Membrane Systems with Carriers

Carlos MARTÍN-VIDE
Research Group in Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: cmv@astor.urv.es

Gheorghe PĂUN[1]
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
E-mail: gpaun@imar.ro

Grzegorz ROZENBERG
Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
and
Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309, U.S.A.
E-mail: rozenber@liacs.nl

Abstract. A membrane system is a model of computation which is inspired by some basic features of biological membranes. In this paper we consider another biologically inspired notion, viz., the notion of a carrier (or vehicle), as, e.g., used in gene cloning. We investigate the power of membrane systems where the rules for the evolving of objects are replaced by the rules that carry objects (by vehicles) through membranes. It turns out that these systems (even with a small number of membranes, a small number of carriers, and a small number of passengers taken by carriers) are computationally universal.

KEYWORDS: Natural computing, molecular computing, membrane computing, P systems, Turing computability

1 Introduction

A fundamental role of biological membranes (see, e.g., [1]) is to ensure that certain substances (molecules) stay within (do not escape from) the space enclosed by the membrane

[1]Research supported by the Direcció General de Recerca, Generalitat de Catalunya (PIV).

(e.g., a cell), while others, e.g., toxic molecules, stay out of this space. Moreover, membranes allow certain molecules to pass through: e.g., waste products to leave, and certain nutrients to enter. Also, membranes form a communication structure, allowing messages (signals) to be received or to be transmitted by the enclosed space. This communication is crucial for establishing multicellular communication and hence for establishing multicellular organization (see, e.g., [9]). This compartimentation by membranes, with each enclosed area having its own set of molecules and (enzymes enhancing) reactions, with the transport of molecules and (hence) the communication through membranes, is the paradigm underlying *membrane systems* (see, e.g., [10] and [14], as well as Chapter 3 of [3]).

It must be stressed that membrane systems are *not* intended to model the functioning of biological membranes. Rather, membrane systems abstract from a number of principles underlying the functioning of biological membranes, and use this abstraction to construct a novel model of computing. Such an approach is typical for the area of Natural Computing, where one studies all kinds of computing inspired by (or gleaned from) nature.

The *membrane structure* of a membrane system is a hierarchical arrangement of membranes (understood as three dimensional vesicles), embedded in a *skin* membrane, the one which separates the system from its *environment*. A membrane without any membrane inside is called *elementary*. Each membrane defines a *region*. For an elementary membrane 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 included in it. Figure 1 illustrates these notions. As the reader can see, we give labels (positive integers) to membranes in order to be able to address them. Since each region is delimited ("from outside") by a unique membrane, we will use the labels of membranes to also label the corresponding regions.

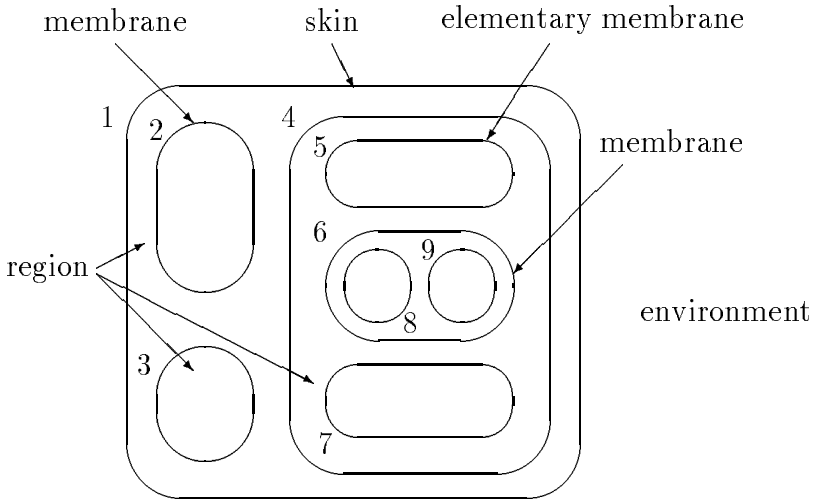

Figure 1: A membrane structure

Each region contains a multiset of *objects* and a set of (*evolution*) *rules*. The objects are represented by symbols from a given alphabet. Typically, an evolution rule is of the form $ca \rightarrow cb_{in_j} d_{out} d_{here}$, and it "says" that a copy of the object a, in the presence of a

copy of the *catalyst* c (this is an object which is never modified, it only assists the evolution of other objects), is replaced by a copy of the object b and two copies of the object d, where the copy of b has to "immediately" enter the membrane labeled by j (hence to enter region j), providing that it is adjacent to the region where the rule is applied, a copy of object d is sent out of the current membrane, and a copy of d remains in the same region.

In order not to complicate too much many formulations in the sequel of this paper we will use interchangeably the phrases "an object" and "a copy of an object" – the real meaning will always be clear from the context of the considerations.

A global clock is assumed (that is, the same clock for all regions of the system). In each time unit we pass from a *configuration* of the system to another one, by applying the rules in a nondeterministic and maximally parallel manner: the objects to evolve and the rules governing this evolution are chosen in a nondeterministic way; this choice is "exhaustive" in the sense that no rule can be applied anymore in this evolution step (there are not enough objects available anymore for any rule to be applied now – this is the maximality of application). More specifically, it is instructive to see a single step as a "macro-step" consisting of several "micro-steps" performed one after each other. Consider a region of the system. First, we assign objects to rules, nondeterministically chosing rules and objects, until no further assignment is possible (note that the multiplicity of objects present in each region is crucial in this micro-step). Then, all these objects are removed from the current multiset of the region. Then, all objects specified by the right hand sides of the chosen rules are added to this multiset, together with their "transfer commands": in_j, *out, here*. Then, all transfers indicated by commands in_j and *out* are executed (if a copy of an object is introduced in the skin region, i.e., the region delimited by the skin membrane, and its transfer command is *out*, then it will be sent out of the system, to the environment, and it never "comes back"), and copies of objects with the transfer command *here* remain in the given region. Then, the transfer commands are removed, and a "macro-step" is completed.

In this way, one gets *transitions* between the configurations of the system. A sequence of transitions is called a *computation*. A configuration is *halting*, if no rule is applicable in any region (nothing can happen anymore). A computation is *halting* if it reaches a halting configuration. We consider only halting computations, and the *result* of such a computation is the number of objects present in a prespecified region (called the *output region*).

Many modifications/extensions of this very basic model described above are discussed in the literature.

For instance, a priority relation among rules have been considered. This means that in each region a partial order relation on the set of rules in this region is given – then, a rule can be chosen (to process a multiset of objects) only if no rule of a higher priority is applicable.

Another "control device" for membrane systems considered in the literature is a modification of membrane permeability. Thus, the membranes can be *dissolved* (the objects of a dissolved membrane remain in the region surrounding it, while the rules are removed; the skin membrane cannot be dissolved), or made *impermeable* (no object can pass through such a membrane). We refer the reader to, e.g., [3], [10], [12], and [14] for these and some other modifications of membrane systems.

A typical way of investigating the influence of various features of membrane systems is to ask about their computational power: e.g., are membrane systems using these features computationally universal? (see, for example, [10], [6], [14], [16]). Several papers have also considered solving some NP-complete problems by membrane systems in polynomial (even linear) time – clearly, the price paid for this reduction of time is the exponential increase in the number of membranes or objects. This has led to a number of interesting computation techniques – see, for example, [13], [8], [4], [15].

In this paper we introduce the idea of carriers to be used by membrane systems. The origin of this idea is twofold. It abstracts the work of the carrier proteins assisting molecules to pass through membranes (see, e.g., [1]), and it also abstracts from the fundamental idea of vectors used in gene cloning (see, e.g., [2]). A vector in gene cloning is a vehicle that transports the needed gene into the host cell. Two very popular vehicles that occur in nature are plasmids and bacteriophages. Also, recently plasmids were used as data registers for the purpose of DNA computing, see, for example, [7]. Here, several so-called "recognition sites" are planted into a plasmid – they are used as bits representing the presence/absence of some elements of the data structures considered (for example, presence/absence of nodes in a graph). Eventually, these plasmids are transported into *E. Coli* cells for cloning.

In membrane systems with carriers the objects never evolve (there are no rules for evolving objects), but rather objects are carried back and forth, by carriers, through the membranes during the computation process.

Thus, in membrane systems with carriers we have objects of two types: the carriers ("vehicles") and the passengers. None of them is evolving; the passengers can pass through membranes only when carried by carriers. We also have objects, of both types, in the environment. Rules to handle objects (attaching and detaching carriers to/from passengers, and passing through membranes) are associated with regions, and also with the environment. Otherwise, the functioning of a membrane system with carriers is the same as of an ordinary membrane system: rules are applied in a nondeterministic maximally parallel manner, and transitions between configurations yield computations.

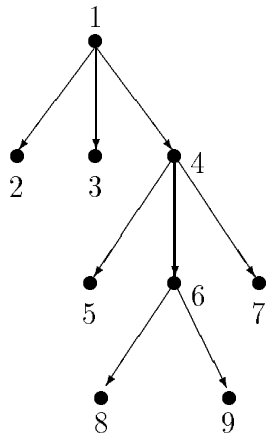

Figure 2: The tree describing the membrane structure from Figure 1

The basic question asked in this paper is the computational strength of membrane

systems with carriers. Somewhat surprisingly, it turns out that such systems are computationally universal (i.e., they have the power of Turing machines). This holds even for systems with a reduced number of membranes/carriers and for the systems with a limited (passenger) transporting power of carriers.

One can perceive sending (exchanging) objects through a membrane as a communication between the two regions defined by the membrane. Consequently, one can see this paper as investigating the power of communication in its purest form. In a membrane system with carriers no objects are created, and no objects are destroyed, and the whole computation process is accomplished by communication between regions (which is implemented by carriers). Thus, our result on the computational universality of membrane systems with carriers may be seen as a result on the power of communication in membrane systems.

2 Membrane Systems with Carriers

A membrane structure is pictorially represented by an Euler-Venn diagram (like the one in Figure 1); it can be mathematically represented by a tree or by a string of matching parentheses. The tree of the structure from Figure 1 is given in Figure 2. The same structure is also represented by the following parentheses expression:

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

Since the membranes are having labels, also here the pairs of corresponding parentheses have labels. It should be noted that the same membrane structure may be represented by different parenthetic expressions.

The multisets over a given finite support (alphabet) are represented by strings of symbols. The order of symbols clearly does not matter, the number of copies of an object in a multiset is given by the number of occurrences of the corresponding symbol in the string.

We are ready now to introduce membrane systems.

A *membrane system*, also called a *P system* (of degree $m \geq 1$), *with carriers* is a construct

$$\Pi = (O, V, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, E, i_o),$$

where:

1. O is the alphabet of *objects*;
2. V is the alphabet of *carriers* ("vehicles", hence the use of V);
3. μ is a membrane structure with m membranes (injectively labeled by positive integers $1, 2, \ldots, m$);
4. $w_1, \ldots, w_m$ are strings over $O \cup V$, representing the multisets of objects and carriers initially present in the regions of the system, where w_i is the multiset of objects present in the region delimited by the membrane (labelled by) i;

5. $R_1, \ldots, R_m$ are finite sets of *rules*, governing the work of carriers in the regions of the system; each rule has one of the following forms:

   - $va_1 \ldots a_k \to [va_1 \ldots a_k]$, for $v \in V, a_1, \ldots, a_k \in O, k \geq 1$ (*attaching rules*);
   - $[va_1 \ldots a_k] \to va_1 \ldots a_k$, for $v \in V, a_1, \ldots, a_k \in O, k \geq 1$ (*detaching rules*);
   - $[va_1 \ldots a_k] \to in$, for $v \in V, a_1, \ldots, a_k \in O, k \geq 0$; when $k = 0$ we write $v \to in$ instead of $[v] \to in$ (*carry-in rules*);
   - $[va_1 \ldots a_k] \to out$, for $v \in V, a_1, \ldots, a_k \in O, k \geq 0$ when $k = 0$ we write $v \to out$ instead of $[v] \to out$ (*carry-out rules*);

   in each of these rules, v is the carrier and $a_1, \ldots, a_k$ are the passengers.

6. E is a finite set of *rules*, of the first three forms above, placed in the environment of the system;

7. $i_o \in \{1, \ldots, m\}$ is an elementary membrane of μ.

The meaning of the rules in $R_i, 1 \leq i \leq m$, and in E is as follows. By an attaching rule, the objects $a_1, \ldots, a_k$ get attached to the carrier v (a multiset of objects plus one copy of a carrier yields a *conglomerate* which behaves as a single body; the whole multiset is represented by a string; the fact that we have a single body is indicated by enclosing this string in square brackets). A detaching rule performs the opposite operation, separating all objects and the carrier from a conglomerate $[va_1 \ldots a_k]$. By moving-in and moving-out rules we can move conglomerates through membranes: *in* indicates that we have to go to one, nondeterministically chosen, of the adjacent inside membranes (if there is no lower level membrane, then the rule cannot be applied); *out* indicates the move outside the current membrane. Note that carriers can pass alone through membranes, but an object can never pass through a membrane alone (without being attached to a carrier). Moreover, an object is never modified by these rules. By carrier rules we can also send objects out of the system and, by using the rules from E, we can carry objects into the system, from the environment. At the beginning of a computation, we assume that the environment contains arbitrarily many copies of each object from O (but no carrier, they are present only inside the system, in a finite number of copies each, as specified by the initial multisets $w_1, \ldots, w_m$).

The number m of membranes is called the *degree* of the system, while the maximum number of objects which can be attached to a carrier (the maximum k in the rules from $R_1, \ldots, R_m, E$) is called the *carrying index* of the (carriers in the) system.

The multisets of objects and of carriers (including the information about their attachments) present in the m regions of the system, plus the multiset of carriers present outside the system (maybe with attached passengers) describe the *configuration* of the system at a given instance; thus, a configuration is an $(m+1)$-tuple of multisets of objects, carriers,

and conglomerates. The *initial configuration* is $(w_1, \ldots, w_m, \emptyset)$, with no conglomerate inside the system and no carrier outside it.

We pass from configuration to configuration by using the rules from $R_1, \ldots, R_m, E$, as explained in the Introduction, but now taking into account the specific nature of the rules of membrane systems with carriers. Again, as customary in P systems, there is a universal clock, the same for all membranes, and the use of any rule is supposed to take one time unit. The rules are applied in a maximally parallel manner; the rules and the objects/carriers for them are chosen in a nondeterministic manner, in an exhaustive way, so that no rule can be applied anymore to the remaining objects and carriers; the moving rules move objects and carriers from a region to an adjacent region; objects and carriers which are not used by the rules at a given step remain unchanged in the resulting configuration.

A sequence of transitions between configurations of the system constitutes a *computation*; a computation is successful if it *halts*, i.e., it reaches a configuration where no rule can be applied to any of the objects, carriers, or conglomerates.

The *result* of a successful computation is the number of objects (carriers are ignored) present within the membrane with the label i_o in the halting configuration. A computation which never halts yields no result. The set of all the numbers computed by Π is denoted by $N(\Pi)$.

The family of all sets $N(\Pi)$, computed as above by systems Π of degree at most $m \geq 1$, using at most $p \geq 1$ carriers, and with the carrying index not exceeding $k \geq 1$, is denoted by $NPC(m, p, k)$; when any of the parameters m, p, k is not limited, then we write $*$, getting in this way families of the form $NPC(m, *, *), NPC(*, *, k)$, etc.

Also, we use NRE to denote the family of recursively enumerable sets of natural numbers; this is precisely the family of the length sets of recursively enumerable languages, and we will make below an essential use of this observation.

3 The Computational Power

We will prove that P systems with carriers (even with a small number of membranes, a small number of carriers, and a small carrying index) are able to simulate Turing machines. In proofs we need the notion of a *matrix grammar with appearance checking* (see, e.g., [5]).

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

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied – 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}.

It is known that matrix grammars with appearance checking generate precisely the family RE of recursively enumerable languages.

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 $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$,
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 $x \in T^*, |x| \leq 2$.

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 caled 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 Lemma 1.3.7 in [5], for each matrix grammar there is an equivalent matrix grammar in the binary normal form.

We are now ready to prove the main result of this paper.

Theorem 1. *$NPC(*,*,*) = NPC(m,p,k) = NRE$, for all $m \geq 2, p \geq 3, k \geq 3$.*

Proof. For the inclusion $NPC(*,*,*) \subseteq NRE$ we can use the Turing-Church thesis or we can prove it directly, in a straightforward way (but involving a long construction). The inclusions $NPC(m,p,k) \subseteq NPC(m',p',k') \subseteq NPC(*,*,*)$, for all $1 \leq m \leq m', 1 \leq p \leq p', 1 \leq k \leq k'$, follow directly from the definitions. So, we only have to prove the inclusion $NRE \subseteq NPC(2,3,3)$. To this aim, we make use of the equality $RE = MAT_{ac}$. More precisely, we have $NRE = NMAT_{ac} = \{length(L) \mid L \in MAT_{ac}, L \subseteq a^*\}$ (where $length(L)$ is the length set of L, that is, the set of lengths of all strings in L). Consequently, it suffices to consider matrix languages over a one-letter alphabet.

Let $G = (N, \{a\}, S, M, F)$ be a matrix grammar with appearance checking in the binary normal form, with $N = N_1 \cup N_2 \cup \{S, \#\}$ and matrices of the four forms given above. Assume that we have s matrices of the form $(X \to \alpha, A \to x)$, with $X \in N_1, \alpha \in N_1 \cup \{\lambda\}, x \in (N_2 \cup \{a\})^*$, and t matrices of the form $(X \to Y, A \to \#)$, $X, Y \in N_1, A \in N_2$. Consider two new symbols, f, g and replace the matrices $(X \to \lambda, A \to x)$ by $(X \to f, A \to x)$; then, replace all matrices $(X \to \alpha, A \to x)$, $\alpha \in N_1 \cup \{f\}$, $x \in (N_2 \cup T)^*$, by $(X \to \alpha, A \to xg^i), i = 0, 1, 2$, such that $|xg^i| = 2$. We will still denote by $(X \to \alpha, A \to x)$ the so obtained matrices and by G the so obtained grammar. We label by $m_i, 1 \leq i \leq s$, the matrices $(X \to \alpha, A \to x)$ and by $m_{s+j}, 1 \leq j \leq t$, the matrices of the form $(X \to Y, A \to \#)$.

We construct the P system (of degree 2)

$$\Pi = (O, V, [_1[_2\ ]_2]_1, w_1, w_2, R_1, R_2, E, 2),$$

with

$$
\begin{aligned}
&O = N \cup \{a, d, e, f, g\} \cup \{b_i, c_i \mid 1 \leq i \leq s+t\},\\
&V = \{v, v', v''\},\\
&w_1 = v'v''d,\\
&w_2 = vb_1 \ldots b_{s+t}c_1 \ldots c_{s+t}XA, \text{ for } (S \to XA) \text{ the initial matrix of } G,
\end{aligned}
$$

and the sets of rules constructed in the following way.

1. For each matrix $m_i : (X \to \alpha, A \to \alpha_1\alpha_2), 1 \leq i \leq s$, of G, the following rules are in R_2:

   $$
   \begin{aligned}
   &vb_ic_iX \to [vb_ic_iX],\\
   &[vb_ic_iX] \to out,\\
   &[v'c_id] \to v'c_id,\\
   &v'c_idA \to [v'c_idA],\\
   &[v'c_idA] \to out,\\
   &[vb_i\alpha_1\alpha_2] \to vb_i\alpha_1\alpha_2,\\
   &[v'c_i\alpha e] \to v'c_i\alpha e;
   \end{aligned}
   $$

   the following rules are in R_1:

   $$
   \begin{aligned}
   &[vb_ic_iX] \to vb_ic_iX,\\
   &vb_iX \to [vb_iX],\\
   &[vb_iX] \to out,\\
   &v'c_id \to [v'c_id],\\
   &[v'c_id] \to in,\\
   &[v'c_idA] \to v'c_idA,\\
   &v'c_iA \to [v'c_iA],\\
   &[v'c_iA] \to out,\\
   &[vb_i\alpha_1\alpha_2] \to in,\\
   &[v'c_i\alpha e] \to in;
   \end{aligned}
   $$

   and the following rules are in E:

   $$
   \begin{aligned}
   &[vb_iX] \to vb_iX,\\
   &vb_i\alpha_1\alpha_2 \to [vb_i\alpha_1\alpha_2],\\
   &[vb_i\alpha_1\alpha_2] \to in,\\
   &[v'c_iA] \to v'c_iA,\\
   &v'c_i\alpha e \to [v'c_i\alpha e],\\
   &[v'c_i\alpha e] \to in.
   \end{aligned}
   $$

2. For each matrix $m_i : (X \to Y, A \to \#), s+1 \leq i \leq s+t$, in G, the following rules are in R_2:

   $$
   \begin{aligned}
   &vb_ic_iX \to [vb_ic_iX],\\
   &[vb_ic_iX] \to out,
   \end{aligned}
   $$

$[v'c_i] \to v'c_i$,
$v'c_iA \to [v'c_iA]$,
$[v'c_iA] \to out$,
$[vb_iYe] \to vb_iYe$;

the following rules are in R_1:

$[vb_ic_iX] \to vb_ic_iX$,
$vb_iX \to [vb_iX]$,
$[vb_iX] \to out$,
$v'c_i \to [v'c_i]$,
$[v'c_i] \to in$,
$[v'c_iA] \to in$,
$[vb_iYe] \to in$;

and the following rules are in E:

$[vb_iX] \to vb_iX$,
$vb_iYe \to [vb_iYe]$,
$[vb_iYe] \to in$.

3. For all $i = 1, 2, \ldots, s+t$ and for each $\alpha \in N_2$, the following rules are in R_2:

$vfb_i \to [vfb_i]$,
$[vfb_i] \to out$,
$vfc_i \to [vfc_i]$,
$[vfc_i] \to out$,
$vf\alpha \to [vf\alpha]$,
$[vf\alpha] \to out$,
$v''b_i \to [v''b_i]$,
$[v''b_i] \to out$,
$v''c_i \to [v''c_i]$,
$[v''c_i] \to out$,
$v''\alpha \to [v''\alpha]$,
$[v''\alpha] \to out$;

and the following rules are in R_1:

$[vfb_i] \to vfb_i$,
$[vfc_i] \to vfc_i$,
$[vf\alpha] \to in$,
$[v''b_i] \to in$,
$[v''c_i] \to in$,
$[v''\alpha] \to in$.

4. Also, the following rules are in R_2:

$v'e \to [v'e]$,
$[v'e] \to out$,
$[vf] \to vf$,
$vfg \to [vfg]$,
$[vfg] \to out$,
$v''f \to [v''f]$,
$[v''f] \to out$,
$v''g \to [v''g]$,
$[v''g] \to out$;

the following rules are in R_1:

$[v'e] \to v'e$,
$vf \to [vf]$,
$[vf] \to in$,
$[vfg] \to vfg$,
$[v''g] \to in$,
$[v''f] \to out$,
$v'' \to out$,
$v'' \to in$;

and the following rules are in E:

$[v''f] \to v''f$,
$v'' \to in$.

5. For all $\alpha \in N_1 \cup \{f\}$ we also introduce in R_2 the rules:

$v'\alpha \to [v'\alpha]$,
$[v'\alpha] \to out$,

and the following rule in R_1:

$[v'\alpha] \to in$.

The sets of rules R_1, R_2, and E contain only the rules specified by 1 through 5 above. This system works as follows.

In the initial configuration we have objects and carriers both in region 2 and in region 1, but the computation starts in region 2 (the carrier v'' from region 1 just goes to the environment and comes back, by using the rules $v'' \to out$ from R_1 and $v'' \to in$ from E). Assume that at some moment in region 2 we have a multiset consisting of one copy of a symbol $X \in N_1$, some copies of symbols from N_2 (initially, we have here the multiset XA), possibly copies of the object a, as well as the objects $b_1, \ldots, b_{s+t}, c_1, \ldots, c_{s+t}$.

Assume that in region 2 we use the rule $vb_ic_iX \to [vb_ic_iX]$ for some $1 \leq i \leq s$. This will start the simulation of the matrix $m_i : (X \to \alpha, A \to \alpha_1\alpha_2)$, which is done in the following way. The conglomerate $[vb_ic_iX]$ exits membrane 2 and gets detached within membrane 1. The object b_i together with v and X will exit the system, the object c_i will go inside membrane 2 together with the carrier v' and the object d, which was present

in region 1 from the beginning of the computation; v' and c_i will bring out also a copy of A. The conglomerate $[v'c_idA]$ gets detached within membrane 1, while v', c_i, A will go together into the environment, and d remains in region 1 for a further use.

From the environment, v (together with b_i) will bring into the central membrane the objects α_1, α_2, while v' (and c_i) will bring into membrane 2 the object α together with the auxiliary object e. Note that α is brought later than α_1, α_2, hence the simulation of using another matrix cannot be started before completing the simulation of the matrix m_i. The carrier v remains in region 2, the carrier v' returns to region 1, by making use of the object e (by using rules introduced in group 4).

Here is an important detail: if there is no copy of A in region 2, then the carrier v' waits here until v returns from the environment. In the presence of any symbol α from $N_1 \cup \{f\}$, the carrier v' uses a rule $v'\alpha \to [v'\alpha]$, and the conglomerate $[v'\alpha]$ passes forever back and forth through membrane 2, preventing in this way the halting of the computation. These operations are performed by using the rules from group 5. Consequently, the simulation of the matrix should be complete: both of its rules must be simulated.

The simulation of a matrix $m_i : (X \to Y, A \to \#)$, $s+1 \leq i \leq s+t$, is performed in the following way. Again, the carrier v brings to region 1 the objects b_i, c_i, together with X. The carrier and the objects b_i, X exit then the system, c_i goes back to the inner membrane, together with the carrier v'. If A is present in the multiset from region 2, then the rules $v'c_iA \to [v'c_iA], [v'c_iA] \to out$ from region 2 and $[v'c_iA] \to in$ from region 1 will be used forever. If A is not present, then v' waits in region 2 until v comes back, together with the objects Y and e; then v' returns to region 1 together with e.

Note that when waiting in region 2, the carrier v' cannot get attached to an object from $N_1 \cup \{f\}$, because no such object is present; when a symbol from N_1 is brought back by the carrier v, also e is present, hence we can continue the computation without entering a cycle.

Therefore, in both cases, the simulation of matrices is correct and we return to a configuration where all objects b_i, c_i, and the carriers v, v' are in the same regions as they were at the beginning of the computation. The fact that we have new copies of the object e in region 1 is of no importance, because no rule can be applied to this object here and, moreover, the output of a computation is read in membrane 2.

When the object f is brought to the inner membrane, this means that no symbol from N_1 is present and then, that the derivation in G should be terminal. In the presence of f, the carrier v carries to region 2 all objects b_i, c_i, g (the computation is not terminated as long as at least one symbol b_i, c_i is present in region 2) and it also checks whether or not the derivation is terminal. If the check yields a negative result, then the rules $[vf\alpha] \to out$ from membrane 2 and $[vf\alpha] \to in$ from region 1 will be used forever.

In order to remove also the object f from region 2 we use the carrier v''. During the whole computation, v'' has just passed across membrane 1 by using the rules $v'' \to out$ and $v'' \to in$ from region 1 and the environment, respectively. At any moment, this carrier can also go to membrane 2, by using the rule $v'' \to in$ from region 1. Here v'' can get attached to f and brings it to the environment; then it eventually returns to region 2. If any symbol $b_i, c_i, 1 \leq i \leq s+t$, or $\beta \in N_2 \cup \{g\}$ is present here, then the computation never stops, because of the rules in group 3 of the form $[v''\beta] \to out, [v''\beta] \to in$ which are present in regions 2 and 1, respectively. If no such a symbol is present in region 2,

then the computation stops. Thus, we cannot finish the computation before bringing the carrier v'' in region 2; but if this is done prematurely (while nonterminals from N_2 or symbols b_i, c_i are still present), then the computation will never halt.

Consequently, $N(\Pi) = \{n \mid a^n \in L(G)\}$. Because the largest conglomerates in our system have four elements ($[vb_ic_iX], [vb_i\alpha_1\alpha_2]$, $[vb_iYe]$, etc), we get $length(L(G)) \in NPC(2,3,3)$, which concludes the proof. □

4 Decreasing the Carrying Index

We do not know whether or not the number of carriers in the proof of Theorem 1 can be reduced (to two or to one), and whether or not the carrying index can be reduced without increasing the number of carriers used. At the price of the unbounded increase in the number of carriers, we can reduce the carrying index to two (but we do not know whether or not this result is optimal).

Theorem 2. *$NPC(*,*,*) = NPC(m,*,k) = NRE$, for all $m \geq 2, k \geq 2$.*

Proof. As in the previous proof, we start from a matrix grammar $G = (N, \{a\}, S, M, F)$ in the binary normal form, with $N = N_1 \cup N_2 \cup \{S, \#\}$ and with s matrices $m_i : (X \to \alpha, A \to \alpha_1\alpha_2), 1 \leq i \leq s$, and t matrices $m_{s+i} : (X \to Y, A \to \#), 1 \leq i \leq t$. If one of α_1, α_2 above is empty, then we replace it by the dummy symbol g; if α is empty, then we replace it by the special symbol f. Because a matrix of the form $(X \to \lambda, A \to x)$ is used at the last step of a derivation, the use of the corresponding matrix $(X \to f, A \to x)$ indicates the end of a derivation.

We construct the P system (of degree 2)

$$\Pi = (O, V, [_1[_2\,]_2]_1, w_1, w_2, R_1, R_2, E, 2),$$

with

$$\begin{aligned}
&O = N \cup \{a, d, e, f, g\} \cup \{b_i, c_i \mid 1 \leq i \leq s+t\},\\
&V = \{v_i, v_i' \mid 0 \leq i \leq s+t\},\\
&w_1 = v_0'v_1' \ldots v_{s+t}'d,\\
&w_2 = v_0v_1 \ldots v_{s+t}b_1 \ldots b_{s+t}XA, \text{ for } (S \to XA) \text{ the initial matrix of } G,
\end{aligned}$$

and the sets of rules constructed in the following way.

1. For each matrix $m_i : (X \to \alpha, A \to \alpha_1\alpha_2), 1 \leq i \leq s$, of G, the following rules are in R_2:

$$\begin{aligned}
&v_ib_iX \to [v_ib_iX],\\
&[v_ib_iX] \to out,\\
&[v_i'b_id] \to v_i'b_id,\\
&v_i'dA \to [v_i'dA],\\
&[v_i'dA] \to out,\\
&[v_i\alpha_1\alpha_2] \to v_i\alpha_1\alpha_2,\\
&[v_i'\alpha e] \to v_i'\alpha e;
\end{aligned}$$

the following rules are in R_1:

$[v_i b_i X] \to v_i b_i X$,
$v_i X \to [v_i X]$,
$[v_i X] \to out$,
$v_i' b_i d \to [v_i' b_i d]$,
$[v_i' b_i d] \to in$,
$[v_i' d A] \to v_i' d A$,
$v_i' A \to [v_i' A]$,
$[v_i' A] \to out$,
$[v_i \alpha_1 \alpha_2] \to in$,
$[v_i' \alpha e] \to in$;

and the following rules are in E:

$[v_i X] \to v_i X$,
$v_i \alpha_1 \alpha_2 \to [v_i \alpha_1 \alpha_2]$,
$[v_i \alpha_1 \alpha_2] \to in$,
$[v_i' A] \to v_i' A$,
$v_i' \alpha e \to [v_i' \alpha e]$,
$[v_i' \alpha e] \to in$.

2. For each matrix $m_i : (X \to Y, A \to \#), s+1 \leq i \leq s+t$, in G, the following rules are in R_2:

$v_i b_i X \to [v_i b_i X]$,
$[v_i b_i X] \to out$,
$[v_i' b_i] \to v_i' b_i$,
$v_i' A \to [v_i' A]$,
$[v_i' A] \to out$,
$[v_i Y e] \to v_i Y e$,
$v_i' e \to [v_i' e]$,
$[v_i' e] \to out$;

the following rules are in R_1:

$[v_i b_i X] \to v_i b_i X$,
$v_i X \to [v_i X]$,
$[v_i X] \to out$,
$v_i' b_i \to [v_i' b_i]$,
$[v_i' b_i] \to in$,
$[v_i' A] \to in$,
$[v_i Y e] \to in$,
$[v_i' e] \to v_i' e$;

and the following rules are in E:

$[v_iX] \to v_iX,$
$v_iYe \to [v_iYe],$
$[v_iYe] \to in.$

3. For all $i = 1, 2, \ldots, s+t$ and for each $\alpha \in N_2$, the following rules are in R_2:

$v_0fb_i \to [v_0fb_i],$
$[v_0fb_i] \to out,$
$v_0fc_i \to [v_0fc_i],$
$[v_0fc_i] \to out,$
$v_0f\alpha \to [v_0f\alpha],$
$[v_0f\alpha] \to out,$
$v_0'b_i \to [v_0'b_i],$
$[v_0'b_i] \to out,$
$v_0'\alpha \to [v_0'\alpha],$
$[v_0'\alpha] \to out;$

the following rules are in R_1:

$[v_0fb_i] \to v_0fb_i,$
$[v_0fc_i] \to v_0fc_i,$
$[v_0f\alpha] \to in,$
$[v_0'b_i] \to in,$
$[v_0'\alpha] \to in;$

the following rules are in R_2:

$v_i'e \to [v_i'e],$
$[v_i'e] \to out,$
$[v_0f] \to v_0f,$
$v_0fg \to [v_0fg],$
$[v_0fg] \to out,$
$v_0'g \to [v_0'g],$
$[v_0'g] \to out;$

the following rules are in R_1:

$[v_i'e] \to v_i'e,$
$v_0f \to [v_0f],$
$[v_0f] \to in,$
$[v_0fg] \to v_0fg,$
$[v_0'g] \to in,$
$[v_0'f] \to out,$
$v_0' \to out,$
$v_0' \to in;$

and the following rules are in E:

$[v_0' f] \to v_0' f,$
$v_0' \to in.$

4. For $i = 1, \ldots, s$ and $\alpha \in N_1 \cup \{f\}$ the following rules are in R_2:

   $v_i' \alpha \to [v_i' \alpha],$
   $[v_i' \alpha] \to out;$

   and the following rule is in R_1:

   $[v_i' \alpha] \to in.$

This system works very much in the same way as the system constructed in the proof of Theorem 1, but instead of controlling the work of carriers through objects b_i, c_i, we now also use the subscripts of the carriers. Moreover, the objects $b_i, 1 \leq i \leq s+t$, and g are removed from region 2 using the carrier v_0 (v_0 is doing nothing as long as the object f is not present in region 2), while the object f is removed using the carrier v_0' (v_0' plays the role of v'' from the previous proof).

Thus we conclude that $N(\Pi) = \{n \mid a^n \in L(G)\}$. Since no carrier in the construction carries more than two passengers, the result holds. □

5 Final Remarks

We have introduced a class of membrane systems where objects do not evolve, but instead they are carried through membranes by carriers; "sufficient" numbers of copies of each object are available in the environment. Such systems are shown to be computationally complete, i.e., they can compute all recursively enumerable sets of natural numbers. This result is true even for systems with a reduced number of membranes, a reduced number of carriers, and a reduced carrying index.

It is worth mentioning that the systems with carriers have a property which was not considered yet in other sorts of membrane systems investigated so far: they observe the *conservation law*. This means that no object is created "from nothing" and no object is destroyed (but we can have arbitrarily many objects inside the system, because the environment provides sufficient copies of each object).

As explained in the Introduction, membrane systems with carriers perform computations by communication only. Hence, in our opinion, this paper contributes to the understanding of the role of communication in membrane systems – in fact, the results of this paper point out the power of communication in membrane systems. We think that this paper is only a beginning of a systematic investigation of this topic. A possible next step would be to classify various sorts of communication (like, e.g., one-way versus two-way, or contextual versus context-independent), and investigate their power.

References

[1] B. Alberts et al., *Essential Cell Biology. An Introduction to the Molecular Biology of the Cell*, Garland Publ. Inc., New York, London, 1998.

[2] T.A. Brown, *Gene Cloning. An Introduction*, Chapman & Hall, London, 1996.

[3] C. Calude, Gh. Păun, *Computing with Cells and Atoms*, Taylor and Francis, London, 2000.

[4] 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, and *Auckland University, CDMTCS Report* No 123, 2000 (www.cs.auckland.ac.nz/CDMTCS).

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

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

[7] T. Head, G. Rozenberg, R. Bladergroen, K. Breek, P.H.M. Lommerese, H. Spaink, The plasmid alternative for biomolecular computing, *Biosciences*, 2000, to appear.

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

[9] W.R. Loewenstein, *The Touchstone of Life. Molecular Information, Cell Communication, and the Foundations of Life*, Oxford Univ. Press, New York, Oxford, 1999.

[10] 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).

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

[12] Gh. Păun, Computing with membranes – A variant: P systems with polarized membranes, *Intern. J. of Foundations of Computer Science*, 11, 1 (2000), 167–182.

[13] Gh. Păun, P systems with active membranes: Attacking NP complete problems, *J. Automata, Languages, and Combinatorics*, 6, 1 (2001), to appear, and *Auckland University, CDMTCS Report* No 102, 1999 (www.cs.auckland.ac.nz/CDMTCS).

[14] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, 41, 3 (2000), 259–266.

[15] Cl. Zandron, Cl. Ferretti, G. Mauri, Solving NP-complete problems using P systems with active membranes, *Proc. Second Conf. Unconventional Models of Computing* (I. Antoniou, C. S. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000.

[16] Cl. Zandron, Cl. Ferretti, G. Mauri, Using membrane features in P systems, *Pre-proc. Workshop on Multiset Processing*, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 2000, 296–320.