Prof. Dr. Jürgen Dassow
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik

F O R M A L   L A N G U A G E S

A N D

B I O L O G I C A L   P R O C E S S E S

Vorlesungsmanuskript

Magdeburg, April - July 2008

Introduction

In the end of the fifties as N. Chomsky has introduced the well-known classes of regular, context-free and context-sensitive languages the aim was to model the syntax of natural languages. Based on the Backus-Naur form for the description of the syntax of programming languages, in the beginning of the sixties S. Ginsburg and H.G. Rice noticed that the grammars introduced by Chomsky can be used for programming languages, too. Since that time until at least the middle of the seventies most investigations to formal languages followed this approach. The central feature of such grammars is a sequential process of rewriting of subwords.

On the other hand one has to mention that already since the fifties there exist some devices nearly related to formal languages which were motivated and/or applied to biological phenomena. The well-known Kleene Theorem on the description of regular languages by means of algebraic operations was discovered by S.C. Kleene as he represented the events in nerve nets. Furthermore, it was known that cellular automata are able to a self-replicating behaviour known from biological organisms or colonies of organisms. But in both cases, in order to model the biological processes finite automata or collections of finite automata have been used.

Since the seventies the situation changed completely. Motivated by biological processes new types of grammars have been introduced and their investigation dominated in a certain sense the development of the theory of formal languages.

In 1968 the first approach was initiated by A. Lindenmayer (see [16]. Cell divisions, changes of states of the cells, death of cells etc. were modelled by production as one uses in Chomsky grammars. However, the rewriting process by application of rules is a parallel one because cell divisions, changes of cell states etc. proceed in parallel. The large interest in these Lindenmayer systems originated from the biological motivation as well as by the interest in a comparison between sequential and parallel processes in computer science. The monograph [13] presents a summary of the state of the theory of developmental systems and languages in 1975 and considers intensively motivation from and application to bilogy, whereas the monograph [27] emphasizes the mathematical theory of such systems. Further summaries and material can be found in [26], [17], [28], [29], [15]. In [25] the authors use Lindenmayer systems to generate graphical representations of plants.

Although DNA sequences are twisted strands (in a 3-dimensional space) it is very natural to model them by (linear) strings/words. Mutations of DNA sequences, genes, chromosomes etc. caused by deletions, insertions, splicings, inversions etc. can be described by operations on words. Iterated applications of these operations model the evolution of molecules. Thus we have sequential process, again, however, the basic step is not a rewriting. After the first investigations in this direction by T. Head (see [11]) in the last decade a lot of papers appeared studying the behaviour of formal languages under these operations. Moreover, one has to mention that these considerations are nearly related to some aspects of molecular computing (see [1], [18]). The book [23] is the first monograph on this topic, summaries are contained in [2], [12], [24], [7].

An approach – called membrane systems – to describe the behaviour of a single cell was startet by Gh. Păun in the paper [21]. A cell is considered as an object with membranes which define substructures of the cell, e.g. the kernel of the cell. Changes of the objects in the different regions of the cell are described by rules associated with the regions. However, the rules are not applied to words as in the two types of grammars mentioned above, the rules are applied to multisets since the objects in a region form a multiset. The books [22] and [2] summarize parts of the theory developed for these grammatical systems.

We mention that these three new types of grammars/languages are natural by their motivation from biology as well as by the fact that they allow nice characterizations of well-known classes of formal languages.

In this lecture we shall emphasize Lindenmayer systems, languages and systems using operations as splicing and membrane systems. We shall omit grammars with valuations (see [5]), eco-grammar systems (see [4]) and other language generating devices modelling aspects of biology.

Throughout this lecture we assume that the students/reader is familiar with the basic concepts of the theory of formal languages as usually presented in basic courses on Theoretical Computer Science and with some facts of mathematics (especially linear algebra, theory of difference equations, combinatorial formulae, etc). The notation, some definitions and results are summarized in the first chapter.

Jürgen Dassow April - July 2008

Contents

Chapter 4

Membrane Systems

4.1 Further Basics

In this section we introduce two further types of grammars. The common feature is that they use only context-free rules, however, by some restrictions in the application of rules a larger generative power than that of context-free grammars is obtained. These grammars will be used in the sequel to discuss the power of membrane systems which are the subject of this chapter.

We start with the definition of a matrix grammar[1]. Essentially instead of context-free rules finite sequences of context-free rules are considered and if one applies the first rule of such a sequence one has to apply the further rules of this sequence in the given order.

Definition 4.1 *i) A matrix grammar is a quintuple $G = (N, T, M, S, F)$ where*

- *N, T and S are specified as in a context-free grammar,*
- *$M = \{m_1, m_2, \dots m_n\}$ is a finite set of finite sequence of context-free rules, i.e., for $1 \le i \le n$,*
  $$m_i = (A_{i,1} \to w_{i,1}, A_{i,2} \to w_{i,2}, \dots, A_{i,r_i} \to w_{i,r_i})$$
  for some $r_i \ge 1$, $A_{i,j} \in N$, $w_{i,j} \in (N \cup T)^*$, $1 \le j \le r_i$,
- *F is a subset of the rules occurring in the sequences m_i, $1 \le i \le n$.*

ii) For a matrix $m = (A_1 \to w_1, A_2 \to w_2, \dots, A_r \to w_r) \in M$, we say that x derives y by m, written as $x \Longrightarrow_m y$ if there exist words $x_1, x_2, \dots x_{r+1}$ such that the following conditions hold:

- *$x = x_1$, $y = x_{r+1}$,*
- *for $0 \le i \le r-1$, $x_i = x_i' A_i x_i''$ and $x_{i+1} = x_i' w_i x_i''$ or A_i does not occur in x_i, $x_{i+1} = x_i$ and $A_i \to w_i \in F$.*

[1]To be precise, we introduce matrix grammar with appearance checking and with erasing rules. Because the other more restricted types of matrix grammars will not be used we only use the term matrix grammar.

iii) The language $L(G)$ generated by G consists of all words $z \in T^*$ which have a derivation

$$S \Longrightarrow_{m_{i_1}} w_1 \Longrightarrow_{m_{i_2}} w_2 \Longrightarrow_{m_{i_3}} \ldots \Longrightarrow_{m_{i_t}} = w_t = z$$

where $t \geq 1$ and $m_{i_j} \in M$ for $1 \leq j \leq t$.

The sequences $m \in M$ are called matrices. By definition the rules of a matrix have to be applied in the given order and all matrices of a matrix have to be applied where applications means a usual application if the left hand side occurs in the sentential form or no change if the left hand side does not occur in the sentential form and the rule belongs to F.

By $\mathcal{L}(MAT)$ we denote the family of all languages which can be generated by matrix grammars.

We give two examples.

Example 4.2 Let $G_1 = (\{S, A, B\}, \{a, b, c\}, \{m_1, m_2, m_3\}, S, \emptyset)$ be a matrix grammar with

$$m_1 = (S \to AB),\ m_2 = (A \to aAb, B \to Bc),\ \text{and } m_3 = (A \to ab, B \to c).$$

Then any derivation has the form

$$\begin{aligned} S &\Longrightarrow_{m_1} AB \Longrightarrow_{m_2} aAbBc \Longrightarrow_{m_2} a^2Ab^2Bc^2 \Longrightarrow_{m_2} a^3Ab^3Bc^3 \Longrightarrow_{m_2} \cdots \\ &\Longrightarrow_{m_2} a^{n-1}Ab^{n-1}Bc^{n-1} \Longrightarrow_{m_3} a^nb^nc^n, \end{aligned}$$

which yields that

$$L(G_1) = \{a^nb^nc^n \mid n \geq 1\}.$$

Example 4.3 We consider the matrix grammar

$$G_2 = (\{S, A, B, X, Y, Z, \#\}, \{a\}, \{m_1, m_2, \ldots, m_8\}, S, \{A \to \#, B \to \#\})$$

where
$$\begin{array}{ll} m_1 = (S \to XA), & m_2 = (X \to X, A \to BB), \\ m_3 = (X \to Y, A \to \#), & m_4 = (Y \to Y, B \to A), \\ m_5 = (Y \to X, B \to \#), & m_6 = (Y \to Z, B \to \#), \\ m_7 = (Z \to Z, A \to a), & m_8 = (Z \to \lambda, A \to a). \end{array}$$

Let us assume, that we have a sentential form XA^n for some $n \geq 1$; note that by the application of the matrix m_1 (which has been used in the first step) we obtain such a word with $n = 1$. We cannot apply the matrix m_3 since it introduces the nonterminal # which cannot be replaced, i.e., the derivation cannot be terminated. Hence the only applicable rule is m_2 which gives $XA^{n_1}BBA^{n_2}$ with $n_1 + n_2 = n - 1$. Again, m_2 is the only applicable if $n - 1 > 1$; moreover, this situation holds as long as a letter A is present. Thus we get after n applications of m_2 the sentential form XB^{2n}. Now the only applicable matrix is m_3 where $A \to \#$ cannot be applied which is allowed by $A \to \# \in F$. Now we have to proceed with $2n$ application of m_4 which yields YA^{2n}. Now we have two possibilities; we use m_5 or m_6. In the former case we obtain the sentential form XA^{2n} which has the same form as our starting sentential form; only the number of occurrences of A is doubled. In the latter case we have to apply $2n - 1$ times the matrix m_7 and once m_8 which results in a^{2n} (note that m_8 cannot be applied earlier since we then obtain a sentential with no occurrence of X, Y, Z, i.e., the derivation is blocked). Thus we double the number of A's or we terminate. Therefore

$$L(G_2) = \{a^{2^n} \mid n \geq 1\}.$$

Obviously, if all matrices have length 1, i.e., they consist of one rule only, then the application of the matrix coincides with the application of its rule. Thus such matrix grammars generate only context-free languages and all context-free languages can be generated. The example shows that also non-context-free languages can be generated by matrix grammars. Without proof we give that the generative power of matrix grammars equals the power of arbitrary phrase structure grammars.

Theorem 4.4 *$\mathcal{L}(MAT) = \mathcal{L}(RE)$.* □

We now present a normal form for matrix grammars.

Definition 4.5 *A matrix grammar $G = (N, T, M, S, F)$ is in normal form if the following conditions hold:*

- $N = N_1 \cup N_2 \cup \{S, Z, \#\}$, $S, Z \notin N_1 \cup N_2$, $N_1 \cap N_2 = \emptyset$
- *any matrix of M has one of the following forms*
  — $(S \to XA)$ *with* $X \in N_1$, $A \in N_2$,
  — $(X \to Y, A \to w)$ *with* $X, Y \in N_1$, $A \in N_2$, $w \in (N_2 \cup T)^*$,
  — $(X \to Y, A \to \#)$ *with* $X \in N_1$, $Y \in N_1 \cup \{Z\}$, $A \in N_2$,
  — $(Z \to \lambda)$,
- *there is only one matrix of the form $(S \to XA)$ in M,*
- *F consists of all rules of the form $A \to \#$ with $A \in N_2$.*

The following theorem shows that the naming normal form is used correctly.

Theorem 4.6 *For any recursively enumerable language L, there is a matrix grammar G in normal form such that $L(G) = L$.*

Proof. We first proof that the required special forms of matrices are sufficient. Let L be a recursively enumerable language. By Theorem 4.4, there is a matrix grammar $G' = (N, T, M, S', F)$ such that $L(G') = L$. We assume that

$$\begin{aligned} N &= \{A_1, A_2, \ldots, A_t\}, \\ M &= \{m_1, m_2, \ldots m_n\}, \\ m_i &= (A_{i,1} \to w_{i,1}, A_{i,2} \to w_{i,2}, \ldots, A_{i,r_i} \to w_{i,r_i}) \text{ for } 1 \leq i \leq n. \end{aligned}$$

We construct the matrix grammar G in normal form by the settings

$$\begin{aligned} N_1 &= \{[i,j] \mid 1 \leq i \leq n,\ 1 \leq j \leq r_i\} \cup \{[k] \mid 1 \leq k \leq t, \\ N_2 &= N, \\ &\quad \text{new letters } S, Z, \#, \end{aligned}$$

(1) $(S \to [i,1]S')$ for $1 \le i \le n$,
(2) $([i,j] \to [i,j+1], A_{i,j} \to w_{i,j})$ for $1 \le i \le n,\ 1 \le j < r_i$,
(3) $([i,j] \to [i,j+1], A_{i,j} \to \#)$ for $1 \le n,\ 1 \le j < r_i,\ A_{i,j} \to w_{i,j} \in F$,
(4) $([i,r_i] \to [i',1], A_{i,r_i} \to w_{i,r_i})$ for $1 \le i \le n,\ 1 \le i' < n$,
(5) $([i,r_i] \to [i',1], A_{i,r_i} \to \#)$ for $1 \le i \le n,\ 1 \le i' < n,\ A_{i,r_i} \to w_{i,r_i} \in F$,
(6) $([i,r_i] \to [1], A_{i,r_i} \to w_{i,r_i})$ for $1 \le i \le n$,
(7) $([i,r_i] \to [1], A_{i,r_i} \to \#)$ for $1 \le i \le n,\ A_{i,r_i} \to w_{i,r_i} \in F$,
(8) $([i] \to [i+1], A_i \to \#)$ for $1 \le i \le t-1$,
(9) $([t] \to Z, A_t \to \#)$,
(10) $(Z \to \lambda)$.

We have $L(G') = L(G)$ by the following reasons. We start with an application of a matrix of type (1), which says that the application of the i-th matrix is started. The simulation is performed by applying in succession the rules of type (2) or (3) with left hand sides $[i,1], [i,2], \ldots, [i, r_i - 1]$ in their first rules and finishing the simulation with rules of type (4), (5), (6) or (7) with left hand side $[i, r_i]$ in its first rule. The matrices of types (3) and (5) can only be applied if the nonterminal $A_{i,j}$ and A_{i,r_i} does not occur in the sentential form since otherwise the nonterminal # is introduced which cannot be changed (there are no rules with left hand side #), i.e., we cannot derive a terminal word. After the simulation of a complete matrix of G', we start another simulation of a matrix if we applied a rule of type (4) or (5) and we start the applications of type (8) and (9) if we applied matrices of type (6) or (7). By the matrices of type (8) and (9) we check that no nonterminal is present in the sentential form (otherwise a # is introduced). Finally, we cancel the first letter Z. Thus any derivation consists of simulations of the application of matrices in G followed by a check that the word is terminal.

It remains to show that one rule of the form $(S \to XA)$ is sufficient. In order to prove this we change G' to $G'' = (N \cup \{S''\}, T, M \cup \{(S'' \to S'), S'', F)$. It is obvious that $L(G') = L(G'')$ since any derivation has to start with $S'' \Longrightarrow S'$. Moreover, there is a unique matrix $(S'' \to S')$ of G'' which has to be used in the first step. Such the construction of G as above starting from G'' requires only the matrix $(S \to [i,1]S'')$ where i refers to $(S'' \to S')$. $\square$

The second concept is that of grammar systems[2] The basic idea can be illustrated as follows. Some (context-free) grammars are sitting around a table and a word is placed on the table. Now a grammar G can take the word and derive it as long productions of the grammar G are applicable. If no rule can be applied by G, then G puts the newly derived word back to the table. Obviously, this process can be iterated. We have a cooperation of the grammars since rules of another grammar cannot be used if a grammar works.

We now give the formal definition.

Definition 4.7 *i) A grammar system with n components is an $(n+3)$-tuple*

$$G = (N, T, P_1, P_2, \ldots, P_n, S)$$

[2]To be precise we consider here cooperating distributed grammar systems with terminating derivation mode t; however, since other types of grammar systems are not used, we use the term grammar system only.

where

- *N, T, S are specified as in a context-free grammar,*
- *$P_1, P_2, \ldots, P_n$ are finite subsets of $N \times (N \cup T)^*$, i.e., P_i is a finite set of context-free rules for $1 \leq i \leq n$.*

ii) We say that x derives y by the set P_i, $1 \leq i \leq n$, written as $x \Longrightarrow^t_{P_i} y$ if $x \Longrightarrow_{P_i} y$, i.e., y can be obtained from x by a derivation which only uses rules from P_i, and no rule of P_i can be applied to y.

iii) The language $L(G)$ generated by the grammar system G consists of all word $z \in T^*$ which can be generated by a derivation of the form

$$S \Longrightarrow^t_{P_{i_1}} w_1 \Longrightarrow^t_{P_{i_2}} w_2 \Longrightarrow^t_{P_{i_3}} \ldots \Longrightarrow^t_{P_{i_s}} w_s = z$$

for some $t \geq 1$, $1 \leq i_j \leq n$, $1 \leq j \leq s$.

The sets $P_1, P_2, \ldots, P_n$ are called the components of the grammar system.

By $\mathcal{L}_n(CF)$ we denote the set of languages which can be generated by grammar systems with n components.

We now present two examples which generate the same languages as the matrix grammars considered in Examples 4.2 and 4.3.

Example 4.8 Let $G'_1 = (\{S, A, B, A', B'\}, \{a, b, c\}, P_1, P_2, P_3, S)$ be a grammar system with the three components

$$P_1 = \{S \to AB, A \to aA'b, B \to B'c\},\ P_2 = \{A' \to A, B' \to B\},\ P_3 = \{A \to \lambda, B \to \lambda\}.$$

Obviously, any derivation in the grammar system G'_1 has the form

$$\begin{aligned} S &\Longrightarrow^t_{P_1} aA'bB'c \Longrightarrow^t_{P_2} aAbBc \Longrightarrow^t_{P_1} a^2A'b^2B'c^2 \Longrightarrow^t_{P_2} a^2Ab^2Bc^2 \Longrightarrow^t_{P_1} \ldots \\ &\Longrightarrow_{P_2} a^nAb^nBc^n \Longrightarrow_{P_3} a^nb^nc^n, \end{aligned}$$

which gives

$$L(G'_1 = \{a^nb^nc^n \mid n \geq 1\}.$$

Example 4.9 We consider the grammar system $G'_2 = (\{S, S'\}, \{a\}, P_1, P_2, P_3, S)$ with the three components

$$P_1 = \{S \to S'S'\},\ P_2 = \{S' \to S\},\ P_3 = \{S \to a\}.$$

Then any derivation is of the form

$$\begin{aligned} S &\Longrightarrow^t_{P_1} S'S' \Longrightarrow^t_{P_2} SS \Longrightarrow^t_{P_1} (S')^4 \Longrightarrow^t_{P_2} S^4 \Longrightarrow^t_{P_1} (S')^8 \Longrightarrow^t_{P_2} S^8 \ldots \\ &\Longrightarrow^t_{P_2} S^{2^t} \Longrightarrow^t_{P_3} a^{2^n} \end{aligned}$$

and, consequently,

$$L(G'_2) = \{a^{2^n} \mid n \geq 0\}.$$

It is clear that a grammar system with one component is a context-free grammar. Therefore $\mathcal{L}_1(CF) = \mathcal{L}(CF)$. However, if we use three components, then non-context-free languages can be generated. The following theorem says that we need three components in order to generate non-context-free languages and that three components are sufficient to generate languages which can be obtained by an arbitrary number of components. We omit the proof which needs some knowledge on further closure properties of $\mathcal{L}(CF)$ and on extended tabled Lindenmayer systems.

Theorem 4.10 *i)* $\mathcal{L}(CF) = \mathcal{L}_1(CF) = \mathcal{L}_2(CF)$.
ii) For any $n \geq 3$, $\mathcal{L}_n(CF) = \mathcal{L}_3(CF)$. $\square$

4.2 Basic Type of Membrane Systems and its Power

The idea of membrane systems is to model a biological cell as a computing device. A cell is considered as a membrane which contains further membranes which can contain membranes again. For instance the kernel of a cell gives a membrane contained in the skin membrane of the cell. Moreover, there is a change of the contents of each of the cells according to bio-chemical reactions inside a membrane, and there is an exchange of molecules through the membranes. If one considers the state of the cell, i.e., the molecules inside the membranes, as a configuration, then the above mentioned reactions lead to a change of the configuration. Therefore we have something which looks as a computation. However, inside of each membrane we only have a finite multiset of objects; therefore the computation is not done via words, it is done via multisets.

In this chapter a multiset M over a finite alphabet V will be described by a word w_M such that the number $\#_a(w_M)$ of the letter a in the word w_M coincides with the multiplicity $M(a)$. Obviously, w_M is not determined uniquely if M is given, since a change of the order of the letters in a word does not change the multiplicities but the word. In the sequel we shall mostly used a word w_M which fits best for our purposes. Moreover, we shall use the words "multiset" and "word" as representatives of the same object.

In Figure 1 we give a cell by the outer skin membrane 1 containing two membranes 2 and 3 and the membrane 2 contains a further membrane 4. Moreover, the content of the cell itself is *abb*, the contents of the three membranes 2, 3, and 4 inside the cell are *bc*, *aac*, and *abc* respectively.

The first problem is to describe the membrane structure. This can be done by a tree, where the outer skin membrane is the root and x is a son of y if and only if the membrane y contains the membrane x. The membrane structure of the cell given in Figure 4.1 is then represented by

$$\begin{tikzcd}
 & 1 \arrow[dl, no head] \arrow[dr, no head] & \\
2 \arrow[d, no head] & & 3 \\
4 & &
\end{tikzcd}$$

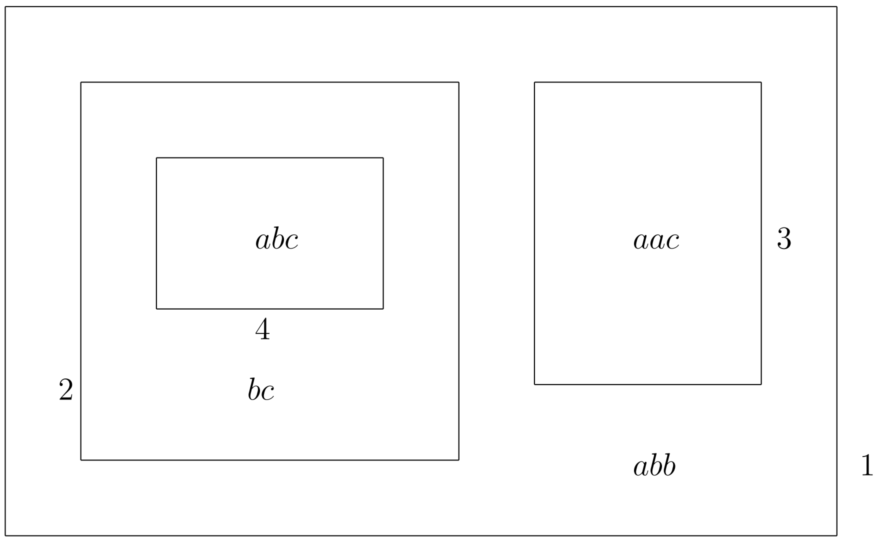

Figure 4.1: A membrane structure

A further possibility to give a membrane structure is a correct sequence of indexed brackets where the index refers to the membrane. The outer membrane is represented by $[_1]_1$. If one has already a membrane structure where $[_i$ is followed by $]_i$, i.e., the sequence of brackets has the form $w[_i]_iw'$, and the i-th membrane contains the membranes $j_1, j_2, \ldots, j_s$, then we get a bracket word

$$w[_i[_{j_1}]_{j_1}[_{j_2}]_{j_2} \cdots [_{j_s}]_{j_s}]_i \ .$$

The structure given in Figure 4.1 is represented by $[_1[_2[_4]_4]_2[_3]_3]_1$.

A membrane is called *simple* if there is no membrane inside of it. In terms of trees which describe a membrane structure, the leaves correspond to simple membranes.

We also have to clarify the concept of a rule in a membrane system because we cannot only change a letter or a multiset of letters, i.e., a word, we can also move letters or multisets of letters through membranes. Obviously, a letter is kept in a membrane, it can go out of the membrane, or it can move into a membrane which is inside the given membrane. Therefore we define the set Tar consisting of *here*, *out* and in_j where j refers to the j-th membrane. Thus we formally define a rule in a membrane system as a pair

$$(x_1x_2 \ldots x_n, (y_1, t_1)(y_2, t_2) \ldots (y_m, t_m))$$

where x_i and y_j are letters for $1 \leq i \leq n$ and $1 \leq j \leq m$, and $t_j \in Tar$ for $1 \leq j \leq m$. The application of this rule to the multiset $x_1x_2 \ldots x_n$ in membrane k is performed as follows: the multiset $x_1x_2 \ldots x_n$ is taken away from the multiset of j, the letters y_q, $1 \leq q \leq m$,
— are added to the multiset in membrane k, if $t_j = here$,
— are added to the multiset in membrane k', if $t_j = out$ and membrane k' contains membrane j,
— are given to the environment (and are lost) if $t_j = out$ and membrane k is the outer membrane,
— are added to the multiset in membrane p, if $t_j = in_p$ and membrane k contains membrane p. We note that, obviously, given a membrane k, the targets of the rules

applicable to multisets in membrane j – besides *here* and *out* – can only be numbers of membranes which are contained in membrane k, i.e., which are sons of k in the tree describing the membrane structure. Moreover, *out* defines a unique membrane or the environment to which the letters have to go.

Again, we write $x_1x_2\dots x_n \to (y_1,t_1)(y_2,t_2)\dots(y_m,t_m)$ for a rule.

In order to simplify the notation, we write a instead of $(a, here)$.

Before giving the formal definition of a membrane system we shortly discuss the problem of defining the generated languages. Obviously, since the membranes contain multisets, only multisets can be generated. In a (context-free) grammar a derivation is finished iff the generated word contains only terminals, or in other words, no rule can be applied to the generated sentential forms. Therefore it is of interest to consider such multisets which are in the system if no rule is applicable. There are at least two possibilities for the choice of the generated multiset: take the union of all multisets present in the membranes or choose a special membrane and take the multiset in that membrane. We shall follow the second idea. Moreover, we shall not consider multisets, which count how often a letter occurs; we shall consider only the number of letters occurring in the multiset, that is the length of the word describing the multiset.

Let L be a language. Then we set

$$N(L) = \{n \mid n = |w| \text{ for some } w \in L\},$$

i.e., $N(L)$ is the set of all lengths of words in L.

Let X be a set of grammars. Then we set

$$N(X) = \{N(L) \mid L \in \mathcal{L}(X).$$

Without proof we mention the following statements.

Theorem 4.11 *i)* $N(REG) = N(CF) \subset N(CS) \subset N(RE)$.

ii) A set M of natural numbers belongs to $N(CF)$ if and only if there are numbers $r, s, p, q_1, q_2, \dots q_r, p_1, p_2, \dots, p_s$ such that $r \geq 0$, $s \geq 0$, $p \geq 1$, $q_1 < q_2 < \dots < q_r < p_1 < p_2 < \dots < p_s$ and

$$M = \{q_1, q_2, \dots, q_r\} \cup \bigcup_{i=1}^{s} \{p_i + np \mid n \in \mathbf{N}_0\}.$$

We now give the formal definition of a membrane system.

Definition 4.12 *i) A membrane system with m membranes is a $(2m+3)$-tuple*

$$\Gamma = (V, \mu, w_1, w_2, \dots w_m, R_1, R_2, \dots R_m, i)$$

where

- *V is a finite alphabet (of objects occurring in the membranes),*
- *μ is a membrane structure (of m membranes),*
- *for $1 \leq j \leq m$, w_j is a word over V (giving the initial content of membrane j),*

- *for* $1 \leq j \leq m$, R_j *is a finite set of rules which can be applied to words in membrane* j,
- i *is a natural number such that* $1 \leq i \leq m$ *and the membrane* i *is a simple membrane (the output membrane).*

ii) A configuration of Γ *is an* m*-tuple of multisets/words.*

For two configurations $C = (u_1, u_2, \ldots, u_m)$ *and* $C' = (u'_1, u'_2, \ldots, u'_m)$, *we say that* C *is transformed to* C' *by* Γ, *written as* $C \vdash C'$ *if and only if* C' *is obtained from* C *by a maximal parallel application of rules of* R_i *to* u_i *for all* i, $1 \leq i \leq m$, *i.e., no rule of* R_i *can be applied to the multiset which remains after subtracting all sets to which rules are applied from* u_i.

iii) A configuration $C = (u_1, u_2, \ldots, u_m)$ *is called halting iff no rule of* R_i *is applicable to* u_i *for* $1 \leq i \leq m$.

The language $L(\Gamma)$ *generated by a membrane system* Γ *is the set of all numbers* n *such that there is a halting configuration* $C = (u_1, u_2, \ldots, u_m)$ *of* Γ *with* $|u_i| = n$.

We give two examples.

Example 4.13 We consider the membrane system

$$\Gamma_1 = (\{a, b, c\}, [_1[_2]_2]_1, a^2, \lambda, R_1, \emptyset, 2)$$

with

$$R_1 = \{a \to (a, here)(b, in_2)(c, in_2)^2,\ a^2 \to (a, out)^2\}.$$

Since, initially, we have two letters a in the membrane 1, we have two possibilities: we apply two times the rule $a \to (a, here)(b, in_2)(c, in_2)^2$ or we apply once the rule $a^2 \to (a, out)^2$. In the latter case both letters a are send in the environment and are lost such that the derivation stops since no further letters are in membrane 1. In the former case, two letters a remain in membrane 1 and two letter b and four letters c are send inside membrane 2. If we apply n times $a \to (a, here)(b, in_2)(c, in_2)^2$ and finish by one application of $a^2 \to (a, out)^2$, then we have finally $2n$ letters b and $4n$ letters c in membrane 2. Hence

$$L(\Gamma_1) = \{6n \mid n \geq 0\}.$$

Example 4.14 Let

$$\Gamma_2 = (\{A, B, D, E, X, Y, Z, a, \#\}, [_1[_2]_2]_1, XADE, \lambda, R_1, \emptyset, 2)$$

be a membrane systems with two membranes where

$$\begin{aligned} R_1 = \{ & XADE \to XBBDE,\ XE \to YE,\ AD \to \#, \# \to \#, \\ & YBDE \to YADE,\ YD \to YD,\ BE \to \#, \\ & YD \to Z,\ ZA \to Z(a, in_2)\ \} \end{aligned}$$

We note that any application of a rule requires an occurrence of X or Y or Z. the initial configuration contains one such letter, namely X, and each rule produces at most one such letter. Therefore only one such letter occurs in any configuration (and as we see below, hence we can only apply one rule of R_1 in each step). Furthermore, if the letter # is introduced by some rule, then we can apply the rule $\# \to \#$ at every moment and thus the system cannot reach a halting configuration, i.e., no word of $L(\Gamma_2)$ can be generated.

Let a configuration (XA^nDE, λ) with $n \geq 1$ be given: note that the initial configuration is given by $n = 1$. Then we cannot apply $XE \to YE$ since we also have to apply $AD \to \#$ by the maximal parallelism, which introduces #. This holds as long A is present in the first component of the configuration. Hence we get

$$(XA^nDE, \lambda) \vdash (XA^{n-1}B^2DE, \lambda) \vdash (XA^{n-2}B^4DE, \lambda) \vdash \ldots \vdash (XB^{2n}DE, \lambda).$$

Now we can use $XE \to YE$ (and only this rule is applicable) since it cannot be accompanied by $DA \to \#$. Thus we have $(YB^{2n}DE, \lambda)$. By arguments as above we have to replace all occurrences of B by A using the rule $YBDE \to YADE$. This yields $(YA^{2n}DE, \lambda)$. Now we have two cases for the continuation.

Case 1. We apply $YD \to XD$. Then we obtain the configuration $(XA^{2n}DE, \lambda)$ which has the form as the configuration from which we started and the process of doubling the A's can be iterated.

Case 2. We apply $YD \to Z$. We get $(ZA^{2n}E, \lambda)$. In this configuration only $ZA \to Z(a, in_2)$ is applicable. Thus we obtain

$$(ZA^{2n}E, \lambda) \vdash (ZA^{2n-1}E, a) \vdash (ZA^{2n-2}E, a^2) \vdash \ldots \vdash (ZE, a^{2n}).$$

The last configuration is a halting one and therefore a^{2n} belongs to $L(\Gamma_2)$. Therefore

$$L(\Gamma_2) = \{2^n \mid n \geq 1\}.$$

We ask the reader to note that the membrane systems Γ_2 works as the matrix grammar G_2. In both cases the introduction of # leads to a non-terminating derivation or only to non-halting configurations, and it is necessary to replace all A's or all B's, before X can be changed to Y or Y to X or Z, respectively.

We say that a rule is
– *non-cooperating* if it has the form $a \to w$ with $a \in V$ and $w \in (V \times Tar)^*$,
– *cooperating* if it has the form $u \to w$ with $u \in V^*$, $|u| \geq 2$, and $w \in (V \times Tar)^*$.
These notions non-cooperating and cooperating correspond to context-free and context-sensitive in usual grammars. However since in a membrane system the word are interpreted as multisets we have no context in membrane systems and therefore we have only a cooperation between the letters in a multiset if the multiset is replaced.

A letter $c \in V$ is called a *catalyst* iff all rules where c occurs have the form $ca \to cw$ with $a \in V$ and $w \in (V \times Tar)^*$, i.e., the catalyst is not changed by the reaction, however, it is necessary that a can perform the change to w. We say that $ca \to cw$ is a *catalytic rule.* Obviously, catalyst rules are a special case of cooperating rules.

We say that a membrane system is
– *non-cooperating* if all its rules are non-cooperating and
– *catalytic* if all its rules are non-cooperating or catalytic.
Otherwise, the membrane system is called *cooperating.*

By $\mathcal{L}_n(P, nco)$, $\mathcal{L}_n(P, cat)$, and $\mathcal{L}_n(P, coo)$ we denote the families of languages which can be generated by non-cooperating, catalytic, and cooperating membrane systems with at most n membranes, respectively. For $X \in \{nco, cat, coo\}$,

$$\mathcal{L}_*(P, X) = \bigcup_{n \geq 1} \mathcal{L}_n(P, X).$$

By definition, for $X \in \{nco, cat, coo\}$, we have

$$\mathcal{L}_1(P, X) \subseteq \mathcal{L}_2(P, X) \subseteq \mathcal{L}_3(P, X) \subseteq \ldots \subseteq \mathcal{L}_n(P, X) \subseteq \ldots \subseteq \mathcal{L}_*(P, X). \tag{4.1}$$

We first prove that the hierarchies given in (4.1) is finite for all X under consideration and has at most two levels.

Lemma 4.15 *For $X \in \{nco, cat, coo\}$ and $n \geq 2$, $\mathcal{L}_1(P, X) \subseteq \mathcal{L}_2(P, X) = \mathcal{L}_n(P, X) = \mathcal{L}_*(P, X)$.*

Proof. Obviously, by (4.1) it is sufficient to prove that $\mathcal{L}_*(P, X) \subseteq \mathcal{L}_2(P, X)$.

The idea of the proof consist in an indexing of letters in such a way that the index gives the membrane in which the letter is. Thus we set

$$V' = \{a_j \mid a \in V, 1 \leq j \leq m, j \neq i\}$$

and define for $1 \leq j \leq m$, $j \neq i$, the morphisms $h_j : V \to V'$ by $h(a) = a_j$.

Let $L \in \mathcal{L}_*(P, X)$. Then $L = L(\Gamma)$ for some membrane system Γ. Let

$$\Gamma = (V, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m, i)$$

with $m \geq 3$ (if $m \leq 2$, then $L \in \mathcal{L}_2(P, X)$ by definition). We construct the membrane system

$$\Gamma' = (V' \cup V, [_1[_i]_i]_1, w'_1, w_i, R'_1, R'_i, i)$$

with

$$w'_1 = h_1(w_1)h_2(w_2)\ldots h_{i-1}(w_{i-1})h_{i+1}(w_{i+1})h_{i+2}(w_{i+2})\ldots h_m(w_m)$$

and R'_1 and R'_i consisting of all rules which are constructed in the following way:

- If $u \to (b_1, t_1)(b_2, t_2)\ldots(b_s, t_s) \in R_k$ with $1 \leq k \leq m$, $k \neq i$, then $h_k(u) \to c_1c_2\ldots c_s \in R'_1$ where
  – $c_r = ((b_r)_k, here)$ if $t_r = here$
  – $c_r = ((b_r)_p, here)$ if $t_r = in_p$ and $p \neq i$,
  – $c_r = (b_r, in_i)$ if $t_r = in_i$,
  – $c_r = ((b_r)_l, here)$ if $t_r = out$ and l is the unique membrane which contains membrane k in μ.

- If $u \to (b_1, t_1)(b_2, t_2)\ldots(b_s, t_s) \in R_i$ with $1 \leq k \leq m$, $k \neq i$, then $h_k(u) \to c_1c_2\ldots c_s \in R'_i$ where
  – $c_r = (b_r, here)$ if $t_r = here$
  – $c_r = ((b_r)_{l'}, out)$ if $t_r = out$ and l' is the unique membrane which contains membrane i in μ.

By these definitions,

$$(v_1, v_2, \ldots, v_m) \vdash (v'_1, v'_2, \ldots, v'_m)$$

in Γ if and only if

$$(h_1(v_1)\ldots h_{i-1}(v_{i-1})h_{i+1}(v_{i+1})\ldots h_m(v_m), v_i) \vdash (h_1(v'_1)\ldots h_{i-1}(v'_{i-1})h_{i+1}(v'_{i+1})\ldots h_m(v'_m), v'_i)$$

in Γ'. Moreover, we have that $(v_1, v_2, \ldots, v_m)$ is a halting configuration of Γ if and only if $(h_1(v_1)h_2(v_2)\ldots h_{i-1}(v_{i-1})h_{i+1}(v_{i+1})\ldots h_m(v_m), v_i)$ is a halting configuration of Γ'. Therefore the membrane i contains the same multisets if a halting configuration is obtained. Thus $L(\Gamma) = L(\Gamma')$. This implies $L = L(\Gamma') \in \mathcal{L}_2(P, X)$. $\square$

We now prove that Lemma 4.15 can be improved for *nco* and *coo* to $n \geq 1$. Moreover, we characterize $\mathcal{L}_*(P, nco)$ and $\mathcal{L}_*(P, coo)$.

Theorem 4.16 *For all $n \geq 1$, $\mathcal{L}_1(P, nco) = \mathcal{L}_n(P, nco) = \mathcal{L}_*(P, nco) = N(CF)$.*

Proof. By (4.1) and Lemma 4.15, it is sufficient to prove that $N(CF) \subseteq \mathcal{L}_1(P, nco)$ and $\mathcal{L}_2(P, nco) \subseteq N(CF)$.

Let $L \in N(CF)$. Then there is a context-free language L' such that $L = N(L')$. Let G be a context-free grammar generating L'. We construct the membran system $\Gamma = (N \cup T, [_1]_1, S, P, 1)$. Note that the rules of P in Γ are a short writing of rules where the target is *here* in all cases. It is obvious that a derivation $S \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow \ldots \Longrightarrow w_n$ in G corresponds to $(S) \vdash (w_1) \vdash (w_2) \vdash \ldots \vdash (w_n)$ in Γ (any configuration has only one component). Moreover, $z \in L(G)$ iff $z \in T^*$ iff no rule is applicable in G iff (z) is a halting configuration. Hence $L(\Gamma) = N(L(G)) = N(L') = N$. This proves $N(CF) \subseteq \mathcal{L}_1(P, nco)$.

Let $L = L(\Gamma)$ for some membrane system with 2 membranes, i.e.,

$$\Gamma = (V, [_1[_2]_2]_1, w_1, w_2, R_1, R_2, 2).$$

Without loss of generality we assume that w_1 contains no letter of F_1 since such letters cannot be changed by Γ, and therefore they are superfluous for $L(\Gamma)$. For $1 \leq i \leq 2$, we define F_i as the set of all $a \in V$ such that there is no rule with left-hand side a in R_i,

$$V_i = \{a_i \mid a \in V\} \text{ and } V'_i = \{a'_i \mid a \in V\},$$

the homomorphisms

$$h_i : V \to V'_i,\ g_1 : V \times \{here, out, in_2\} \to F_2 \cup V'_1 \cup V'_2 \text{ and } g_2 : V \times \{here, out\} \to F_2 \cup V'_1 \cup V'_2$$

by

$$\begin{aligned}
h_i(a) &= a'_i, \\
g_1((b, here)) &= \begin{cases} \lambda & \text{if } b \in F_1 \\ b'_1 & \text{otherwise,} \end{cases} \\
g_1((b, out) &= \lambda, \\
g_1((b, in_2)) &= \begin{cases} b & \text{if } b \in F_2 \\ b'_2 & \text{otherwise,} \end{cases}
\end{aligned}$$

$$g_2((b, here)) = \begin{cases} b & \text{if } b \in F_2 \\ b'_2 & \text{otherwise,} \end{cases}$$

$$g_2((b, here)) = \begin{cases} \lambda & \text{if } b \in F_1 \\ b'_1 & \text{otherwise,} \end{cases}$$

and the grammar system $G = (N, V \setminus F_2, P_1, P_2, S)$ with two components by

$$\begin{aligned} N &= V_1 \cup V_2 \cup V'_1 \cup V'_2 \\ P_1 &= \{S \to h_1(w_1)h_2(w_2)\} \cup \{a_i \to g_i(x) \mid a \to x \in R_i, 1 \le i \le 2\}, \\ P_2 &= \{a'_i \to a_i \mid a \in V, 1 \le i \le 2\}. \end{aligned}$$

A configuration (w_1v, w_2u) with $w_1 \in (V \setminus F_1)^*$, $v \in F_1{*}$, $w_2 \in (V \setminus F_2)^*$ and $u \in F_2^*$ is described in the grammar system G by a word $h_1(w_1)h_2(w_2)u$. Such a word cannot be processed by the first component of G and the second component of G cancels all the primes, i.e., we obtain the word v_1v_2 where v_1 is the variant of w_1 where all letters have the index 1 and v_2 is the variant of w_2 where all letters have the index 2. The first component of G transforms a word v_1v_2 with $v_1 \in V_1^*$ and $v_2 \in V_2^*$ in u_1u_2 where u_1 and u_2 are the indexed and primed versions of w'_1 and w'_2 with $(w_1, w_2u) \vdash (w'_1, w'_2u)$ besides the letters of F_1 which are cancelled since they do not contribute to Γ and the letters of F_2 which remain in the second membrane. Therefore there are words $z_1 \in (V \setminus F_1)^*$, $z \in F_1{*}$, $z_2 \in (V \setminus F_2)^*$ and $u' \in F_2^*$ such that $w'_1 = z_1z$, $w_2 = z_2u'$ and

$$h(w_1)h(w_2)u \Longrightarrow_{P_1} v_1v_2 \Longrightarrow_{P_2} h_1(z_1)h_2(z_2)u'u$$

in G. Moreover, the derivation stops in G if and only if all letters belong to F_2, and a halting configuration in Γ is obtained if and only if all letters in membrane 1 belong to F_2 and all letters in membrane 2 belong to F_2. Taking into consideration that the letters of F_1 are cancelled in G, we obtain that $L(\Gamma) = N(L(G))$. By Theorem 4.10 i), $L(G)$ is a context-free language. Hence $N(L(G)) \in N(CF)$. Therefore we have $L(\Gamma) \in N(CF)$ and $\mathcal{L}_2(P, nco) \subseteq N(CF)$ is shown. □

Theorem 4.17 *For all $n \ge 1$, $\mathcal{L}_1(P, coo) = \mathcal{L}_n(P, coo) = \mathcal{L}_*(P, coo) = N(RE)$.*

Proof. By (4.1) it is sufficient to prove that $N(RE) \subseteq \mathcal{L}_1(P, coo)$.

Let $L \in N(RE)$. By Theorem 4.4, there is a matrix grammar $G = (N, T, M, S, F)$ such that $L = N(L(G))$. By Theorem 4.6, we can assume that G is in normal form. We construct the membrane system

$$\Gamma = (N_1 \cup N_2 \cup T \cup \{S, Z, \#, H, H', H''\} \cup \{H_A \mid A \in N_2\}, [_1]_1, S, R_1, 1)$$

with R_1 consisting of all rules of the forms

(1) $S \to HXA$ for $(S \to XA) \in M$,

(2) $HXA \to HYx$ for $(X \to Y, A \to x) \in M$,

(3) $HX \to H'H_AY,\ H_AA \to \#,\ \# \to \#, H' \to H'',\ H''H_A \to H$
for $(X \to Y, A \to \#) \in M$,

(4) $HZ \to \lambda$

Obviously, we have $S \Longrightarrow XA$ in G and $(S) \vdash (HXA)$ in Γ, i.e., besides the additional symbol H we have simulated a derivation step of G.

If we have a sentential form $w = Xw_1Aw_2$ in G, then we can apply a matrix of the form $(X \to Y, A \to x)$ and obtain Yw_1xw_2. In Γ we simulate this by applying $HXA \to HYx$ to HXw_1Aw_2 which gives HYw_1xw_2, i.e., the simulation is correct.

The matrix $(X \to y, A \to \#)$ is only applicable to Xw if A does not occur in w and results in Yw. Accordingly, if we apply $HX \to H'H_AY$ to Xw, we get $H'H_AYw$. If A is present, i.e. $w = w_1Aw_2$, we have to apply $H' \to H''$ and $H_AA \to \#$ in parallel (maximal parallelism) and get $H''\#w_1w_2$. However, now $\# \to \#$ can be applied at any moment and thus we cannot come to a halting configuration. If A is not present, we get

$$H'H_AYw \vdash H''H_AYw \vdash HYw,$$

i.e., again, the application of $(X \to y, A \to \#)$ is correctly simulated by the rules of (3).

If $Z \to \lambda$ is used in G we simulate this by $HZ \to lambda$.

By these explanations it follows that $L(\Gamma) = N(L(G)) = L$ and thus $L \in \mathcal{L}_1(P, coo)$ which proves $N(RE) \subseteq \mathcal{L}_1(P, coo)$.

□

For catalytic systems the situation is different. However, before we present the result on the power with respect to language generation we change the definition slightly and obtain a possibility to calculate by membrane systems.

Definition 4.18 *Let $\Gamma = (V, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m, i)$ be a membrane system with m membranes and $o \in V$ be a distinguished element such that, for any natural number n, the language of the membrane system*

$$\Gamma_x == (V, \mu, w_1o^n, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m, i)$$

is a singleton $\{f(x)\}$. Then we say that Γ computes the function $f : \mathbf{N} \to \mathbf{N}$.

We show that any partial recursive function can be computed by membrane systems.

Theorem 4.19 *For any partial recursive function $f : \mathbf{N} \to \mathbf{N}$, there is a catalytic membrane system*

$$\Gamma = (V, [_1[_2]_2]_1, w, \lambda, R_1, R_2, 2)$$

with two membranes which computes f.

Proof. The proof consists in the simulation of a register machine. We assume that the reader is familiar with the notions and results on register machines. Especially we need two facts:

- For any partial recursive function f, there exist a register machine which computes f.
- For any register machine, there exists an equivalent register machine (i.e., both machines compute the same function) which has only three types of commands:

  – $i\,:\,a+$ which adds one to the contents of register a and the computation is continued by command $i+1$,

  – $i\,:\,a-(k)$ which subtracts one from the register a if it is not empty and continues with command $i+1$ or it continues with command k without a change of the registers if register a is empty,

  – $Halt$ which ends the computation.

- The command $Halt$ is only used as the last command.
- The result is the contents of a distinguished register where no subtraction is done.

Let f be a partial recursive function. Then there is a register machine M which computes f. Let M have m registers $r_1, r_2, \ldots, r_m$ which are used in the computation and n commands $i_1, i_2, \ldots, i_n$, $i_n = Halt$, and r_m be the distinguished register for the result.

We set

$$\begin{aligned}
\Gamma &= (V, [_1[_2]_2]_1, w, \lambda, R_1, \emptyset, 2),\\
V &= \{o_m, r, r', \#\} \cup \bigcup_{i=0}^{m-1} \{o_i, o_i', c_i, c_i'\} \cup \bigcup_{j=1}^{n} \{p_j, p_j', p_j''\},\\
w &= (r')^{m-1} p_1' o_0 o_0' c_0 c_0' c_1 c_1' \ldots c_{m-1} c_{m-1}'.
\end{aligned}$$

The elements c_i and c_i', $0 \le i \le m-1$, will be the catalysts of the system. The elements o_i and o_i', $1 \le i \le m$, are used to represent the contents of the register r_i, o_m is used for the contents of the register r_m. The additional elements o_0 and o_0' will be used in the simulation of the halting command. The elements p_j, p_j', p_j'', $1 \le j \le n$, are used for the counting of commands.

The rules of R_1 are all rules of the following form:

$$r \to \#,\ r' \to \#,\ \# \to \#, \text{ and } c_i r \to c_i r',\ c_i' r' \to c_i' r \text{ for } 1 \le i \le m-1$$

(in order to avoid the introduction of # in a configuration which does not allow a halting the rules $c_i'r' \to c_i'r$ and $c_i r \to c_i r'$ have to be used for $1 \le i \le m-1$ in order to rewrite all symbols of r^{m-1} or $(r')^{m-1}$; moreover, we have alternately $(r')^{m-1}$ and r^{m-1} in the beginning of the word representing the contents in membrane 1; thus we can speak that there are odd steps which replace $(r')^{m-1}$ and even steps which replace r^{m-1})

$$(*)\ c_i o_i \to c_i o_i',\ c_i' o_i' \to c_i' o_i, \emptyset_i' \to \#, c_i o_i \to c_i' \text{ for } 1 \le i \le m-1$$

(by these rules the remaining catalysts are used; the latter two rules will be used in the case of decrementation),

$$p_1' \to p_1$$

(by this rule in the first step the first command i_1 is announced to be applied)

$$\begin{aligned}
&p_j \to p_{j+1}' o_a,\ p_{j+1}' \to p_{j+1} \text{ for } i_j = a+,\ a < m,\\
&p_j \to p_{j+1}'(o_a, in_2),\ p_{j+1}' \to p_{j+1} \text{ for } i_j = a+,\ a < m
\end{aligned}$$

(we simulate the command $a+$ by generating an additional element o_a which is sent into membrane 2 if the incrementation concerns the distinguished register r_m; we use two steps in order to come back to an element p_{j+1} which is the next command to be simulated),

$$\begin{array}{l} c_a p_j \to c_a p''_{j+1},\ p''_{j+1} \to p'_{j+1},\ P'_{j+1} \to p_{j+1},\ c'_a p_j \to c'_a p'_k,\ p'_k \to p_k \\ \qquad \text{for } i_j = a-(k) \end{array}$$

(if the register a is empty, we have to apply $c'_a p_j \to c'_a p'_k$ and $p'_k \to p_k$ in the even and odd step, i.e., the configuration is changed as follows (where we only give in detail the part of word which is essential) $(wc'_a p_j, z) \vdash (w'c'_a p'_k, z) \vdash (wc'_a p_k, z)$; if a is non-empty, then o'_a is present and we have to avoid the application of $o'_a \to \#$ from (*), hence $c'_a o'_a \to c'_a$ has to be used, which yields

$$(wc_a c'_a o'_a p_j, z) \vdash (w'c'_a p_j, z) \vdash (w''c_a c'_a p''_{j+1}, z) \vdash (w'''c_a c'_a p'_{j+1}, z) \vdash (wc_a c'_a p_{j+1}, z);$$

thus in both cases the command $a-(k)$ is correctly simulated),

$$(^{**})\ c_0 p_n \to c_0,\ c'_0 r \to c_0,\ c'_0 r' c_0$$

(if the halting command i_n is reach, we get only considering the interesting part

$$c_0 c'_0 o_0 o'_0 p_n \vdash c_0 c'_0 o_0 o_0 \vdash c_0 c'_0 o'_0 \vdash c_0 c'_0 o_0 \vdash c_0 c'_0$$

by the use of $c_0 p_n \to c_0$ and rules of (*); now we cancel all occurrences of r and r' by the latter two rules of (**) and then all occurrences of o_i and o'_i by rules of (*). After these cancellation we get a halting configuration since any rule requires the presence of at least one of the symbols p_j, p'_j, p''_j, $1 \le j \le n$ or o_i, o'_i, $0 \le i \le m-1$ or r, r' which all are not existing anymore). By these explanation it is easy to see that Γ reaches a halting configuration with $f(x)$ symbols o_m in the second membrane if we start with wo_1^x. Therefore Γ computes f. □

Theorem 4.20 *For all $n \ge 2$, $\mathcal{L}_1(P, cat) \subset \mathcal{L}_2(P, cat) = \mathcal{L}_n(P, cat) = \mathcal{L}_*(P, cat) = N(RE)$.*

Proof. We omit the proof of $\mathcal{L}_1(P, cat) \subset \mathcal{L}_2(P, cat)$.

Let L be a recursively enumerable subset of $\mathbf{N}_0$. It is known that there is a partial recursive function f such that L is the range of f. Let Γ be the membrane system given in the proof of Theorem 4.19 which computes f. We consider

$$\Gamma' = (V', [_1[_2]_2]_1, w', \lambda, R'_1, \emptyset, 2)$$

with

$$\begin{aligned} V' &= V \cup \{p_0, p'_0\}, \\ w' &= r^{m-1} p_0 o_0 o'_0 c_0 c'_0 c_1 c'_1 \ldots c_{m-1} c'_{m-1}, \\ R'_1 &= R_1 \cup \{p_0 \to p'_0 o_1,\ p'_0 \to p_0,\ p_0 \to p'_1\}, \end{aligned}$$

where V and R_1 are taken from Γ.

It is easy to see that first some elements o_1 are introduced, say o_1^x, and then $p_0 \to p_1'$ is used which gives (wo_1^x, λ) with w from Γ. From here we get $o_m^{f(x)}$ in the second membrane if we reach the unique halting configuration. Consequently, $L(\Gamma')$ is the range of f and thus $L(\Gamma') = L$ which proves $L \in \mathcal{L}_2(P, cat)$ and therefore $N(RE) \subseteq \mathcal{L}_2(P, cat)$. From (4.1 we get $\mathcal{L}_2(P, cat) = \mathcal{L}_n(P, cat) = \mathcal{L}_*(P, cat)$. $\square$

For completeness we remark that the number of catalysts (which was two times the number of registers in the proof of Theorems 4.19 and 4.20) can be decreased to 2 and it is open whether one catalyst is sufficient.

4.3 Membrane Systems with Symport/Antiport Rules

In this section we discuss membrane systems without the ability of changing objects. Hence only the moving through the membranes can be used for computation. Thus we have a process which only works by the exchange of information. Hence the study of these systems is also of interest from an information-theoretic point of view, since it is investigated the power of communication.

In biology it is known that there are many cases where two chemicals pass through a membrane at the same time with the help of each other. Both chemicals go in the same direction (this is called symport) or in opposite direction (called antiport). Formally, such movements can be written as (ab, in) or (ab, out) in symport case where both chemicals come in or leave out a membrane, respectively, or as $(b, out; a, in)$ denoting that a comes in and b leaves a given membrane.

Obviously, if one considers membrane systems where any rule is a symport or antiport rule, then no change of the involved chemicals occurs, and therefore finitely many objects are only moving around, which gives only a strongly limited power. Therefore we add symbols in the environment and assume that an infinite number of copies of each of these symbols is present in the environment.

We now give the formal definition of a membrane system with symport/antiport rules.

Definition 4.21 *i) A membrane system with m membranes and symport/antiport rules is a construct*

$$\Gamma = (V, \mu, E, w_1, w_2, \dots, w_m, R_1, R_2, \dots R_m, i)$$

where V, μ, $w_1, w_2, \dots w_m$, $R_1, R_2, \dots, R_m$ and i are specified as in membrane system, E is a subset of V and, for $1 \leq j \leq m$, R_j is a finite set of rules of the form (x, in) or (x, out) or $(x, out; , y, in)$ with $x, y \in V^+$.

ii) A configuration of a membrane system with symport/antiport rules is a m-tuple $C = (u_1, u_2, \dots, u_m)$ of words (or equivalently, multisets) over V.

Let j, $1 \leq j \leq m$, be a membrane and let j' be the unique membrane which contains membrane j. The application of a rule (x, in) of R_j to C results in taking the multiset x out $c_{j'}$ and adding to c_j; the application of (x, out) is performed by subtracting x from c_j and adding to $c_{j'}$; the application of $(x, out; y, in)$ consists in a parallel application of (x, out) and $y, in)$ as described. If j is the outer membrane, then E takes the rule of membrane j' where any element of E is present in E infinitely often.

The transformation of a configuration C into a configuration C' (written as $C \vdash C'$) is done by a maximal parallel application of the rules of all R_j, $1 \leq j \leq m$, to C.

A configuration C is called halting if no rules from the sets R_j, $1 \leq j \leq m$, can be applied to C

iii) The language $L(\Gamma)$ generated by a membrane system Γ with symport/antiport rules is the set of all numbers n such that there is a halting configuration $C = (u_1, u_2, \ldots, u_m)$ of Γ with $|u_i| = n$.

Example 4.22 We consider the membrane system

$$\Gamma = (V, [_1[_2]_2]_1, E, ac, df, R_1, R_2, 2)$$

with

$$\begin{aligned}
V &= \{a, b, c, c', d, e, e', f, g, \#\}, \\
E &= \{a, b, c, c', e, e', f, g, \#\}, \\
R_1 &= \{(c, out; \#, in),\ (ca, out; cbb, in),\ (ca, out, c'bb, in),\ (da, out; \#, in) \\
&\quad (c'd, out; e, in),\ (eb, out; ea, in),\ (eb, out; e'a, in),\ (fb, out; \#, in), \\
&\quad (e'f, out; cdf, in),\ (e'f, out; g, in)\}, \\
R_2 &= \{(d, out; c', in),\ (c', out),\ (f, out; e', in),\ (e', out),\ (df, in), \\
&\quad (\#, in),\ (\#, out),\ (ga, in),\ (g, out)\}.
\end{aligned}$$

First we mention that the introduction of $\#$ is forbidden, again, because by the rules $(\#, out)$ and $(\#, in)$ in R_2 the symbol $\#$ can alternately moved from membrane 2 to membrane 1 and conversely such that no halting configuration can be reached. Therefore we have the following sequence of configurations (note that the case $n = 1$ is given initially)

$$\begin{array}{rcll}
(ca^n, df) & \vdash & (cbba^{n-1}, df) & \text{by } (ca, out; cbb, in) \in R_1 \\
& \vdash & (cb^4a^{n-2}, df) & \text{by } (ca, out; cbb, in) \in R_1 \\
& & \vdots & \\
& \vdash & (cb^{2n-2}a, df) & \text{by } (ca, out; cbb, in) \in R_1 \\
& \vdash & (c'b^{2n}, df) & \text{by } (ca, out; c'bb, in) \in R_1 \\
& \vdash & (db^{2n}, c'f) & \text{by } (d, out; c', in) \in R_2 \\
& \vdash & (dc'b^{2n}, f) & \text{by } (c', out) \in R_2 \\
& \vdash & (eb^{2n}, f) & \text{by } (dc', out; e, in) \in R_1 \\
& \vdash & (eab^{2n-1}, f) & \text{by } (eb, out; ea, in) \in R_1 \\
& \vdash & (ea^2b^{2n-2}, f) & \text{by } (eb, out; ea, in) \in R_1 \\
& & \vdots & \\
& \vdash & (ea^{2n-1}b, f) & \text{by } (eb, out; ea, in) \in R_1 \\
& \vdash & (e'a^{2n}, f) & \text{by } (eb, out; e'a, in) \in R_1 \\
& \vdash & (fa^{2n}, e') & \text{by } (f, out; e', in) \in R_2 \\
& \vdash & (e'fa^{2n}, \lambda) & \text{by } (e', out) \in R_1.
\end{array}$$

Now we have two possibilities of continuation:

$$\begin{array}{rcll}
(e'fa^{2n}, \lambda) & \vdash & (cdfa^{2n}, \lambda) & \text{by } (e'f, out; cdf, in) \in R_1 \\
& \vdash & (cbba^{2n-1}, df) & \text{by } (ca, out; cbb, in) \in R_1,\ (df, in) \in R_2
\end{array}$$

which means that, essentially, we have doubled the number of occurrences of a in membrane 1 and can iterate this process, or

$$\begin{array}{lcll}
(e'fa^{2n}, \lambda) & \vdash & (ga^{2n}, \lambda) & \text{by } (e'f, out; g, in) \in R_1 \\
& \vdash & (a^{2n-1}, ga) & \text{by } (ga, in) \in R_2 \\
& \vdash & (ga^{2n-1}, a) & \text{by } (g, out) \in R_2 \\
& \vdash & (a^{2n-2}, ga^2) & \text{by } (ga, in) \in R_2 \\
& \vdash & (ga^{2n-2}, a^2) & \text{by } (g, out) \in R_2 \\
& & \vdots & \\
& \vdash & (\lambda, ga^{2n}) & \text{by } (ga, in) \in R_2 \\
& \vdash & (g, a^{2n}) & \text{by } (g, out) \in R_2
\end{array}$$

and a halting configuration is obtained. Therefore

$$L(\Gamma) = \{2^n \mid n \geq 1\}.$$

We now prove that membrane systems with symport/antiport rules, i.e., membrane systems which only work on the basis of communication, are able to generate all recursively enumerable sets of numbers.

Theorem 4.23 *For any set $L \in N(RE)$, there is a membrane system Γ with symport/antiport rules such that $L(\Gamma) = L$.*

Proof. By Theorems 4.4 and 4.6 there is a matrix grammar $G = (N, T, M, S, F)$ in normal form such that $L = N(L(G))$. Let $(S \to X'A')$ be the only matrix in G of this form. Let M have n matrices of the form $(X \to Y, A \to x)$ or $(X \to Y, A \to \#)$.

We define the membrane system

$$\Gamma = (V, [_1[_2]_2]_1, E, cX'A', \lambda, R_1, R_2, 2)$$

with

$$\begin{array}{rcl}
V & = & N_1 \cup N_2 \cup T \cup \{c, g, h, Z, \#\} \cup \bigcup_{i=1}^{n} \{c_i, c_i', d_i\}, \\
E & = & N_1 \cup N_2 \cup T \cup \{g, h, Z, \#\} \cup \bigcup_{i=1}^{n} \{c_i, c_i', d_i\}, \\
Q_i & = & \{(cX, out; c_iY),\ (c_iA, out; cc_i', in),\ (c_i', out; x, in),\ (c_i, out; \#, in)\} \\
& & \qquad \text{for } m_i = (X \to Y, A \to x), \\
Q_i & = & \{(cX, out; c_id_i, in),\ (d_i, out; Yh, in),\ (c_iA, out; \#, in),\ (h, out; cg, in),\ (c_ig, out)\} \\
& & \qquad \text{for } m_i = (X \to Y, A \to \#), \\
R_1 & = & \{(c, out; \#, in),\ (cZ, out)\} \cup \bigcup_{i=1}^{n} Q_i, \\
R_2 & = & \{(\#, in),\ (\#, out)\} \cup \{(a, in) \mid a \in T\}.
\end{array}$$

We note, again, that introducing the symbol # does not allow reaching of a halting configuration since it can be move from the second membrane to the first membrane or conversely at every moment.

Moreover, the second membrane only collects the terminals occurring at some moment in the first membrane.

Therefore we now consider only the first component of a configuration. If it has the form $cXAw$ (as it is the case for the initial configuration), then we can without introducing # only perform the following steps

$$cXAw \vdash c_iYAw \vdash cc_i'Yw \vdash cYxw$$

which means that we have correctly applied the simulation of the matrix $m_i = (X \to Y, A \to x)$ and can proceed with the simulation of a further matrix.

If the configuration is cXw and A does not occur in w, then the following steps have to be done

$$cXw \vdash c_id_iw \vdash c_iYhw \vdash c_iYcgw \vdash cYw$$

which is a correct simulation of the application of $(X \to Y, A \to \#)$.

Moreover, in both cases we stop no nonterminal is present in the sentential form or in the first membrane (since the configuration (cZw, x) is only reachable if $w =$). Therefore $L(\Gamma) = N(L(G)) = L$. $\square$