One and Two Polarizations, Membrane Creation and Objects Complexity in P Systems

Artiom Alhazov[1,2], Rudolf Freund[3] and Agustín Riscos-Núñez[4]

[1] Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: artiome.alhazov@estudiants.urv.es

[2] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md

[3] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, A–1040 Vienna, Austria
E-mail: rudi@emcc.at

[4] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: ariscosn@us.es

Abstract

We improve, by using register machines, some existing universality results for specific models of P systems. P systems with membrane creation are known to generate all recursively enumerable sets of vectors of non-negative integers, even when no region (except the environment) contains more than one object of the same kind. We here show that they generate all recursively enumerable languages*, and two membrane labels* are sufficient (the same result holds for* accepting *all recursively enumerable vectors of non-negative integers). Moreover, at most* two objects *are present inside the system at any time in the generative case. Then we prove that* $10 + m$ symbols are enough to generate any recursively enumerable language over m symbols.*

P systems with active membranes without polarizations are known to generate all recursively enumerable sets of vectors of non-negative integers. We show that they generate all recursively enumerable languages*; four starting membranes* with three labels or seven starting membranes with* two labels *are sufficient. P systems with active membranes and two polarizations are known to generate/accept all recursively enumerable sets of vectors of non-negative integers, only using rules of rewriting and sending objects out. We show that accepting can be done by* deterministic *systems. Finally, remarks and open questions are presented.*

1. Introduction

P systems with symbol objects are a theoretical framework of distributed parallel multiset processing, launched by Gh. Păun in 1998 ([6]). A systematic field survey can be found in [7]; [9] contains a comprehensive bibliography.

The aim of this article is to improve descriptive complexity parameters or properties of a few universality results. More precisely, we shall speak about object complexity (bounds in the starting configuration, in any configuration, in the alphabet) and also about membrane complexity.

Let us denote the set of all recursively enumerable sets of (k-dimensional) vectors of non-negative integers by $PsRE$ ($PsRE(k)$, respectively), while the set of all recursively enumerable languages (over a k-letter alphabet) is denoted by RE ($RE(k)$, respectively).

It was shown in [1] that P systems with membrane creation generate $PsRE$, even when every region (except the environment) contains at most one object of every kind, but using an unbounded number of membrane labels. We will show that RE is generated using only two membrane labels and at most two objects present inside the system throughout any computation. The accepting case is considered, too, and again two membrane labels are enough. On the

other hand, using an unbounded membrane alphabet we can bound the symbol alphabet by $10 + m$ objects, where m is the size of the output alphabet.

We also know from [1] that P systems with active membranes without polarization generate $PsRE$, again working with an unbounded number of membranes. We will show that RE is generated by P systems with four membranes and three labels or seven membranes with two labels in the initial configuration.

As shown in [2, 3], P systems with two polarizations and rules of types (a) – rewriting – and (c) – sending an object out – generate $PsRE$ using two membranes or accept $PsRE$ using one membrane. In this article we will show that *deterministic* systems of this kind with one membrane accept $PsRE$.

2. Definitions

After some preliminary definitions, we recall basic facts of register machines and matrix grammars (without appearance checking) and give the necessary definitions for the specific models of P systems considered in this paper.

2.1. Preliminaries

The set of non-negative integers is denoted by $\mathbb{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the *empty string* is denoted by λ, and $V^* - \{\lambda\}$ is denoted by V^+. By $|x|$ we denote the length of the string x over V.

For more notions as well as basic results from the theory of formal languages, the reader is referred to [4] and [8].

2.2. Register machines

A *register machine* is a construct $M = (m, P, l_0, l_h)$, where m is the number of registers, P is a finite set of instructions injectively labelled with elements from a given set $lab(M)$, l_0 is the initial/start label, l_h is the final label.

The instructions are of the following forms:

– $l : (A(r), l', l'')$
  Add 1 to the contents of register r and proceed to the instruction (labelled with) l' or l'' (ADD instruction). For *deterministic* machines, we require $l' = l''$.

– $l : (S(r), l', l'')$
  If register r is not empty, then subtract 1 from its contents and go to instruction l', otherwise proceed to instruction l'' (conditional SUB instruction).

– $l_h : halt$
  Stop the machine.

When considering the generation of languages, we use the model of a *register machine with output tape*, which also uses a tape operation:

– $l : (write(a), l'')$
  Write symbol a on the output tape and go to l''.

We then also specify the output alphabet T in the description of the register machine with output tape, i.e., we write $M = (m, T, P, l_0, l_h)$.

The following results are folklore (e.g., see [5]):

Proposition 1 *Let $L \subseteq N^m$ be a recursively enumerable set of (vectors of) non-negative integers. Then L can be generated/accepted by a register machine/deterministic register machine with at most $m + 2$ registers; moreover, at the beginning/at the end of a computation, all registers are empty; the result/the input of a computation appears in the first m registers.*
Let $L \subseteq V^*$ be a recursively enumerable language. Then L can be generated by a register machine with output tape with 2 registers.

2.3. Matrix grammars

A context-free *matrix grammar* (without appearance checking) is a construct $G = (N, T, S, M)$ where N and T are sets of *non-terminal* and *terminal symbols,* respectively, with $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, M is a finite set of *matrices*, $M = \{m_i \mid 1 \le i \le n\}$, where the matrices m_i are sequences of the form $m_i = (m_{i,1}, \cdots, m_{i,n_i})$, $n_i \ge 1$, $1 \le i \le n$, and the $m_{i,j}$, $1 \le j \le n_i$, $1 \le i \le n$, are context-free productions over (N, T).

For $m_i = (m_{i,1}, \cdots, m_{i,n_i})$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \cdots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each j, $1 \le j \le n_i$, w_j is the result of the application of $m_{i,j}$ to w_{j-1}. The language generated by G is

$$L(G) = \{w \in T^* \mid S \Longrightarrow_{m_{i_1}} w_1 \cdots \Longrightarrow_{m_{i_k}} w_k = w,$$
$$w_j \in (N \cup T)^*, \; m_{i_j} \in M \text{ for } 1 \le j \le k, k \ge 1\}.$$

The family of languages generated by matrix grammars without appearance checking is denoted by MAT^λ. It is known that for the family of Parikh sets of languages generated by matrix grammars $PsMAT^\lambda$ we have $PsMAT^\lambda \subset PsRE$. Further details about matrix grammars can be found in [4] and in [8]. We only mention that the power of matrix grammars is not decreased if we only work with matrix grammars in the *f-binary normal form* where N is the disjoint union of N_1, N_2, and $\{S, f\}$, and M contains rules of the following forms:

1. $(S \to XA)$, with $X \in N_1$, $A \in N_2$;

2. $(X \rightarrow Y, A \rightarrow x)$, with $X \in N_1$, $Y \in N_1 \cup \{f\}$, $A \in N_2$, and $x \in (N_2 \cup T)^*, |x| \leq 2$;

3. $(f \rightarrow \lambda)$.

Moreover, there is only one matrix of type 1 and only one matrix of type 3, which is only used in the last step of a derivation yielding a terminal result.

2.4. P Systems

The first model of P systems we consider is that of *P systems with non-cooperative multiset-rewriting* rules ($ncoo$) with specifying targets (tar), also using *membrane creation* ($mcre$) and *membrane dissolution* (δ) possibilities. Such a system (of initial degree $m \geq 1$) is of the form

$$\Pi = (O, H, \mu, w_1, \cdots, w_m, R_1, \cdots, R_n)$$

where O is the alphabet of objects, H is the set of labels for membranes (we assume here that H contains n labels), μ is the initial membrane structure, consisting of m membranes labelled (not necessarily in a one-to-one manner) with elements of H, $w_1, \cdots, w_m$ are strings over O representing the multisets of objects present in the m compartments (also called regions) of μ, and $R_1, \cdots, R_n$ are the (finite) sets of rules associated with the n labels from H. These rules can be of the forms (1) $a \rightarrow v$ and (2) $a \rightarrow [_i \ b]_i$ where $v \in O^*$, $a, b \in O$, $i \in H$, and either $v \in (O \times tar)^*$ or $v \in (O \times tar)^* \{\delta\}$, with $tar = \{here, out\} \cup \{in_j \mid j \in H\}$. The presence of δ on the right-hand side of a rule means that the application of the rule leads to the dissolution of the membrane.

The meaning of a rule of type (1) is that the object a from the region associated with the rule "reacts", and as a result the objects specified by v are produced. The objects from v have associated *target commands*, of the forms *here, out*, in_j, which specify where the object should be placed: *here* means that the object remains in the region where it is produced, in_j means that it has to go to the membrane j, provided that it is directly inside the membrane where the rule is applied (otherwise the rule cannot be used), and *out* indicates that the object should exit the current membrane, going to the surrounding region – which is the environment in the case of the skin membrane of the system. In general, the indication *here* is not explicitly written. If the special symbol δ is present, this means that after using the rule the membrane is dissolved, and all its contents, objects and membranes alike, become elements of the surrounding region. The skin membrane is never dissolved. A rule $a \rightarrow [_i \ b]_i$ of type (2) means that object a produces a new membrane, with label i, containing the object b.

Knowing the label of the membrane, we also know the rules associated with it. We recall that the number of membrane labels (i.e., kinds of membranes) is n (the rules are associated with the membrane labels), while the number of membranes initially is m and can change during the computation. The rules are used in the non-deterministic (the objects and the rules are chosen non-deterministically) maximally parallel way (no further object can evolve after having chosen the objects for the rules), thus obtaining transitions from one configuration of the system to another one.

P systems with active membranes with k polarizations ($active_k$) are constructs

$$\Pi = (O, H, E, \mu, w_1, \cdots, w_m, R)$$

where the components $O, H, \mu, w_1, \cdots, w_m$ are defined in the same way as above, the membranes of μ also have associated *polarizations* from E (having cardinality k), and the rules from R are of the following forms: (a): $[\ a \rightarrow v \ ]_h^e$ – rewriting-like, (b): $a[\ ]_h^e \rightarrow [\ b]_h^{e'}$ and (c): $[\ a]_h^e \rightarrow [\ ]_h^{e'} b$ – bring an object inside the membrane/send an object out of the membrane (possibly changing its polarization), (d): $[\ a]_h^e \rightarrow b$ – dissolve the membrane, producing another object (the contents of the dissolved membrane is released to the surrounding region), (e): $[\ a]_h^e \rightarrow [\ b]_h^{e'} [\ c]_h^{e''}$ – membrane division, where two membranes with the same label (but possibly different polarizations) are produced, each containing a new object (all other objects are duplicated). In all cases, $a, b, c \in O$, $v \in O^*$, $e, e', e'' \in E$ and $h \in H$.

The rules of type (a) are applied in the maximally parallel way, while at most one rule of types $(b), (c), (d), (e)$ can be applied for each membrane at any step of the computation. If we have only one polarization, we omit specifying it (and in the literature, the subscript 0 then is usually added to the types of rules defined above, thus yielding (a_0), etc.).

In both models, when a configuration is reached where no rule can be applied, the computation stops, and the multiplicity of objects sent into environment during the computation is said to be computed by the system along that computation. By $Ps(\Pi)$ we denote the set of vectors generated in this way (by means of all computations) by a system Π. If we take into account the sequence of symbols as they are sent out into the environment (when two or more objects leave the system at the same moment, then all permutations of these objects are considered), then we obtain the string language generated by Π, denoted by $L(\Pi)$. When considering Π as an accepting system for a set of vectors, we put the input multiset into the skin membrane and accept by halting computations.

We denote the resulting families generated by such P systems by $XO_{n_1, n_2, n_3} P_{n_4, n_5, n_6} F$ where (1) X is either L for languages or Ps for sets of vectors of non-negative integers; we add the subscript a when considering accepting systems; (2) F is the list of features used in the model (e.g., we consider $(ncoo, tar, mcre, \delta)$, $(active_1, a, b, c, d, e)$, and $(active_2, a, c)$); (3) the numbers n_4, n_5, n_6 represent the

bounds on the starting number of membranes, the maximal number of membranes in any computation, and the number of membrane labels, $*$ representing the absence of a bound (if all three numbers are $*$, then we simply omit them all); (4) the numbers n_1, n_2, n_3 have the same meaning, but for the objects inside the system; the middle parameter, n_2 or n_5, can be replaced by n_2'/n_2 or n_5'/n_5 where the primed numbers indicate the bounds on the number of objects or membranes ever present in the system during halting computations only, thus refining this complexity measure.

3. Membrane Creation

Figure 1 describes the membrane structure used in the first theorems.
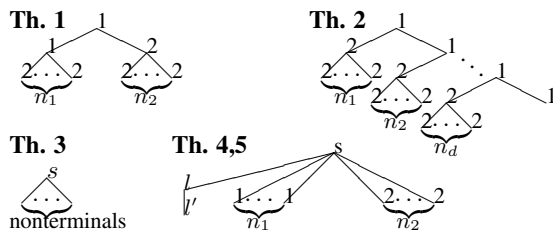
Figure 1. Membrane structures for membrane creation proofs

Generating   The first theorem shows how recursively enumerable languages can be generated by P systems with a small number of objects inside the system and a small number of membrane labels.

Theorem 1 $LO_{1,2,*}P_{1,*,2}(ncoo, tar, mcre, \delta) = RE.$

Proof. Due to Proposition 1, we construct a P system simulating a register machine $M = (2, T, P, l_0, l_h)$; P_- denotes the set of all SUB instruction labels.

$$
\begin{aligned}
\Pi &= \left(O, H, [_1 \ ]_1, w_1, R_1, R_2\right), \\
O &= T \cup \{a_1, a_2, C_1, C_2, g_0, g_1, g_2, t\} \\
&\quad \cup\ P \cup \{l_1, l_2, l_3, l_4, l_5, l_6, l_7 \mid l \in P_-\}, \\
H &= \{1, 2\}, \\
w_1 &= g_0, \\
R_1 &= R_{1,I} \cup R_{1,A} \cup R_{1,S} \cup R_{1,D} \cup R_{1,Z} \cup R_{1,O}, \\
R_2 &= R_{2,I} \cup R_{2,A} \cup R_{2,S} \cup R_{2,D} \cup R_{2,Z}.
\end{aligned}
$$

For clarity, the rules are grouped in categories (initialization, add, subtract, decrement case, zero case, output).

Initialization:

$$
\begin{aligned}
R_{1,I} &= \{g_0 \to [_2\, g_0\,]_2,\ g_1 \to [_1\, g_2\,]_1,\ g_2 \to (l_0)_{out}\}, \\
R_{2,I} &= \{g_0 \to (g_1)_{out}\},
\end{aligned}
$$

Output:

$$
\begin{aligned}
R_{1,O} &= \{l \to l' a_{out} \mid a \in T, \\
&\qquad l : (write(a), l') \in P\},
\end{aligned}
$$

Add:

$$
\begin{aligned}
R_{1,A} &= \{l \to l'\, (C_2)_{in_i},\ l \to l''\, (C_2)_{in_i} \\
&\qquad \mid l : (A(i), l', l'') \in P,\ i \in \{1, 2\}\} \\
&\quad \cup\ \{C_2 \to [_2\, t\,]_2,\ t \to \lambda\} \\
R_{2,A} &= \{C_2 \to [_2\, t\,]_2,\ t \to \lambda\},
\end{aligned}
$$

Subtract:

$$
\begin{aligned}
R_{1,S} &= \{l \to (l_1 C_1)_{in_i} \\
&\qquad \mid l : (S(i), l', l'') \in P,\ i \in \{1, 2\}\}, \\
&\quad \cup\ \{C_1 \to [_1\, t\,]_1\} \\
&\quad \cup\ \{l_1 \to (l_2)_{in_2}, \mid l : (S(1), l', l'') \in P\}, \\
R_{2,S} &= \{C_1 \to [_1\, t\,]_1\} \\
&\quad \cup\ \{l_1 \to (l_2)_{in_2} \mid l : (S(2), l', l'') \in P\},
\end{aligned}
$$

Decrement case:

$$
\begin{aligned}
R_{1,D} &= \{l_4 \to l_5 \delta \mid l \in P_-\} \\
&\quad \cup\ \{l_3 \to (l_4)_{in_1},\ l_5 \to (l')_{out} \\
&\qquad \mid l : (S(1), l', l'' \in P)\}, \\
R_{2,D} &= \{l_2 \to l_3 \delta \mid l \in P_-\} \\
&\quad \cup\ \{l_3 \to (l_4)_{in_1},\ l_5 \to (l')_{out} \\
&\qquad \mid l : (S(2), l', l'' \in P)\},
\end{aligned}
$$

Zero case:

$$
\begin{aligned}
R_{1,Z} &= \{l_6 \to l_7 \delta \mid l \in P_-\} \\
&\quad \cup\ \{l_1 \to (l_6)_{in_1},\ l_7 \to (l'')_{out} \\
&\qquad \mid l : (S(1), l',\ l'' \in P)\}, \\
R_{2,Z} &= \{l_1 \to (l_6)_{in_1}, l_7 \to (l'')_{out} \\
&\qquad \mid l : (S(2), l', l'' \in P)\}.
\end{aligned}
$$

Initially, by means of the auxiliary objects g_i, we create two membranes inside the skin region, labelled by 1 and 2, respectively. These membranes will be referred to as *cluster-membranes* (because they will contain inside them a number of elementary membranes). We finish the initialization phase by generating an object l_0 in the skin region.

The values of the two registers i, $i \in \{1, 2\}$, are represented by the number of elementary membranes labelled by 2 that occur inside the corresponding cluster-membrane i. The duty of the object C_i is to create membrane i. Object t is not needed for the computation, it is only used to keep the usual form of membrane creation rules and immediately erased after having been created.

Writing an output symbol $a \in T$ is done by a non-cooperative rule changing the instruction label and producing a symbol a that is immediately sent out. To increment a register, a membrane labelled by 2 is created inside the corresponding cluster-membrane.

In order to simulate a subtraction on register i we send the objects l_1 and C_1 into the cluster-membrane i and then proceed in the following way: while creating a membrane with label 1, object l_1 tries to enter in some membrane with label 2 as l_2. If such a membrane exists (i.e., register i is not empty), then l_2 changes to l_3 and dissolves the membrane, thus being spilled back into the cluster-membrane.

Before proceeding to the next label, we have to get rid of the auxiliary membrane 1 that was created inside the cluster-membrane by C_1. To this aim, l_3 enters into membrane 1 as l_4 and dissolves it, thereby changing to l_5. Finally, l_5 sends an object l' out to the skin region. As an overall result, l has been replaced by l' and the number of membranes with label 2 inside the cluster-membrane i has been reduced by 1. If, on the other hand, no membrane with label 2 exists in the cluster membrane, then l_1 waits for one step and then enters the newly created membrane 1 as l_6. Immediately afterwards, it changes to l_7 and dissolves the membrane. Finally, l_7 sends out an object l'' into the skin region. As an overall result, in the absence of membranes with label 2 inside the cluster-membrane i, l has been replaced by l''.

Notice that inside the system there can never be more than one copy of the same object. In fact, the number of objects inside the system never exceeds two (it can only be two after the first step of an ADD or a SUB instruction). □

Accepting Notice that the simulation of the register machine instructions in Theorem 1 is deterministic (the non-determinism arises from the non-determinism of the register machine program itself, not from the simulation). For the case of accepting sets of vectors in a deterministic way, we also specify the cardinality of the input alphabet, i.e., the number of components in the (Parikh) vectors.

Theorem 2 $DPs_a(m)OP_{1,*,2}(ncoo, tar, mcre, \delta) = PsRE(m).$

Proof. Given a recursively enumerable set of vectors of non-negative integers, we now simulate a register machine $M = (m + 2, P, l_0, l_h)$ (see Proposition 1); the input vector is represented in the skin by the corresponding numbers of symbols (a_i, i), $1 \leq i \leq m$, for the i-th component.

$$
\begin{aligned}
\Pi &= \big(O, H, [_1 \ ]_1, w_1, R_1, R_2\big), \\
O &= \{(a_i, j) \mid 1 \leq i \leq m + 2, \ 1 \leq j \leq i\} \\
&\cup \ \{C_1, C_2, t\} \cup \{g_i \mid 0 \leq i \leq 4(m + 2)\} \\
&\cup \ P \cup \{l_1, l_2, l_3, l_4, l_5, l_6, l_7 \mid l \in P_-\}, \\
&\cup \ \{(l, k, j) \mid l \in P_-, \\
&\quad 0 \leq k \leq 2, \ 1 \leq j \leq m + 2\}, \\
H &= \{1, 2\}, \\
w_1 &= g_0, \\
R_1 &= R_{1,I} \cup R_{1,A} \cup R_{1,S} \cup R_{1,D} \cup R_{1,Z}, \\
R_2 &= R_{2,I} \cup R_{2,A} \cup R_{2,S} \cup R_{2,D} \cup R_{2,Z}.
\end{aligned}
$$

Like in the previous theorem, we list the rules by categories (initialization, add, subtract, decrement case, zero case). We again start with the object g_0 which now starts the creation of the membrane structure $\left([_1 \ [_2 \ ]_2\right)^{m+2} \left(\ ]_1\right)^{m+2}$, where each membrane with label 2 (cluster membrane) corresponds to a register. In par-

allel, the input objects enter the corresponding membranes and become membranes with label 2, see Figure 1.

Initialization:
$$
\begin{aligned}
R_{1,I} &= \{g_{3i} \to [_2 \ g_{3i+1} \ ]_2, \ g_{3i+2} \to [_1 \ g_{3i+3} \ ]_1, \\
&\quad g_{3(m+2)+i} \to \big(g_{3(m+2)+i+1}\big)_{out} \\
&\quad \mid 0 \leq i \leq m + 1\} \\
&\cup \ \{g_{4(m+2)} \to l_0\} \\
&\cup \ \{(a_i, j) \to (a_i, j - 1)_{in_1} \mid 1 \leq i \leq m, \\
&\quad 2 \leq j \leq i\} \\
&\cup \ \{(a_i, 1) \to (C_2)_{in_2} \mid 1 \leq i \leq m\}, \\
R_{2,I} &= \{g_{3i+1} \to (g_{3i+2})_{out} \mid 0 \leq i \leq m + 1\} \\
&\cup \ \{C_2 \to [_2 \ t \ ]_2, \ t \to \lambda\}.
\end{aligned}
$$

Then we perform a deterministic simulation of the instructions in the program of the deterministic accepting register machine M. We now list the rules for ADD instructions (registers $m + 1$, $m + 2$ for the simulation of the working registers; observe that for the input initialization of the registers i, $1 \leq i \leq m$, similar rules for the corresponding symbols (a_i, i) are used):

Add:
$$
\begin{aligned}
R_{1,A} &= \{l \to l' \, (a_i, i), \ l \to l'' \, (a_i, i), \\
&\quad (a_i, j) \to (a_i, j - 1)_{in_1}, \\
&\quad (a_i, 1) \to (C_2)_{in_2} \mid l : (A(i), l', l'') \in P, \\
&\quad m + 1 \leq i \leq m + 2, \ 2 \leq j \leq i\}, \\
R_{2,A} &= \{C_2 \to [_2 \ t \ ]_2, \ t \to \lambda\}.
\end{aligned}
$$

The main reason for additional rules is that we have to go to that level of the membrane structure which corresponds to the register affected by the instruction. The same has to be done for the SUB instructions:

Subtract:
$$
\begin{aligned}
R_{1,S} &= \{l \to (l, 0, i), \ (l, 0, 1) \to (l_1 C_1)_{in_2} \\
&\quad \mid l : (S(i), l', l'') \in P\} \\
&\cup \ \{(l, 0, j) \to (l, 0, j - 1)_{in_1} \mid 2 \leq j \leq i, \\
&\quad l : (S(i), l', l'') \in P\} \\
&\cup \ \{(l, k, j) \to (l, k, j - 1)_{out} \mid 2 \leq j \leq i, \\
&\quad 1 \leq k \leq 2, \ l : (S(i), l', l'') \in P\} \\
&\cup \ \{(l, 1, 1) \to l', \ (l, 2, 1) \to l'' \\
&\quad \mid l : (S(i), l', l'') \in P\} \\
&\cup \ \{t \to \lambda\}, \\
R_{2,S} &= \{l_1 \to (l_2)_{in_2}, \ C_1 \to [_1 \ t \ ]_1\}.
\end{aligned}
$$

The decrement case and the zero case are handled very much like it is done in Theorem 1 for register 2.

Decrement case:
$$
\begin{aligned}
R_{1,D} &= \{l_4 \to l_5 \delta \mid l \in P_-\} \\
R_{2,D} &= \{l_2 \to l_3 \delta \mid l \in P_-\} \\
&\cup \ \{l_3 \to (l_4)_{in_1}, \ l_5 \to (l, 1, i)_{out} \\
&\quad \mid l : (S(i), l', l'' \in P)\},
\end{aligned}
$$

Zero case:
$$
\begin{aligned}
R_{1,Z} &= \{l_6 \to l_7 \delta \mid l \in P_-\}, \\
R_{2,Z} &= \{l_1 \to (l_6)_{in_1}, \ l_7 \to (l, 2, i)_{out} \\
&\quad \mid l : (S(i), l', l'' \in P)\}.
\end{aligned}
$$

After the correct simulation of the decrement case or the zero case, the symbol $(l, 1, i)$ or the symbol $(l, 2, i)$, respectively, is released; by decrementing the third component, these symbols can travel along the line of membranes labelled by 1 back to the skin membrane, where the rule $(l, 1, 1) \rightarrow l'$ or the rule $(l, 2, 1) \rightarrow l''$ is applied; hence, after decrementing, the label l' is obtained, whereas in the zero case we continue with label l''.

We finally observe that all the derivations in Π are performed deterministically, which completes the proof. $\square$

Generating with one object   Considering P systems only having one object inside the system during the whole computation, we realize that such P systems with one object work in a sequential way, hence, the following holds:

Theorem 3 *$PsMAT^\lambda$ is characterized by P systems*

$$\Pi = (O, H, \mu, w_1, \cdots, w_m, R_1, \cdots, R_n)$$

where (a) the initial membrane structure is limited by two levels (any membrane inside the skin is elementary) and the label 1 of the skin membrane is unique (i.e., the labels of the other membranes inside are different from 1), (b) exactly one of the multisets w_i consist of exactly one object, whereas all the other initial multisets are empty, (c) the rules in R_1 are of the forms $a \rightarrow bu_{out}$, $a \rightarrow b_{in_i}u_{out}$, $a \rightarrow b$, or $a \rightarrow [_i b]_i$ with $a, b \in O$, $u \in O^*$, and $i \in H'$ where $H' = H - \{s\}$, (d) the rules in R_i, $2 \le i \le n$, are of the forms $a \rightarrow b$, $a \rightarrow b_{out}$, or $a \rightarrow b\delta$.

Proof. Let Π be a P system obeying to the conditions given above. We first observe that due to the form of the initial configuration as well as due to the restricted forms of the rules, during any computation (i) the membrane structure is limited by two levels (any membrane inside the skin is elementary), (ii) the number of symbols inside the system is exactly one. Therefore, when constructing a matrix grammar $G = (N, T, S, M)$ simulating Π, with $T \subseteq O$, a membrane i can be represented by the nonterminal i, while the object a and its position in membrane i can be stored as a pair (a, i). With $InCon$ denoting the set of representations of the initial configuration given by $\mu, w_1, \cdots, w_m$ (in fact only one such representation is necessary), we now can specify the matrix grammar G as follows:

$$
\begin{aligned}
N \;=\;& H \cup O \times H \cup \{\overline{(a, i)} \mid (a, i) \in F\}, \\
M \;=\;& \{(S \rightarrow w) \mid w \in InCon\} \\
\cup\;& \{((a, s) \rightarrow (b, s)u) \mid a \rightarrow bu_{out} \in R_s\} \\
\cup\;& \{((a, s) \rightarrow (b, i)u, i \rightarrow i) \mid \\
& a \rightarrow b_{in_i}u_{out} \in R_s, i \in H'\} \\
\cup\;& \{((a, i) \rightarrow (b, i)) \mid a \rightarrow b \in R_i, i \in H\} \\
\cup\;& \{((a, i) \rightarrow (b, s)) \mid a \rightarrow b_{out} \in R_i, i \in H'\} \\
\cup\;& \{((a, s) \rightarrow (b, i)i) \mid a \rightarrow [_i b]_i \in R_s, i \in H'\}
\end{aligned}
$$

$$
\begin{aligned}
\cup\;& \{((a, i) \rightarrow (b, s), i \rightarrow \lambda) \\
& \mid a \rightarrow b\delta \in R_i, i \in H'\} \\
\cup\;& \{((a, i) \rightarrow \overline{(a, i)}), (\overline{(a, i)} \rightarrow \lambda) \mid (a, i) \in F\} \\
\cup\;& \{(a, i) \rightarrow \overline{(a, i)}, j \rightarrow \lambda) \mid (a, i) \in F, \\
& i \in H', j \in H\} \\
\cup\;& \{\overline{(a, s)} \rightarrow \overline{(a, s)}, j \rightarrow \lambda) \mid (a, s) \in F, \\
& j \in H - H_a\}.
\end{aligned}
$$

The set $F \subset O \times H$ is defined in such a way that a pair (a, i) is in F if and only if no rules of region i are applicable to a, eventually except for rules of the form $a \rightarrow b_{in_j}u_{out}$ in R_1, and for any $a \in O$, $H_a \subset H'$ denotes the set of membrane labels j such that R_s contains rules of the form $a \rightarrow b_{in_j}u_{out}$, $a, b \in O$, $u \in O^*$. We now can see that a configuration with object a in region i is a halting one if and only if $(a, i) \in F$ and (i) either $i \ne s$ or (ii) $i = s$ and no membranes with labels from H_a are present. The rules with the barred symbols $\overline{(a, i)}$ allow us to remove all non-terminal symbols from the sentential form of M when Π has reached a halting configuration. Hence, $Ps(L(M))$ equals the set of vectors generated by Π.

For showing the converse inclusion, we consider a matrix grammar $G = (N, T, S, M)$ without appearance checking in the f-binary normal form and construct a P system Π simulating G: Π starts with the initial configuration $[_s S]_s$.

$$
\begin{aligned}
\Pi \;=\;& (O, H, [_s\;]_s, S, R_{A_{init}}, \cdots, R_s), \\
O \;=\;& \{S, S_1, f, \#\} \cup N_1 \\
\cup\;& \{l_i \mid l \in M, 0 \le i \le 4\}, \\
H \;=\;& N_2 \cup \{s\}, \\
R_s \;=\;& \{S \rightarrow [_{A_{init}} S_1]_{A_{init}} \\
& \mid (S \rightarrow X_{init}A_{init}) \in M\} \\
\cup\;& \{X \rightarrow (l_0)_{in_A} \mid l : (X \rightarrow Y, A \rightarrow uv) \in M\} \\
\cup\;& \{l_1 \rightarrow [_u l_2]_u \\
& \mid l : (X \rightarrow Y, A \rightarrow uv) \in M, u \in N_2\} \\
\cup\;& \{l_1 \rightarrow l_3 u_{out} \\
& \mid l : (X \rightarrow Y, A \rightarrow uv) \in M, u \in T \cup \{\lambda\}\} \\
\cup\;& \{l_3 \rightarrow [_v l_4]_v \\
& \mid l : (X \rightarrow Y, A \rightarrow uv) \in M, v \in N_2\}, \\
\cup\;& \{l_3 \rightarrow Y v_{out} \\
& \mid l : (X \rightarrow Y, A \rightarrow uv) \in M, v \in T \cup \{\lambda\}\} \\
\cup\;& \{X \rightarrow \#, \mid X \in N_1 \cup \{\#\} \\
\cup\;& \{f \rightarrow \#_{in_A} \mid A \in N_2\}, \\
R_A \;=\;& \{S_1 \rightarrow (X_{init})_{out} \mid (S \rightarrow X_{init}A) \in M\} \\
\cup\;& \{l_0 \rightarrow l_1\delta \mid l : (X \rightarrow Y, A \rightarrow uv) \in M\} \\
\cup\;& \{l_2 \rightarrow (l_3)_{out} \mid l : (X \rightarrow Y, B \rightarrow Av) \in M\} \\
\cup\;& \{l_4 \rightarrow (Y)_{out} \mid l : (X \rightarrow Y, B \rightarrow uA) \in M\} \\
\cup\;& \{\# \rightarrow \#\}.
\end{aligned}
$$

The rules $S \rightarrow [_{A_{init}} S_1]_{A_{init}} \in R_s$ as well as $S_1 \rightarrow (X_{init})_{out} \in R_{A_{init}}$ simulate the start matrix by producing a membrane corresponding to the literal symbol A_{init} and an object corresponding to the control symbol X_{init}. A matrix $l : (X \rightarrow Y, A \rightarrow uv)$ is simulated as follows: first, object X removes a membrane with label A by

the rules $X \to (l_0)_{in_A} \in R_s$ and $l_0 \to l_1 \delta \in R_A$; then, it creates membranes with labels u, v if u, v are non-terminal symbols, or sends u, v into the environment if u, v are terminal symbols. Finally, the object X changes to Y. The rules $X \to \#, \# \to \# \in R_s$ guarantee that if the derivation of G is "stuck" in a form which is not terminal, then the corresponding computation in Π will enter an infinite loop. Finally, the result of a derivation in G is terminal if and only if f is produced and no other non-terminal symbols have remained; this is checked by the rules $f \to \#_{in_A} \in R_s$ (after the application of such a rule, the rule $\# \to \# \in R_A$ then guarantees that the corresponding computation in Π will enter an infinite loop).

Observing that the P system constructed above fulfills all the conditions stated in the theorem concludes the proof. $\square$

We conjecture that we need not restrict the membrane structure, i.e., even the following should be true:

$$PsMAT^\lambda = PsO_{1,1,*}P_{*,*,*}(ncoo, tar, mcre, \delta).$$

Number of Symbols The cardinality of the alphabet in the computational completeness proofs usually depends on the complexity parameters of the simulated device. We will now show that any recursively enumerable set of m-dimensional vectors of non-negative integers can be generated by P systems with membrane creation and dissolution, having the alphabet of $10 + m$ symbols.

Theorem 4 $L(m)O_{2,2,10+m}P_{1,*,*}(ncoo, tar, mcre, \delta)$
$= RE(m)$.

Proof. We simulate a register machine $M = (2, T, P, l_0, l_h)$ where $T = \{a_i \mid 1 \le i \le m\}$; let P_+ denote the set of all ADD instruction labels, and let P_- denote the set of all SUB instruction labels.

$$
\begin{aligned}
\Pi &= (O, H, [_s [_I \;]_I \;]_s, \lambda, b, R_{l_0}, \cdots, R_D), \\
O &= T \cup \{b, c, d, e\} \cup \{r_+, r_-, r'_- \mid r \in \{1, 2\}\}, \\
H &= P \cup \{I, 1, 2, s, D\}, \\
R_i &= R_{i,I} \cup R_{i,O} \cup R_{i,A} \cup R_{i,S} \cup R_{i,D} \\
&\quad \cup \; R_{i,Z} \cup R_{i,N}, \; i \in H.
\end{aligned}
$$

For clarity, the rules are grouped (initialization, output, add, subtract, decrement case, zero case, next instruction).

Initialization:
$$
\begin{aligned}
R_{I,I} &= \{b \to [_{l_0} d \,]_{l_0}\}, \\
R_{i,I} &= \emptyset, \; i \in H - \{I\},
\end{aligned}
$$
Output:
$$
\begin{aligned}
R_{l,O} &= a \to a_{out} \mid l : (write(a), l'), \; a \in T\}, \\
&\quad \cup \; \{a \to a\delta \mid a \in T\}, \; l \in P \cup \{I\}, \\
R_{s,O} &= \{a \to a_{out}b, \; a \to a_{out}c \mid a \in T\}, \\
R_{i,O} &= \emptyset, \; i \in H - (P \cup \{I, s\}),
\end{aligned}
$$
Add:

$$
\begin{aligned}
R_{l,A} &= \{d \to (r_+)_{out} \mid l : (A(r), l', l'') \in P, \\
&\quad r \in \{1, 2\}\} \cup \{r_+ \to r_+\delta \mid r \in \{1, 2\}\}, \\
&\quad l \in P \cup \{I\}, \\
R_{s,A} &= \{r_+ \to [_r d \,]_r \mid r \in \{1, 2\}\}, \\
R_{r,A} &= \{d \to b_{out}, d \to c_{out}\}, \; r \in \{1, 2\}, \\
R_{D,A} &= \emptyset,
\end{aligned}
$$
Subtract:
$$
\begin{aligned}
R_{l,S} &= \{d \to (r_-)_{out} \mid l : (S(r), l', l'') \in P, \\
&\quad r \in \{1, 2\}\} \cup \{r_- \to r_-\delta \mid r \in \{1, 2\}\}, \\
&\quad l \in P \cup \{I\}, \\
R_{s,S} &= \{r_- \to r'_- d, \; d \to [_D d \,]_D \mid r \in \{1, 2\}\}, \\
R_{D,S} &= \{d \to \lambda\}, \\
R_{i,S} &= \emptyset, \; i \in H - P - \{I, s, D\},
\end{aligned}
$$
Decrement case:
$$
\begin{aligned}
R_{s,D} &= \{r'_- \to (r'_-)_{in_r}, \; e \to e_{in_D} \mid r \in \{1, 2\}\}, \\
R_{r,D} &= \{r_- \to r'_- d, \; d \to [_d d \,]_D\}, \; r \in \{1, 2\}, \\
R_{D,D} &= \{r'_- \to e_{out}\}, \; r \in \{1, 2\}, \\
R_{i,D} &= \emptyset, \; i \in H - \{1, 2, s, D\},
\end{aligned}
$$
Zero case:
$$
\begin{aligned}
R_{s,Z} &= \{r'_- \to (r'_-)_{in_D}\} \mid r \in \{1, 2\}\}, \\
R_{D,Z} &= \{r'_- \to b\delta \mid r \in \{1, 2\}\}, \\
R_{i,D} &= \emptyset, \; i \in H - \{s, D\},
\end{aligned}
$$
Next instruction:
$$
\begin{aligned}
R_{s,N} &= \{b \to b_{in_l}, \; c \to c_{in_l} \mid l \in P\}, \\
R_{l,N} &= \{b \to [_{l'} a \,]_{l'}, \; c \to [_{l''} a \,]_{l''} \mid \\
&\quad l : (X(r), l', l'') \in P, \; X \in \{A, S\}\}.
\end{aligned}
$$

The instruction labels are encoded into membrane labels, and the values of the registers are encoded by the number of copies of some membranes associated with them. The proof mainly relies on the fact that the amount of information needed to be transmitted between the instructions and the registers is "small", i.e., the instructions tell us which operation (ADD or SUB, represented by $r_+, r_-, r \in \{1, 2\}$) has to be applied and to which register r it has to be applied. The objects $r'_-, r \in \{1, 2\}$, and e are used to implement the SUB instruction, and the object d here is used to organize a delay for the appearance checking, similar to the technique from Theorem 1, whereas otherwise it is used when the membranes already contain all the information needed.

After an operation has been simulated, the next instruction is chosen from two variants, non-deterministically chosen in the ADD case and as well in the SUB case here depending on whether decrementing has been successful or not. These variants are represented by the objects b, c. The transition to the next instruction is done in the following way: Object b in membrane l creates membrane l', or object c in membrane l creates membrane l''. After this, the object "memorizes" the next register to be operated on and the operation to be performed, and then membrane l is dissolved, leaving the newly created membrane in the skin. $\square$

If we want to start with the simplest membrane structure, one more symbol is needed as is exhibited in the following:

Theorem 5 $L(m)O_{1,2,11+m}P_{1,*,*}(ncoo, tar, mcre, \delta)$
$= RE(m).$

Proof. We here again simulate a register machine $M = (2, T, P, l_0, l_h)$ as in the proof of the preceding theorem, but in the P system Π' we use an additional symbol a for an initial step starting in the skin membrane:

$$\Pi' = (O, H, [_s \ ]_s, a, R_{l_0}, \cdots, R_D),$$
$$O = T \cup \{a, b, c, d, e\} \cup \{r_+, r_-, r'_- \mid r \in \{1, 2\}\},$$
$$H = P \cup \{I, 1, 2, s, D\},$$
$$R_i = R_{i,I} \cup R_{i,O} \cup R_{i,A} \cup R_{i,S} \cup R_{i,D}$$
$$\cup \ R_{i,Z} \cup R_{i,N}, \ i \in H.$$

Initialization:
$$R_{s,I} = \{a \to [_I \ b \ ]_I\},$$
$$R_{I,I} = \{b \to [_{l_0} \ d \ ]_{l_0}\},$$
$$R_{i,I} = \emptyset, \ i \in H - \{s, I\}.$$

Except for the initialization, the sets of rules are exactly the same as for the P system Π constructed in the preceding proof, which observation already completes this proof. $\square$

Figure 2 depicts the membrane structures of the P systems constructed in the succeeding proofs.
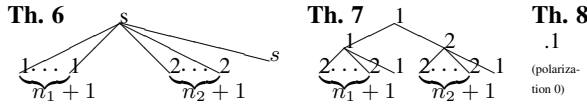
Figure 2. Membrane structures for active membrane proofs

4. One Polarization

The theorem below provides a result, similar to that of Theorem 1, for P systems with active membranes with only one polarization (usually called P systems with active membranes without polarizations). The construction gives no upper bound on the number of objects present inside the system in general, but during any halting computation the number of objects never exceeds 3.

Theorem 6 $LO_{1,3/*,*}P_{4,*,3}(active_1, a, b, c, d, e) = RE.$

Proof. In the description of the P system Π below, w_s describes the initial multiset for the skin membrane, whereas w'_s denotes the initial multiset in the elementary membrane having the same label as the skin membrane. We now simulate a register machine $M = (2, T, P, l_0, l_h)$:

$$\Pi = (O, \mu, w_s, w_1, w_2, w'_s, R),$$
$$\mu = [_s [_1 \ ]_1 [_2 \ ]_2 [_s \ ]_s \ ]_s,$$
$$O = T \cup \{a_i \mid 1 \leq i \leq 2\} \cup \{l, l_1, l_2 \mid l \in P\}$$
$$\cup \ \{b_1, b_2, t, d, \#\},$$
$$w_s = l_0, \ w_1 = w_2 = w'_s = \lambda,$$
$$R = R_O \cup R_A \cup R_S \cup R_D \cup R_Z.$$

The rules are grouped in categories: output, add, subtract, decrement case and zero case.

Output:
$$R_O = \{[\ l \to l'a\ ]_s, \ [\ l \to l''a\ ]_s, [\ a\ ]_s \to [\ ]_s a$$
$$\mid l : (write(a), l'), \ a \in T\},$$

Add:
$$R_A = \{l[\ ]_i \to [\ l\ ]_i, \ [\ l\ ]_i \to [\ l_1\ ]_i [\ t\ ]_i,$$
$$[\ t \to \lambda\ ]_i, \ [\ l_1\ ]_i \to [\ ]_i l',$$
$$[\ l_1\ ]_i \to [\ ]_i l''\mid$$
$$l : (A(i), l', l''), \ i \in \{1, 2\}\},$$

Subtract:
$$R_S = \{[\ l \to db_i l_1\ ]_s, \ b_i [\ ]_i \to [\ b_i\ ]_i,$$
$$[\ b_i\ ]_i \to [\ ]_i t, \ [\ t \to \lambda\ ]_s,$$
$$[\ b_i \to \#\ ]_s, \ [\ b_i \to \#\ ]_i,$$
$$[\ \# \to \#\ ]_i \mid l : (S(i), l', l''), \ i \in \{1, 2\}\}$$
$$\cup \ \{d[\ ]_s \to [\ t\ ]_s, \ [\ t \to \lambda\ ]_s$$
$$[\ d \to \#\ ]_s, \ [\ \# \to \#\ ]_s\},$$

Decrement case:
$$R_D = \{l_1[\ ]_i \to [\ l_1\ ]_i, \ [\ l_1\ ]_i \to l'$$
$$\mid l : (S(i), l', l''), \ i \in \{1, 2\}\},$$

Zero case:
$$R_Z = \{l_1[\ ]_s \to [\ l_2\ ]_s, \ [\ l_2\ ]_s \to [\ ]_s l''$$
$$\mid l \in P_-\}.$$

Like in the previous theorem, we simulate a register machine with output tape and two registers; the values of registers i, $i \in \{1, 2\}$, are represented by the multiplicities of membranes i. However, since new membranes can only be created by dividing existing ones, one extra membrane is needed for every register. The duty of d is to "keep busy" the elementary membrane with label s (otherwise $\#$ appears and the computation does not halt), and the use of the objects b_i is to "keep busy" one membrane with label i for two steps. Object t is not needed for the computation, it is only used to keep the usual form of membrane division and communication rules, it is immediately erased.

Generating an output is done by a non-cooperative rule changing the instruction label and producing the corresponding symbol, which is then sent out. Incrementing a register $(l : (A(i), l', l''))$ is done in the following way: l enters membrane i (there is always at least one), dividing it. The object l_1 in one copy is sent to the skin as l' or l'', while the object t in the other copy is erased.

Subtracting with $(l : (S(i), l', l''))$ is done by keeping busy the elementary membrane with label s for one step and one membrane with label i for two steps, while object l_1 tries to enter any membrane with label i. If the register is not zero, then l_1 enters one of the other membranes with label i, dissolves it, and changes to l'. Otherwise, after having waited for one step, object l_1 enters the elementary membrane with label s and returns to the skin as l''.

During a correct simulation of a run of the register machine (in particular, during any halting computation) there are never more than 3 objects present inside the system. $\square$

It is possible to reduce the number of membrane labels to two at the price of starting with seven membranes.

Theorem 7 $LO_{1,3/*,*}P_{7,*,2}(active_1, a, b, c, d, e) = RE$.

Proof. (sketch) In a way quite similar to the proof of Theorem 1, let us now start with a membrane structure $[_1 [_1 [_1]_1]_2]_2]_1[_2 [_1]_1]_2]_2]_2]_1$ (see also Figure 2), represent the values of a working register i by the number of elementary membranes with label 2, inside the membranes with labels i, minus one. The elementary membranes with label 1 will be used for delay, just like the elementary membrane labelled by s was used in the proof of Theorem 6, and the instructions are simulated accordingly. The main difference is that when simulating an ADD or a SUB instruction we have one additional initial step at the beginning choosing the "cluster membrane" representing the corresponding register (see the proof of Theorem 1). Obviously, at the end of the simulation of the instruction in the right "cluster membrane", we need an additional final step for moving the instruction label back to the skin membrane. $\square$

5. Two Polarizations

The last theorem established in this paper shows that with two polarizations we need only one membrane to simulate register machines in a deterministic way:

Theorem 8 $DPs_aOP_{1,1,1}(active_2, a, c) = PsRE$.

Proof. We will simulate the actions of a deterministic register machine $M = (d, P, l_0, l_h)$ with d registers by a deterministic P system with one membrane and two polarizations. For every instruction l, let us denote the register l acts on by $r(l)$ and the operation l carries out by $op(l)$.

$$
\begin{aligned}
\Pi &= (O, \{0,1\}, [_1 \ ]_1^0, (l_0, 0, 0), R), \\
O &= \{(a, i, j) \mid 1 \le i \le d, 0 \le j \le d+2\} \\
&\cup \{(l, i, j) \mid l \in P, 0 \le i \le 2, \\
&\quad 1 \le j \le d+2\} \cup \{\#\}, .
\end{aligned}
$$

The system receives the input $(a, 1, 0)^{n_1} \cdots (a, d, 0)^{n_d}$ in addition to w_1 in the skin membrane. The set R contains the following rules

$$[\ z \ ]_1^e \to [\ ]_1^{1-e} z, \ e \in \{0, 1\}, \tag{1}$$

$$[\ (a, i, j) \to (a, i, j+1) \ ]_1^0, \ 1 \le i \le d,$$
$$0 \le j \le d+1, \tag{2}$$

$$[\ (a, i, d+2) \to (a, i, 0) \ ]_1^0, \ 1 \le i \le d, \tag{3}$$

$$[\ (a, i, j+1) \ ]_1^1 \to [\ ]_1^0 (a, i, j+1), $$
$$1 \le i \le d, \tag{4}$$

$$[\ (l, 0, j) \to (l, 0, j+1) \ ]_1^0, \ l \in P,$$

$$0 \le j < r(l) - 1, \tag{5}$$

$$[\ (l, i, j) \to (l, i, j+1) \ ]_1^0, \ l \in P, \ i \in \{1, 2\}, $$

$$r(l) \le j \le d+1, \tag{6}$$

$$[\ (l, 1, d+2) \to (l', 0, 0) \ ]_1^0, \ l \in P, \tag{7}$$

$$[\ (l, 2, d+2) \to (l'', 0, 0) \ ]_1^0, \ l \in P, \tag{8}$$

$$[\ (l, 0, j) \to (l, 1, j+1)(a, j, j+1) \ ]_1^0, \ l \in P,$$
$$j = r(l) - 1, \ op(l) = A, \tag{9}$$

$$[\ (l, 0, j) \to (l, 0, j+1) z \ ]_1^0, \ l \in P,$$
$$j = r(l) - 1, \ op(l) = S, \tag{10}$$

$$[\ (l, 0, j) \to (l, 0, j+1) \ ]_1^0, \ l \in P,$$
$$j = r(l), \ op(l) = S, \tag{11}$$

$$[\ (l, 0, j) \to (l, 0, j+1) \ ]_1^1, \ l \in P,$$
$$j = r(l) + 1, \ op(l) = S, \tag{12}$$

$$[\ (l, 0, j) \to (l, 1, j+1) \ ]_1^0, \ l \in P,$$
$$j = r(l) + 2, \ op(l) = S, \tag{13}$$

$$[\ (l, 0, j) \to (l, 2, j) z \ ]_1^1, \ l \in P,$$
$$j = r(l) + 2, \ op(l) = S, \tag{14}$$

$$[\ (l, 2, j) \to (l, 2, j+1) \ ]_1^1, \ l \in P,$$
$$j = r(l), \ op(l) = S, \tag{15}$$

$$[\ (l_h, 0, 0) \ ]_1^0 \to [\ ]_1^1 (l_h, 0, 0), \ 1 \le i \le d. \tag{16}$$

The idea of this proof is similar to the one from [2, 3]: the symbols corresponding to the registers have states (second subscript) $0, \cdots, d+2$, and so do the symbols corresponding to the instructions of the register machine. The first subscript of the instruction symbols is 0 if the instruction has not yet been applied, and it is 1 if increment or decrement has been applied, and 2 if the decrement has failed.

Most of the time the polarization is 0; object z can reverse the polarization by (1). When the polarization is 0, the register symbols cycle through the states by (2), (3). Before the current instruction l is applied, the instruction symbols also cycle through the states until the state becomes $r(l)-1$, i.e., the index of the register (the instruction operates on) minus one. We will explain the details of the application below. After the instruction has been applied, the first subscript of the instruction symbol changes to 1 or 2 and it cycles through the states by (6), finally changing into l' by (7) or into l'' by (8).

Addition is done by rule (9). Decrement is done by a "diagonalization technique": polarization 1 when register i is in state $i+1$ signals a decrement attempt of register i by (4), and the polarization will change if and only if it has been successful. Thus, to apply $l : (S(i), l', l'')$, the instruction symbol in state $i-1$ additionally produces a symbol z. By the time z changes the polarization to 1 by (1), all other symbols reach state $i+1$. After one more step the state symbol checks whether the decrement has been

successful, (13), or not, (14). After a successful decrement all symbols continue changing states with polarization 0 and state $i + 1$. Otherwise, the instruction symbol additionally produces a symbol z and after one more step all symbols continue changing states with polarization 0 and state $i + 1$.

After the simulation of M has reached the final label, the instruction symbol exits the system, changing the polarization to 1. Since the register symbols are in state 1, the system halts. $\square$

Looking into the proof of the preceding theorem we realize that even a more general result is shown: the multisets remaining in the skin membrane at the end of a halting computation can be interpreted as the computation result:

Corollary 1 *Any partially recursive function can be computed by a deterministic P system with one membrane, two polarizations and internal output.*

6. Conclusions and Open Problems

We have shown that P systems with membrane creation generate RE, using two membrane labels and at most two objects present inside the system throughout the computation. Accepting any recursively enumerable language can also be done with two membrane labels. On the other hand, it is possible to bound the number of symbols by $m + 10$ and still generate $RE(m)$, provided that the number of membrane labels is unbounded.

We also have shown that RE is generated by P systems using four membranes and three labels or seven membranes and two labels in the initial configuration, where at most three objects are ever present in any halting computation.

It is known from [2, 3] that P systems with two polarizations and rules of types (a) and (c) generate $PsRE$ using two membranes, or accept $PsRE$ using one membrane. In this article, we have proved that *deterministic* systems of this kind with one membrane accept $PsRE$. Moreover, in the proof of this result (Theorem 8), the rules are global (there is only one membrane) and rules of type (c) are non-renaming (the contents of the environment does not matter).

Improving any complexity parameter greater than one (especially in the case of $*$) in any theorem is an open question. Moreover, the following questions are of interest:

- How can the types of rules be restricted in Theorem 6?

- How can target indications be restricted in Theorem 1?

- What further restrictions cause a complexity trade-off?

- What is the generative power of P systems without polarizations and m membranes, $m = 1, 2, 3$?

- What is the generative power of one-membrane P systems with two polarizations and external output?

References

[1] A. Alhazov. P systems without multiplicities of symbol-objects. *Information Processing Letters*, submitted, 2004 (preprint version: www.geocities.com/aartiom).

[2] A. Alhazov, R. Freund, Gh. Păun. Computational Completeness of P Systems with Active Membranes and Two Polarizations. *Machines, Computations, and Universality*, International Conference, MCU 2004, Saint Petersburg, 2004, Revised Selected Papers (M. Margenstern, Ed.), Lecture Notes in Computer Science **3354**, Springer-Verlag, 2005, 82–92.

[3] A. Alhazov, R. Freund, Gh. Păun. P Systems with Active Membranes and Two Polarizations. *Second Brainstorming Week on Membrane Computing*, Sevilla, 2004, (Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, Eds.), RGNC Tech. Rep **01/2004**, University of Seville, 2004, 20–36.

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

[5] M. Minsky. *Computation. Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.

[6] Gh. Păun. Computing with membranes, *Journal of Computer and System Sciences,* 61 (1): 108–143, 2000 and TUCS Research Report 208, 1998.

[7] Gh. Păun. *Membrane Computing. An Introduction*. Springer-Verlag, Berlin (Natural Computing Series), 2002.

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

[9] P systems page, psystems.disco.unimib.it.