ (a letter A is rewritten into v and then the resulting word is sent into a membrane labelled with h thereby changing its label to h'),

(sc) $[A]_h \rightarrow v[\ ]_{h'}$, where $A \in V$, $v \in V^*$, $h, h' \in H$ (an object A is rewritten into v and then the resulting string is sent out of the membrane labelled with h thereby changing its label to h'),

(sd) $[A]_h \rightarrow v$, where $A \in V$, $v \in V^*$, $h \in H$ (an object A is rewritten into v at the same time dissolving the surrounding membrane labelled with h),

(se) $[A]_h \rightarrow [v]_{h'}[v']_{h''}$, where $A \in V$, $v, v' \in V^*$, $h, h', h'' \in H$

(2-division rule for elementary membranes; in reaction with an object, the membrane is divided into two membranes with, maybe, different labels; the word containing the letter A specified in the rule is replaced in the corresponding words in the two new membranes by possibly new substrings v, v'; at the same time, all the objects except for the word containing the letter A that started the membrane division are duplicated into the two new membranes),

(sg) $[A]_h \longrightarrow [[v]_{h''}]_{h'}$, where $A \in V$, $v \in V^*$, $h, h', h'' \in H$ (a letter A in a membrane labelled with h by changing its label to h' is rewritten into v and then the resulting word is sent into a newly generated membrane labelled with h'').

195

We refer to [10], [11], and [12] for a more precise definition of the way originally P systems with active membranes were supposed to work; here we only informally describe the way of passing from one configuration of the system to the next one. The difference between the original model and our proposed model mainly lies in the fact that we work on strings rather than with symbols (therefore, the types of rules carry the additional marking s); moreover, in describing the membrane structure as well as the rules we only label the right-hand brackets of the matching pairs.

The rules are applied in a maximally parallel manner, yet with the following restrictions: The rules are applied "from bottom up", in one step, starting with the rules of the innermost region and, then, level by level until the region of the skin membrane is reached. In each region, all strings which can evolve using a rule of the forms (sa), (sb), (sc), (sg) can and have to evolve if they do not change the label of the surrounding membrane, yet (at most) only one string (afterwards!) may take a rule changing the label of the surrounding membrane or dissolve it or divide it. Rules of type (sa), (sb), (sc), (sg) can be used in parallel as long as they do not change the label of the membrane they affect. At most one rule of type (se) can be applied to one selected string after the evolution of all other strings in the membrane region not changing the membrane label or dissolving the membrane; if a membrane with label h is divided by a rule of type (se), which involves a word containing letter A, then all other words in membrane h are introduced in each of the resulting membranes.

The rules associated with a membrane labelled with h are used for all copies of this membrane; it does not matter whether the membrane is an initial one or it was obtained by membrane division or membrane generation. The skin membrane can never divide (nor dissolve). The result of a halting computation (no rule can be used in the last configuration) consists of all the words over the terminal alphabet T which can be found in the region of the skin membrane in the last configuration of a halting computation.

By $L(\Pi)$ we denote language generated as described above by a P system Π. The words not from T^* which remain in the region of the skin membrane at the end of a halting computation are not contributing to the generated language; if a computation goes forever, then it provides no output, it does not contribute to the set $L(\Pi)$.

Now let D be a non-empty subset of $\{sa, sb, sc, sd, se, sg, sa_0, sb_0, sc_0, sd_0, se_0, sg_0\}$ – the subscript 0 indicates that the corresponding type of rules does not change the labels of membranes. Then, by

$$LPA_{(n_1,n_2,n_3,n_4)}(D)$$

we denote the family of languages generated by P systems with active membranes and string objects as defined above using only rules of types from D, with having at most n_1 membranes in the initial configuration, the maximum number of membranes in the system at any given time being at most n_2 and the number of membranes in a halting configuration being at most n_3 and with at most n_4 labels in H. If one of these numbers n_1, n_2, n_3, n_4 can be arbitrarily large, then we write $*$ instead of this number.

We now establish the main result of our paper showing that when using rules of types sb_0, sc_0, sd_0, se_0 (i.e., only rules without changing membrane labels) we obtain computational completeness with a bounded number of membranes and a bounded number of membrane labels:

Theorem 3.1 *For all vectors* $(n_1, n_2, n_3, n_4) \in \mathbf{N}^4$ *with* $(n_1, n_2, n_3, n_4) \geq (7, 8, 7, 7)$ *we have*

$$LPA_{(n_1,n_2,n_3,n_4)}(\{sb_0, sc_0, sd_0, se_0\}) = RE.$$

Proof. We only prove the inclusion

$$RE \subseteq LPA_{(7,8,7,7)} \left(\{ sb_0, sc_0, sd_0, se_0 \} \right).$$

For that purpose, let us consider a matrix grammar with appearance checking

$$G = (N_1 \cup N_2 \cup \{S, Z, \#\}, T, S, M, F)$$

in the Z-binary normal form with the standard notations/representation; we now construct the following P system with active membranes and with string objects:

$$\Pi = (V, T, H, \mu, w_0, w_1, w_2, w_3, w_{A_1}, w_{A_2}, w_{A_3}, R),$$

where:

$$
\begin{aligned}
V &= N_1 \cup N_2 \cup T \cup \{X', X'', X''' \mid X \in N_1\} \cup \{\overline{A}, \overline{\overline{A}} \mid A \in N_2\} \\
&\cup \{X_j^i, A_j^i \mid X \in N_1, A \in N_2, 1 \le i \le n, 0 \le j \le k\} \cup \{Z, \#\}, \\
H &= \{0, 1, 2, 3, A_1, A_2, A_3\}, \\
\mu &= [[[[\ ]_3]_2]_1[\ ]_{A_1}[\ ]_{A_2}[\ ]_{A_3}]_0, \\
w_0 &= X_{init} A_{init}, \\
w_1 &= w_2 = w_3 = w_{A_1} = w_{A_2} = w_{A_3} = \emptyset, \\
R &= \{X[\ ]_1 \to [X_i^i]_1, \ A[\ ]_2 \to [A_i^i]_2, \ X_j^i[\ ]_3 \to [X_{j-1}^i]_3, \\
&\quad [A_j^i]_3 \to [\ ]_3 A_{j-1}^i, \ [X_0^i]_2 \to Y[\ ]_2, \ [A_0^i]_1 \to \alpha_1 \alpha_2[\ ]_1, \ [A_0^i]_3 \to \#[\ ]_3, \\
&\quad [A_j^i]_2 \to \#[\ ]_2 \mid m_i : (X \to Y, A \to \alpha_1 \alpha_2) \in M, 1 \le i, j \le k\} \\
&\cup \{X[\ ]_A \to [Y']_A, \ [Y']_A \to [Y'']_A[Y''']_A, \ [Y'']_A \to Y[\ ]_A, \\
&\quad [Y''']_A \to \overline{A}, \ \overline{A}[\ ]_A \to [\overline{\overline{A}}]_A, \ [A]_A \to \# \mid m_i : (X \to Y, A \to \#) \in M\} \\
&\cup \{\#[\ ]_1 \to [\#]_1, \ [\#]_1 \to \#[\ ]_1, \ \#[\ ]_2 \to [\#]_2, \ [\#]_2 \to \#[\ ]_2, \\
&\quad \#[\ ]_3 \to [\#]_3, \ [\#]_3 \to \#[\ ]_3, \ Z[\ ]_1 \to [Z]_1, \ [Z]_1 \to \lambda[\ ]_1\},
\end{aligned}
$$

and A_1, A_2, A_3 are the only non-terminal symbols used in the appearance checking mode in the given matrix grammar G. We will prove now that $L(\Pi) = L(G)$.

To prove first that all the words generated by the matrix grammar G are also in the language generated by the P system Π we list and describe the rules in R defined specifically for each rule in the matrix grammar:

The single matrix of type 1 in G is simulated just by the fact that the word $X_{init} A_{init}$ is initially present in the system in membrane 0 (i.e., the initial multiset in the skin membrane consists of only one word in one copy, whereas the initial multisets in all other regions are empty).

The matrices of type 2 from G are simulated by the following rules, the simulation taking place in the membranes 1, 2, and 3:

For a rule $m_i : (X \to Y, A \to \alpha_1 \alpha_2)$, $1 \le i \le k$, we start with replacing X by X_i^i when X is passing through membrane 1 :

$$X[\ ]_1 \to [_1 X_i^i]_1$$

and then the non-terminal symbol from N_2 that appears in the matrix, A, is transformed into A_i^i when pushing the string into membrane 2:

$$A[\ ]_2 \to [A_i^i]_2$$

The string will move back and forth between the membranes 2 and 3 thereby decreasing the subscripts of X and A:

$$X_j^i[\ ]_3 \rightarrow [X_{j-1}^i]_3, \ [A_j^i]_3 \rightarrow [\ ]_3 A_{j-1}^i$$

When the subscript of X reaches 0 we can move the string from membrane 2 to membrane 1 using the rule

$$[X_0^i]_2 \rightarrow Y[\ ]_2.$$

At that point also the subscript of A has to be 0 (i.e., initially the subscripts were the same for X and for A) in order to guarantee that the simulation was correct; using

$$[A_0^i]_1 \rightarrow \alpha_1 \alpha_2 [\ ]_1$$

we send out the produced string to membrane 0. One can notice that after all these steps we have replaced X by Y and A by $\alpha_1 \alpha_2$, hence, the matrix was simulated correctly. Another important fact is that we are in a similar configuration of the system as before: we have the string containing one symbol from N_1 and some symbols from N_2 and/or T in membrane 0, so the process can be iterated until we reach a final configuration.

The matrices of type 3 from G are also simulated by using several rules now taking place in the membranes labelled with A_1, A_2, A_3 and by dividing these membranes. One should notice that the labels of these membranes are the only symbols from N_2 used in the appearance checking mode in G.

For a rule $m_i : (X \rightarrow Y, A \rightarrow \#)$ we perform the following steps: First we send the string into the membrane carrying the label A corresponding to the symbol to be used in the appearance checking mode, i.e., we use

$$X[\ ]_A \rightarrow [Y']_A;$$

then this particular membrane with label A divides, so in the next step we will have two membranes with the same label A and containing each a word that differs from the other one by one letter:

$$[Y']_A \rightarrow [Y'']_A[Y''']_A$$

In fact, we here use membrane division to produce two strings: one that will continue the simulation and the second one will check the correctness of the simulation. The word containing Y'' will continue as if everything were okay,

$$[Y'']_A \rightarrow Y[\ ]_A,$$

and the second word, the one containing Y''', will first dissolve the second membrane labelled A by using

$$[Y''']_A \rightarrow \overline{A}$$

and then enter the other membrane labelled with A using the rule

$$\overline{A}[\ ]_A \rightarrow [\overline{\overline{A}}]_A.$$

In the next step the concrete appearance checking is simulated by

$$[A]_A \rightarrow \#.$$

If A was present in the word when we started the simulation of the rule, then now we will produce the trap-symbol $\#$ which will effectively kill the computation, since this string will move forever

between membranes 0, 1, 2, and 3, thus we will never reach a halting configuration. On the other hand, if A was not present in the word, then this "checking" word will be stuck in membrane A, but the "evolving" word can continue the process since it is already in membrane 0. One can notice that the simulation also in this case is performed in such a way that at the end we will have the string in membrane 0 and will have also at most one symbol from $N_1 \cup \{Z\}$ in that string, so again, we can iterate the process of simulating matrices from G in this case, too.

It now remains to discuss the halting process. As we know from the special features of the Z-normal form, the non-terminal symbol Z appears only at the end of a derivation at the same time guaranteeing that except for terminal symbols only the trap symbol may appear in the underlying sentential form. Hence, we only have to remove Z, which is performed in the following way:

First the string is sent to membrane 1 by the rule

$$Z[\ ]_1 \rightarrow [Z]_1.$$

Then the symbol Z is eliminated by sendig out the string to the skin region again:

$$[Z]_1 \rightarrow \lambda[\ ]_1$$

If the underlying string contains no trap symbol $\#$, then the resulting string is terminal and the computation halts yielding this string as the result of the computation in the P system Π.

After this discussion it becomes clear that $L(G) \subseteq L(\Pi)$. Let us now consider the converse inclusion $L(\Pi) \subseteq L(G)$: At the end of each correct simulation, in membrane 0 we have a word containing one symbol from N_1 followed by symbols from $N_2 \cup T$. Let us take a closer look to the simulation of matrices of type 2 from G: Once this simulation has been started, the string will go from membrane 0 to membranes $1, 2$ and then start oscillating between membranes 2 and 3. If the subscripts of both X_i^i and A_j^j matched the same matrix $i = j$ then the string will be able to return in membrane 0, but if the subscripts do not match, $i \neq j$, then we have two cases:

If $i > j$, then at some point the subscript of A reaches 0, and the word $X_{i-j}^i...A_0^j...$ cannot exit membrane 3 by decreasing the subscript of A, so the only rule that can be applied is

$$[A_0^j]_3 \rightarrow \#[\ ]_3$$

thus producing the trap symbol and, therefore, this particular computation will yield no output.

If $i < j$, then the subscript of X will first reach 0, so next we can exit membrane 2 by applying

$$[X_0^i]_2 \rightarrow Y[\ ]_2.$$

In this moment we are in membrane 1 with the word $Y...A_{j-i}^j....$ Here a rule

$$B_k^k[\ ]_2 \rightarrow [B_k^k]_2$$

can be applied, but in the next step the word does not contain a variant of X in order to be able to enter membrane 3, so the only rule possible at this point is

$$[A_j^i]_2 \rightarrow \#[\ ]_2$$

which produces the trap symbol. We showed that if $i \neq j$ then we produce the trap symbol in all cases. This was greatly helped by the fact that when moving from membrane 0 to membrane

3 the string was changed: membrane 1 changed X into X_i^i and membrane 2 changed A into A_i^i. When exiting, the string was changed in the reversed order: X_0^i was changed when exiting from membrane 2 into Y and A_0^i when exiting 1 into $\alpha_1 \alpha_2$, thus ensuring that the word cannot "start" a second simulation in the original simulation.

Let us now check the simulation of the appearance checking case: The simulation proceeds straight-forward without any choice in the application of the rules; there is a membrane division rule, so we will show that there are no other membranes created during the life of the second membrane A newly created (so that our result holds in the number of maximal membranes created during a computation). After dividing the membrane A starting from the string Xw, in the very next step the two words Yw and $\overline{A}w$ arrive in membrane 0, one just exiting its surrounding membrane labelled with A, the other one dissolving it. At this moment another simulation can start from the string Yw. If the new simulation involves a matrix of type 1 or type 2 or Z (finishing by entering membrane 1) or an appearance checking on another symbol from $\{A_1, A_2, A_3\} - A$ then everything is okay. If the new simulation is one for a matrix of type 3 involving the same appearance checking symbol A, then both Yw and $\overline{A}w$ will enter membrane A yielding

$$U'w \text{ and } \overline{\overline{A}}w$$

for some $U \in N_1$ in membrane A; if again $U'w$ takes possession of membrane A and divides it, we obtain two membranes labelled by A, one containing (besides all other copies already stored there) $U''w$ and $\overline{\overline{A}}w$, the other one containing $U'''w$ and $\overline{\overline{A}}w$; the worst case scenario now is that $U'''w$ takes possession of its membrane A, thus yielding $\overline{A}w$ and $\overline{\overline{A}}w$ in the skin membrane, whereas $U''w$ leaves the other membrane A yielding Uw in the skin membrane and then - but in the same derivation step of Π - the copy of $\overline{\overline{A}}w$ in membrane A dissolves the membrane just in case A occurs in w, otherwise it will remain there without being able to do any harm in succeeding derivation steps. The copy of $\overline{\overline{A}}w$ in the skin membrane gets stuck there, and for Uw, $\overline{A}w$ we have just the same situation as before. Hence, also the simulation of the matrices of type 3 is performed correctly, without producing any other "unwanted" strings.

Finally, we would like to note that introducing the trap symbol $\#$ in the system will result in an infinite computation because of the following rules:

$$\#[\ ]_1 \to [\#]_1, \ [\#]_1 \to \#[\ ]_1,$$
$$\#[\ ]_2 \to [\#]_2, \ [\#]_2 \to \#[\ ]_2,$$
$$\#[\ ]_3 \to [\#]_3, \ [\#]_3 \to \#[\ ]_3.$$

It is clear now that the constructed P system yields the same language as the matrix grammar, i.e., $L(\Pi) = L(G)$, hence, the theorem is proved. □

For languages over a one-letter alphabet, we can reduce the number of membranes by one:

Corollary 3.1 *For all vectors $(n_1, n_2, n_3, n_4) \in \mathbf{N}^4$ with $(n_1, n_2, n_3, n_4) \geq (6, 7, 6, 6)$ we have*

$$LPA_{(n_1, n_2, n_3, n_4)}(\{sb_0, sc_0, sd_0, se_0\}) = NRE.$$

Proof. According to the results proved in [3], for recursively enumerable languages over a one-letter alphabet only two variables used in the appearance checking mode are needed for matrix grammars generating these languages. Therefore A_3 and the related rules are not necessary in the construction given in the proof of Theorem 3.1, which already proves the corollary. □

Allowing for membrane generating rules allows for considerably decreasing the number of membranes:

Theorem 3.2 *For all vectors* $(n_1, n_2, n_3, n_4) \in \mathbf{N}^4$ *with* $(n_1, n_2, n_3, n_4) \geq (1, 4, 1, 7)$ *we have*

$$LPA_{(n_1,n_2,n_3,n_4)}\left(\{sb_0, sc_0, sd_0, se_0, sg_0\}\right) = RE.$$

Proof. We only prove the inclusion

$$RE \subseteq LPA_{(1,4,1,7)}\left(\{sb_0, sc_0, sd_0, se_0, sg_0\}\right).$$

For that purpose, let us consider a matrix grammar with appearance checking

$$G = (N_1 \cup N_2 \cup \{S, Z, \#\}, T, S, M, F)$$

in the Z-binary normal form with the standard notations/representation; we now construct the following P system with active membranes and with string objects:

$$\Pi = (V, T, H, \mu, w_0, w_1, w_2, w_3, w_{A_1}, w_{A_2}, w_{A_3}, R),$$

where:

$$
\begin{aligned}
V \;=\; & N_1 \cup N_2 \cup T \cup \{X', X'', X''', \overline{X}, \overline{\overline{X}}, \tilde{X}, \hat{X} \mid X \in N_1\} \cup \{\overline{A}, \overline{\overline{A}} \mid A \in N_2\} \\
\cup \;& \{X_j^i, A_j^i \mid X \in N_1, A \in N_2, 1 \leq i \leq n, 0 \leq j \leq k\} \cup \{Z, \#\}, \\
H \;=\; & \{0, 1, 2, 3, A_1, A_2, A_3\}, \\
\mu \;=\; & [[[[\;]_3]_2]_1[\;]_{A_1}[\;]_{A_2}[\;]_{A_3}]_0, \\
w_0 \;=\; & X_{init} A_{init}, \\
w_1 \;=\; & w_2 = w_3 = w_{A_1} = w_{A_2} = w_{A_3} = \emptyset, \\
R \;=\; & \{[X]_0 \to [[X_i^i]_1]_0, \; [A]_1 \to [[A_i^i]_2]_1, \; [X_i^i]_2 \to [[X_{i-1}^i]_3]_2, \; X_j^i[\;]_3 \to [X_{j-1}^i]_3, \\
& [A_j^i]_3 \to [\;]_3 A_{j-1}^i, \; [A_1^i]_3 \to A_0^i, \; [X_0^i]_2 \to Y, \; [A_0^i]_1 \to \alpha_1\alpha_2, \\
& [A_0^i]_3 \to \#, \; [A_j^i]_2 \to \#, \; [\#]_1 \to \#, \; [\#]_0 \to [[\#]_1]_0 \mid \\
& m_i : (X \to Y, A \to \alpha_1\alpha_2) \in M, \; 1 \leq i, j \leq k\} \\
\cup \;& \{[X]_0 \to [[Y']_A]_0, \; [Y']_A \to [Y'']_A[Y''']_A, \; [Y'']_A \to \bar{Y}[\;]_A, \; [Y''']_A \to \overline{A}, \\
& \bar{Y}[\;]_A \to [\overline{\overline{Y}}]_A, \; \overline{A}[\;]_A \to [\overline{\overline{A}}]_A, \; [\overline{\overline{Y}}]_A \to \tilde{Y}[\;]_A, \; [A]_A \to \#, \; \tilde{Y}[\;]_A \to [\hat{Y}]_A \\
& \tilde{Y}[\;]_A \to [\hat{Y}]_A, \; [\hat{Y}]_A \to Y \mid m_i : (X \to Y, A \to \#) \in M\} \\
\cup \;& \{[Z]_0 \to [[Z]_1]_0, \; [Z]_1 \to \lambda\},
\end{aligned}
$$

and A_1, A_2, A_3 are the only non-terminal symbols used in the appearance checking mode in the given matrix grammar G. Following the explanations given in the proof of Theorem 3.1 one can easily prove again that $L(\Pi) = L(G)$. The main difference is that the membranes 1, 2, and 3 as well as the membranes A checking for the appearance of the non-terminal symbol A are created dynamically and after having fulfilled their duty within the simulation of one derivation step of G in Π are dissolved again: For the simulation of a rule of type 2 the membranes 1, 2, and 3 are created dynamically in the first three steps of the simulation and dissolved again in the last three steps. In the same way, for simulating a matrix of type 3 checking for the appearance of the non-terminal symbol A, the corresponding membrane A is dynamically generated in the first step of the simulation and dissolved five steps later, in the meantime giving the checking string the possiblity to generate the trap symbol $\#$ in case A occurs in the current sentential form; observe that the rules

$$\bar{Y}[\;]_A \to [\overline{\overline{Y}}]_A, \; \overline{A}[\;]_A \to [\overline{\overline{A}}]_A$$

enter membrane A in parallel and in the next step,

$$[\overline{\overline{Y}}]_A \to \tilde{Y}[\ ]_A$$

is applied eventually followed in the same step by

$$[A]_A \to \#$$

in case A occurs in the underlying word. If this last production is not applied, we are able to successfully finish the simulation of this appearance checking matrix by using the rules

$$\tilde{Y}[\ ]_A \to [\hat{Y}]_A, \ [\hat{Y}]_A \to Y.$$

As the rest of the simulation is quite similar to the simulation already described in the proof of Theorem 3.1, we leave the remaining details of explanation for this proof to the reader. $\qquad\square$

For languages over a one-letter alphabet, we can reduce the number of membrane labels by one:

Corollary 3.2 *For all vectors* $(n_1, n_2, n_3, n_4) \in \mathbf{N}^4$ *with* $(n_1, n_2, n_3, n_4) \geq (1, 4, 1, 6)$ *we have*

$$LPA_{(n_1,n_2,n_3,n_4)} \left(\{sb_0, sc_0, sd_0, se_0, sg_0\} \right) = NRE.$$

Proof. As the proof of Corollary 3.1 directly followed from the proof of Theorem 3.1, the result of this corollary directly follows from the proof of Theorem 3.2. $\qquad\square$

So far we have only considered types of rules not changing membrane labels; as the following theorem shows, changing of membrane labels provides a powerful control mechanism:

Theorem 3.3 *For all vectors* $(n_1, n_2, n_3) \in \mathbf{N}^3$ *we have*

$$LPA_{(n_1,n_2,n_3,*)} \left(\{sa\} \right) = RE.$$

Proof. We only prove the inclusion

$$RE \subseteq LPA_{(1,1,1,*)} \left(\{sa\} \right).$$

For that purpose, let us consider a matrix grammar with appearance checking

$$G = (N_1 \cup N_2 \cup \{S, Z, \#\}, T, S, M, F)$$

in the Z-binary normal form with the standard notations/representation; we now construct the following P system with active membranes and with string objects:

$$\Pi = (V, T, H, \mu, w_0, R),$$

where:

$$
\begin{aligned}
V &= N_1 \cup N_2 \cup T \cup \{X', X'', X''' \mid X \in N_1\} \cup \{Z, \#\}, \\
H &= \{0\} \cup \{i, \tilde{i} \mid 1 \leq i \leq k\} \cup \{i \mid k+1 \leq i \leq n\}, \\
\mu &= [\ ]_0, \\
w_0 &= \{X_{init}, A_{init}\}, \\
R &= \{[X]_0 \to [Y']_i, \ [A]_i \to [\alpha_1\alpha_2]_{\tilde{i}}, \ [Y']_{\tilde{i}} \to [Y]_0 \mid \\
&\qquad m_i : (X \to Y, A \to \alpha_1\alpha_2) \in M\} \\
&\cup \ \{[X]_0 \to [Y']_i, \ [Y']_i \to [Y'']_i, \ [Y'']_i \to [Y]_0, \ [A]_i \to [\#]_0 \mid \\
&\qquad m_i : (X \to Y, A \to \#) \in M\} \\
&\cup \ \{[Z]_0 \to [\ ]_0, \ [\#]_0 \to [\#]_0\},
\end{aligned}
$$

and A_1, A_2, A_3 are the only non-terminal symbols used in the appearance checking mode in the given matrix grammar G. One can easily prove again that $L(\Pi) = L(G)$:

We start with the two words X_{init}, A_{init} in the skin membrane. For simulating a rule of type 2 $m_i : (X \to Y, A \to \alpha_1 \alpha_2)$, the corresponding membrane label i is taken for the skin membrane by using

$$[X]_0 \to [Y']_i.$$

Changing the label when applying

$$[A]_i \to [\alpha_1 \alpha_2]_{\tilde{i}}$$

guarantees that only one non-terminal symbol A can be replaced. Label $\tilde{i}$ then allows to finish the simulation by applying

$$[Y']_{\tilde{i}} \to [Y]_0.$$

For simulating a matrix of type 3 $m_i : (X \to Y, A \to \#)$ checking for the appearance of the non-terminal symbol A, the corresponding membrane label i is obtained by

$$[X]_0 \to [Y']_i$$

and then we check for the appearance of A :

$$[A]_i \to [\#]_0$$

(which, whenever applied, yields an infinite computation by the rule$[\#]_0 \to [\#]_0$) is possibly applied in the same derivation step of Π immediately after having changed Y' to Y'' using

$$[Y']_i \to [Y'']_i.$$

In the last step of the simulation, we reset the label of the skin membrane to 0 by applying

$$[Y'']_i \to [Y]_0.$$

A successful computation in Π ends with the application of

$$[Z]_0 \to [\ ]_0.$$

It is easy to see that the P system Π can simulate the matrix grammar G and in total we obtain $L(\Pi) = L(G)$. $\qquad\square$

In the preceding proof we only used one type of rules allowing for changing membrane labels; although needing only the simplest membrane structure, we had to pay the price for that by not being able to bound the number of membrane labels.

4 Conclusion and Future Research

For several variants of P systems with active membranes working on strings we have shown computational completeness. As the results proved in this paper show, there is a trade-off between using complex rules like membrane division and the powerful control feature of changing membrane labels.

A lot of problems have remained open in this paper, especially

- concerning the minimal descriptional complexity of the systems investigated in this paper,

- with respect to find the borderline between computational completeness and non-completeness,

- concerning the effect of omitting part of the possible forms of rules or replacing prt of them by other forms.

Moreover, several variants of rule combinations should also be investigated for P systems with active membranes working on symbol objects; especially the new type of rules allowing for membrane generation as well as the feature of changing membrane labels deserve future investigations.

Acknowledgements. This paper was initiated during a visit of the second author at the Vienna University of Technology in the middle of the year 2003 and then further developed in the friendly and inspiring atmosphere of the Brainstorming Week on Membrane Computing taking place at Sevilla in the first week of February 2004, where the first author could take advantage of the fruitful discussions with several participants.

References

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

[2] Freund, R., Oswald, M.: GP systems with forbidding context. *Fundamenta Informaticae*, **49**, 1-3 (2002), 81–102.

[3] Freund, R., Martin-Vide, C., Păun, Gh.: From regulated rewriting to computing with membranes: collapsing hierarchies. *Theoretical Computer Science*, **312** (2004), 143–188.

[4] Hopcroft, J., Ulmann, J.: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[5] Krishna, S.N., Rama, R., A variant of P systems with active membranes: solving NP-complete problems. *Romanian J. of Information Science and Technology*, **2**, 4 (1999), 357–367.

[6] Martin-Vide, C., Păun, Gh.: Elements of Formal Language Theory for Membrane Computing. *Technical Report* 21/01 of the Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona (2001).

[7] The P Systems Web Page: `http://psystems.disco.unimib.it/`

[8] Păun, Gh.: Computing with membranes. *Journal of Computer and System Sciences*, **61,** 1 (2000), 108–143, and TUCS Research Report 208 (1998) (`http://www.tucs.fi`).

[9] Păun, Gh.: Computing with membranes: an introduction. *Bulletin EATCS*, **67** (1999), 139–152.

[10] Păun, Gh.: P systems with active membranes: Attacking NP complete problems, *Journal of Automata, Languages and Combinatorics*, **6,** 1 (2001), 75–90.

[11] Păun, Gh.: *Membrane Computing: An Introduction*. Springer-Verlag, Berlin, 2002.

[12] Păun, Gh., Suzuki, Y., Tanaka, H., Yokomori, T.: On the power of membrane division in P systems, *Proc. Conf. on Words, Languages, and Combinatorics*, Kyoto, 2000.

[13] Salomaa, A., Rozenberg, G. (eds.): *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1997.