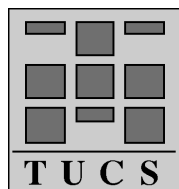


# Computing with Membranes

**Gheorghe Păun**

Institute of Mathematics  
of the Romanian Academy  
PO Box 1-764  
70700 București, Romania  
E-mail: gpaun@imar.ro



**Turku Centre for Computer Science**  
**TUCS Technical Report No 208**  
**November 1998**  
**ISBN 952-12-0303-X**  
**ISSN 1239-1891**

## Abstract

We introduce a new computability model, of a distributed parallel type, based on the notion of a *membrane structure*. Such a structure consists of several cell-like membranes, recurrently placed inside a unique “skin” membrane. A plane representation is a Venn diagram without intersected sets and with a unique superset. In the regions delimited by the membranes there are placed objects; the obtained construct is called a *super-cell*. These objects are assumed to evolve: each object can be transformed in other objects, can pass through a membrane, or can dissolve the membrane in which it is placed. A priority relation between evolution rules can be considered. The evolution is done in parallel for all objects able to evolve. In this way, we obtain a computing device (we call it a *super-cell system*): start with a certain number of objects in a certain membrane and let the system evolve; if it will halt (no object can further evolve), then the computation is finished, with the result given as the number of objects in a specified membrane. If the development of the system goes for ever, then the computation fails to have an output.

We prove that the super-cell systems with the possibility of objects to cooperate characterize the recursively enumerable sets of natural numbers; moreover, systems with only two membranes suffice. In fact, we do not need cooperation, but we only use *catalysts*, specified objects which are present in the evolution rules but are not modified by the rule application. One catalyst suffices.

A variant is also considered, with the objects being strings over a given alphabet. The evolution rules are now based on string transformations. We investigate the case when either the rewriting operation from Chomsky grammars (with respect to context-free productions) or the splicing operation from H systems investigated in the DNA computing is used. In both cases, characterizations of recursively enumerable languages are obtained by very simple super-cell systems: with three membranes in the rewriting case and four in the splicing case.

Several open problems and directions for further research are formulated.

**Keywords:** Membrane structure, Super-cell system, Recursively enumerable set, Matrix grammar, Splicing, Natural computing

**TUCS Research Group**  
Mathematical Structures of Computer Science

# 1 Introduction

The present paper can be considered as a contribution to what is called in the last years with the generic name of *natural computing*, a field of research which tries to imitate the nature in the way it “computes”, learning new computing models and computing paradigms experimented for billions of years by nature and implementing them in computations done *in vitro* (or, in many cases, *in info*, in symbolic terms only, maybe implemented in silicon media).

We start from the observation that any non-trivial biological system is a hierarchical construct, composed of several “organs” which are well defined and delimited from the neighboring organs, which evolve internally and also cooperate with the other organs in order to keep alive the system as a whole. In general, an intricate flow of materials and information underlies the functioning of a complex biological system. Specific to such systems is also the fact that they can contain in the same vicinity subsystems of rather different levels of complexity: individual cells coexist with complex organs, not to mention that the cells and the organs themselves can be very different, of various heights as hierarchical systems (convincing examples can be found everywhere around us).

At a more specific level with respect to the models we are going to define, important to us is the fact that the parts of a biological system are well delimited by various types of *membranes*, in the broad sense of the term, starting from the cell membrane, going to the skin of organisms, and ending with more or less virtual “membranes” which delimit, for instance, parts of an ecosystem. In very practical terms, in biology and chemistry one knows membranes which keep together certain chemicals and leave to pass other chemicals, in a selective manner, sometimes only in one direction. Membranes delimiting subsystems of a symbol manipulating system are also considered in the logical framework to the so-called metabolic systems, as defined in [11].

Another incentive of our work comes from distributed computing, where again rather different but well delimited computing units coexist and are hierarchically arranged in complex systems. From single small processors to the world wide web there is a long way and in all its components we can see aspects as mentioned above. The grammar systems theory mirrors in mathematical terms such distributed symbol processing systems (see, e.g, [3] and, for recent developments, [13]), and will have some resemblance with (some of) the computing models we consider here.

Starting from these observations, we first consider the notion of a *membrane structure*, as a mathematical counterpart of hierarchical architectures composed of membranes recurrently distinguished in a given main membrane. We will represent such a structure as a Venn diagram, with all the considered sets being subsets of a unique set and not allowed to be inter-

sected (two sets are either one the subset of the other, or disjoint).

The next step is to consider the notion of a *super-cell*, which is nothing else than a membrane structure with certain *objects* placed in the regions delimited by the membranes. The objects are identified by their “names” (mathematically, by symbols from a given alphabet). Because several copies of the same object can appear in the same region, we work with multisets, sets with multiplicities associated to their elements.

If the objects of a super-cell are able to evolve, then we obtain a computing device. We call it a *super-cell system*.

Thus, a super-cell system is a membrane structure with objects in its membranes, with specified evolution rules for objects, and with given input-output prescriptions. Any object, alone or together with one more object, evolves, can be transformed in other objects, can pass through one membrane, and can dissolve the membrane in which it is placed. All objects evolve at the same time, in parallel; in turn, all membranes are active in parallel. The evolution rules are hierarchized by a priority relation, given in the form of a partial order relation; always, the rule with the highest priority among the applicable rules is actually applied. If the objects evolve alone, the system is said to be non-cooperative; if there are rules which specify the evolutions of several objects at the same time, then the system is cooperative; an intermediate case is that where there are certain objects (we call them *catalysts*), specified in advance, which do not evolve alone, but appear together with other objects in evolution rules and they are not modified by the use of the rules.

The systems of this basic type are called *transition super-cell systems*, in order to distinguish this variant from other variants considered later.

It is somewhat surprising that with only these simple ingredients (and with designated input and output membranes), the cooperating super-cell systems and the systems with catalysts have computational completeness, they can characterize the recursively enumerable sets of natural numbers. Moreover, very simple membrane structures are enough: two membranes suffice. The input and the output of a computation are codified in the number of objects placed in certain input and output membranes, respectively.

An attractive feature of super-cell systems is their intrinsic parallelism. All objects having access to a rule should use that rule (with the restriction imposed by the priority relation). Moreover, all membranes work in parallel. The effect of this two level parallelism on the complexity of computations done by super-cell systems is not clarified.

The super-cell structure can be used as a support for a computing device based on any type of objects and any type of evolving rules associated with them. In the case above, the objects were single symbols (finitely many for each concrete system, taken from a denumerable alphabet). We can also take strings as objects. In this way, we can use an infinite set of objects, which can evolve in many ways, defined by string processing rules: rewriting, point

mutations, insertion and deletion, and so on and so forth. We consider here only two cases, when the strings evolve by rewriting (using context-free rules) and by splicing, the operation defined in [8] as a model of the recombinant behavior of DNA molecules under the influence of restriction enzymes. It is known that the splicing operation is powerful – see [12]. This observation is confirmed here: super-cell systems based on splicing characterize the family of recursively enumerable languages. Moreover, very simple systems are enough: we need only four membranes, arranged in a two level structure. Splicing super-cell systems with two membranes can generate non-regular languages, while three membranes are sufficient to generate non-context-free languages.

A characterization of recursively enumerable languages is also obtained in the case of super-cell systems based on rewriting. The proof uses the characterization of recursively enumerable languages by means of matrix grammars with appearance checking. This time, the number of used membranes is still smaller: three.

## 2 Some General Prerequisites

We here specify a few elementary notions and notations which will be useful in the subsequent sections.

We denote by  $\mathbf{N}$  the set of natural numbers.

Let  $U$  be an arbitrary set. A *multiset* (over  $U$ ) is a mapping  $M : U \rightarrow \mathbf{N}$ ;  $M(a)$ , for  $a \in U$ , is the *multiplicity of  $a$  in the multiset  $M$* . We indicate this fact also in the form  $(a, M(a))$ . (Of course, the multiplicity of each object with respect to any multiset is finite.) Note that for a usual set  $M \subseteq U$  we have  $M(a) = 1$  when  $a \in M$  and  $M(a) = 0$  otherwise (the set and its membership function are denoted in the same way). The *support* of a multiset  $M$  is the set  $\text{supp}(M) = \{a \in U \mid M(a) > 0\}$ . A multiset  $M$  is empty when its support is empty (it is then denoted by  $\emptyset$ ).

Let  $M_1, M_2 : U \rightarrow \mathbf{N}$  be two multisets. We say that  $M_1$  is included in  $M_2$  iff  $M_1(a) \leq M_2(a)$ , for all  $a \in U$ . The union of  $M_1$  and  $M_2$  is the multiset  $M_1 \cup M_2 : U \rightarrow \mathbf{N}$  defined by  $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$ , for all  $a \in U$ . The difference  $M_1 - M_2$  is here defined only when  $M_2$  is included in  $M_1$  and it is the multiset  $M_1 - M_2 : U \rightarrow \mathbf{N}$  given by  $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ , for all  $a \in U$ .

A multiset  $M$  with a finite support,  $\{(a_1, M(a_1)), (a_2, M(a_2)), \dots, (a_n, M(a_n))\}$ , can be also represented by a string:  $a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)}$  and all permutations of this string precisely identify the objects in the support of  $M$  and their multiplicities. We will frequently use below this more compact representation of multisets of a finite support.

An *alphabet* is a finite nonempty set of abstract *symbols*. Given an alphabet  $V$ , we denote by  $V^*$  the sets of all finite strings of elements in  $V$ ,

including the empty string,  $\lambda$ . (Thus,  $V^*$  is the free monoid generated by  $V$  with the operation of concatenation and the identity  $\lambda$ .) The length of a string  $x \in V^*$  is denoted by  $|x|$ . A set of strings (over an alphabet  $V$ ) is called a *language* (over  $V$ ).

For elements of formal language theory we will use here we refer to [16]. Details about L systems, regulated rewriting, grammar systems, and DNA computing can be found in [15], [6], [3], and [12], respectively. Some notions and notations which will be used only locally will be introduced when necessary.

### 3 Membrane Structures

We now introduce the basic structural ingredient of the computing devices we will define later: membrane structures.

Let us consider first the language  $MS$  over the alphabet  $\{[, ]\}$ , whose strings are recurrently defined as follows:

1.  $[ ] \in MS$ ;
2. if  $\mu_1, \dots, \mu_n \in MS, n \geq 1$ , then  $[\mu_1 \dots \mu_n] \in MS$ ;
3. nothing else is in  $MS$ .

Consider now the following relation on  $MS$ : for  $x, y \in MS$  we write  $x \sim y$  if and only if we can write the two strings in the form  $x = [ \dots [ \dots ] \dots ]$ ,  $y = [ \dots [ \dots ] \dots ]$  (two pairs of parentheses which are not one contained in the other are interchanged, together with their contents). We also denote by  $\sim$  the reflexive and transitive closure of the relation  $\sim$ . This is clearly an equivalence relation. We denote by  $\overline{MS}$  the set of equivalence classes of  $MS$  with respect to this relation. The elements of  $\overline{MS}$  are called *membrane structures*.

We stress the fact that when speaking of a membrane structure we do not take into account the order of the membrane structures which are used when a new membrane structure is constructed and that each membrane structure is bracketed by an external pair  $[ ]$  of parentheses.

It is easy to see that the parentheses  $[, ]$  appearing in a membrane structure are correctly matching, in the usual sense. Conversely, any string of correctly matching pairs of parentheses  $[, ]$ , with a matching pair at the ends, corresponds to a membrane structure. Therefore, we can write

$$MS = [D],$$

where  $D$  is the *Dyck language* over  $\{[, ]\}$ , that is, the language generated by the context-free grammar with the productions

$$S \rightarrow SS, S \rightarrow [S], S \rightarrow \lambda.$$

Each matching pair of parentheses  $[, ]$  appearing in a membrane structure is called a *membrane*. The number of membranes in a membrane structure  $\mu$  is called the *degree* of  $\mu$  and denoted by  $deg(\mu)$ . The external membrane of a membrane structure  $\mu$  is called the *skin* membrane of  $\mu$ . A membrane which appears in  $\mu \in \overline{MS}$  in the form  $[ ]$  (no other membrane appears inside the two parentheses) is called an *elementary* membrane.

The *depth* of a membrane structure  $\mu$ , denoted by  $dep(\mu)$ , is defined recurrently as follows:

1. if  $\mu = [ ]$ , then  $dep(\mu) = 1$ ;
2. if  $x = [\mu_1 \dots \mu_n]$ , for some  $\mu_1, \dots, \mu_n \in MS$ ,  
then  $dep(\mu) = \max\{dep(\mu_i) \mid 1 \leq i \leq n\} + 1$ .

A membrane structure can be represented in a natural way as a Venn diagram. This makes clear the fact that the order of membrane structures of the same level in a larger membrane structure is irrelevant; what matters is the topological structure, the relationships between membranes. In the subsequent sections we will make an extensive use of such a representation.

The Venn representation of a membrane structure  $\mu$  also makes clear the notion of a *region* in  $\mu$ : any closed space delimited by membranes is called a region of  $\mu$ . It is clear that a membrane structure of degree  $n$  contains  $n$  regions, one associated with each membrane.

## 4 Super-cells

We now make one more step towards the definition of a computing device, by adding objects to a membrane structure.

Let  $U$  be a denumerable set of objects. We call  $U$  the *universe* of our investigation and its elements are called *objects*.

Let  $\mu$  be a membrane structure of degree  $n, n \geq 1$ , with the membranes labeled in a one-to-one manner, for instance, with the numbers from 1 to  $n$ . In this way, also the regions of  $\mu$  are identified by the numbers from 1 to  $n$ . If a multiset  $M_i : U \longrightarrow \mathbf{N}$  is associated with each region  $i$  of  $\mu, 1 \leq i \leq n$ , then we say that we have a *super-cell*.

Any multiset  $M_i$  mentioned above can be empty. In particular, all of them can be empty, that is, any membrane structure is a super-cell. On the other hand, each individual object can appear in several regions, in several copies in each of them.

Several notions defined for membrane structures are extended in the natural way to super-cells: degree, depth, region, etc.

The multiset corresponding to a region of a super-cell (in particular, it can be an elementary membrane) is called *the contents* of it. The total multiplicities of the elements in an elementary membrane  $m$  (the sum of their multiplicities) is called *the size* of  $m$  and is denoted by  $size(m)$ .

If a membrane  $m'$  is placed in a membrane  $m$  such that they contribute to delimiting the same region (namely, the region associated with  $m$ ), then all objects placed in the region associated with  $m$  are said to be *adjacent* to membrane  $m'$  (so, they are immediately “outside” membrane  $m'$  and “inside” membrane  $m$ ).

The *support* of a super-cell  $\pi$ , denoted by  $supp(\pi)$  is the set of all objects appearing in  $\pi$  at least once.

A super-cell can be described by a Venn diagram where both the membranes and the objects are represented (in the case of the objects, taking care of multiplicities; for instance, we can write strings as representations of multisets).

Many further notions can be defined and investigated for super-cells as a goal *per se*. We do not step here into this direction, but we only mention that several operations with super-cells are natural: *merge* (putting together two or more super-cells in a new super-cell), *dissolve* a given membrane (but not the skin, because it defines the super-cell itself), *substitute* an elementary membrane with a given super-cell, *separate* membranes and/or objects of a super-cell, according to given criteria and producing two or more super-cells, etc. Such operations remind to us some of the operations with test tubes used in [1], [2], [10]; those “test tube structures” can be considered super-cells of depth two, with all objects – DNA molecules mainly – placed in the elementary membranes, the test tubes.

## 5 Transition Super-cell Systems

We now introduce the main subject of our investigation, a computing mechanism essentially designed as a distributed parallel machinery, having as the underlying structure a super-cell. The basic additional feature is the possibility of objects to evolve, according to certain rules. Another feature refers to the definition of the input and the output of a computation.

A *transition super-cell system* of degree  $n$ ,  $n \geq 1$ , is a construct

$$\Pi = (V, \mu, M_1, \dots, M_n, (R_1, \rho_1), \dots, (R_n, \rho_n), i_0),$$

where:

- (i)  $V$  is an alphabet; its elements are called *objects*;

- (ii)  $\mu$  is a membrane structure of degree  $n$ , with the membranes and the regions labeled in a one-to-one manner with elements in a given set  $\Lambda$ ; in this section we always use the labels  $1, 2, \dots, n$ ;
- (iii)  $M_i, 1 \leq i \leq n$ , are multisets over  $V$  associated with the regions  $1, 2, \dots, n$  of  $\mu$ ;
- (iv)  $R_i, 1 \leq i \leq n$ , are finite sets of *evolution rules* over  $V$  associated with the regions  $1, 2, \dots, n$  of  $\mu$ ;  $\rho_i$  is a partial order relation over  $R_i$ ,  $1 \leq i \leq n$ , specifying a *priority* relation among rules of  $R_i$ .  
An evolution rule is a pair  $(u, v)$ , which we will usually write in the form  $u \rightarrow v$ , where  $u$  is a string over  $V$  and  $v = v'$  or  $v = v'\delta$ , where  $v'$  is a string over

$$(V \times \{here, out\}) \cup (V \times \{in_j \mid 1 \leq j \leq n\}),$$

and  $\delta$  is a special symbol not in  $V$ . The length of  $u$  is called *the radius* of the rule  $u \rightarrow v$ .

- (v)  $i_0$  is a number between 1 and  $n$  which specifies the *output* membrane of  $\Pi$ .

Of course, any of the multisets  $M_1, \dots, M_n$  can be empty and the same is valid for the sets  $R_1, \dots, R_n$  and their associated priority relations  $\rho_i$ .

The components  $\mu$  and  $M_1, \dots, M_n$  of a super-cell system define a super-cell. Grafically, we will represent a super-cell system by representing its underlying super-cell, and also adding the rules to each region, together with the corresponding priority relation. In this way, we can have a complete picture of a super-cell system, much easier to understand than a symbolic description.

The components  $\mu, M_1, \dots, M_n, (R_1, \rho_1), \dots, (R_n, \rho_n)$  constitute the *initial configuration* of  $\Pi$ . In general, any sequence  $\mu', M'_{i_1}, \dots, M'_{i_k}, (R_{i_1}, \rho_{i_1}), \dots, (R_{i_k}, \rho_{i_k})$ , with  $\mu'$  a membrane structure obtained by removing from  $\mu$  all membranes different from  $i_1, \dots, i_k$  (of course, the skin membrane is not removed), with  $M'_j$  multisets over  $V$ ,  $1 \leq j \leq k$ , and  $\{i_1, \dots, i_k\} \subseteq \{1, 2, \dots, n\}$ , is called a *configuration* of  $\Pi$ .

It should be noted the important detail that the membranes preserve the initial labeling in all subsequent configurations; in this way, the correspondence between membranes, multisets of objects, and sets of evolution rules is well specified by the subscripts of these elements.

A more compact and easy to read writing of a configuration, avoiding the use of subscripts for multisets and sets above is that where the objects of the multisets are written (using multisets or in the form of a string) directly in the region to which they belong, and, similarly, the rules are written in the region where they can act. This is in a good correspondence with the

graphical representation of a transition super-cell system and we will use it especially for configurations where many components are empty.

For two configurations

$$\begin{aligned} C_1 &= (\mu', M'_{i_1}, \dots, M'_{i_k}, (R_{i_1}, \rho_{i_1}), \dots, (R_{i_k}, \rho_{i_k})), \\ C_2 &= (\mu'', M''_{j_1}, \dots, M''_{j_l}, (R_{j_1}, \rho_{j_1}), \dots, (R_{j_l}, \rho_{j_l})) \end{aligned}$$

of  $\Pi$  we write  $C_1 \implies C_2$ , and we say that we have a *transition* from  $C_1$  to  $C_2$ , if we can pass from  $C_1$  to  $C_2$  by using the evolution rules appearing in  $R_{i_1}, \dots, R_{i_k}$  in the following manner (rather than a completely cumbersome formal definition we prefer an informal one, explained by examples):

Consider a rule  $u \rightarrow v$  in a set  $R_{i_t}$ . We look to the region of  $\mu'$  associated with the membrane  $i_t$ . If the objects mentioned by  $u$ , with the multiplicities specified by  $u$ , appear in  $M'_{i_t}$  (that is, the multiset identified by  $u$  is included in  $M'_{i_t}$ ), then these objects can evolve according to the rule  $u \rightarrow v$ . The rule can be used only if no rule of a higher priority exists in  $R_{i_t}$  and can use the objects in  $u$ . More precisely, we start to examine the rules in the decreasing order of their priority and assign objects to them. A rule can be used only when there are copies of the objects whose evolution it describes and which were not “consumed” by rules of a higher priority. Therefore, all objects to which a rule *can* be applied *must* be the subject of a rule application. All objects in  $u$  are “consumed” by using the rule  $u \rightarrow v$ , that is, the multiset identified by  $u$  is subtracted from  $M'_{i_t}$ .

The result of using the rule is determined by  $v$ . If an object appears in  $v$  in a pair  $(a, \textit{here})$ , then it will remain in the same region  $i_t$ . (Often, when specifying rules, pairs  $(a, \textit{here})$  are simply written  $a$ , the indication “here” is omitted.) If an object appears in  $v$  in a pair  $(a, \textit{out})$ , then  $a$  will exit the membrane  $i_t$  and will become an element of the region immediately outside it (thus, it will be adjacent to the membrane  $i_t$  from which it was expelled). In this way, it is possible that an object leaves the super-cell itself: if it goes outside the skin of the super-cell, then it never comes back. If an object appears in a pair  $(a, \textit{in}_q)$ , then  $a$  will be added to the multiset  $M'_q$ , providing that  $a$  is adjacent to the membrane  $q$ . If  $(a, \textit{in}_q)$  appears in  $v$  and the membrane  $q$  is not one of the membranes delimiting “from below” the region  $i_t$ , then the application of the rule is not allowed.

If the symbol  $\delta$  appears in  $v$ , then the membrane  $i_t$  is removed (we say *dissolved*) and at the same time the set of rules  $R_{i_t}$  (and its associated priority relation) is removed. The multiset  $M'_{i_t}$  is added (in the sense of multisets union) to the multiset associated with the region which was immediately external to the membrane  $i_t$ . We do not allow the dissolving of the skin, because this means that the super-cell is lost, we do no longer have a correct configuration of the system.

All these operations are done in parallel, for all possible applicable rules  $u \rightarrow v$ , for all occurrences of multisets  $u$  in the region associated with the

rules, for all regions at the same time. No contradiction appears because of multiple membrane dissolving, or because simultaneous appearance of symbols of the form  $(a, out)$  and  $\delta$ . If at the same step we have  $(a, in_i)$  outside a membrane  $i$  and  $\delta$  inside this membrane, then, because of the simultaneity of performing these operations, again no contradiction appears: we assume that  $a$  is introduced in membrane  $i$  at the same time when it is dissolved, thus  $a$  will remain in the region placed outside membrane  $i$ ; that is, the effect of  $(a, in_i), \delta$  is  $(a, here)$ .

If there are rules in a super-cell system  $\Pi$  with the radius at least two, then the system is said to be *cooperative*; in the opposite case, it is called *non-cooperative*. A system is said to be *catalytic* if there are certain objects  $c_1, \dots, c_n$  specified in advance, called *catalysts*, such that the rules of the system are either of the form  $a \rightarrow v$ , or of the form  $c_i a \rightarrow c_i v$ , where  $a$  is a non-catalysts object and  $v$  contains no catalyst. (So, the only cooperative rules involve catalysts, which are reproduced by the rule application, and left in the same place. There are no rules for the separate evolution of catalysts. A natural generalization is to allow the catalysts to evolve, but we do not consider this variant here.) A transition super-cell system with catalysts is given in the form  $\Pi = (V, C, \mu, M_1, \dots, M_n, (R_1, \rho_1), \dots, (R_n, \rho_n), i_0)$ , where  $C \subseteq V$  is the set of catalysts. A system is said to be *propagating* if there is no rule which diminishes the number of objects in the system (note that this can be done by “erasing” rules, but also by sending objects out of the skin membrane).

**Remark 1.** The mode of evolving of objects in a super-cell provided with evolution rules as described above can be interpreted in the following – idealized – biochemical way. We have an organism, delimited by a skin (the skin membrane). Inside, there are organs and free molecules, organized hierarchically. The molecules and the organs float randomly in the “cytoplasmic liquid” of each membrane. Under specific conditions, the molecules evolve, alone or with the help of certain catalysts; these, of course, are not modified by the reactions. This is done in parallel, synchronously for all molecules (a universal clock is assumed to exist). The new molecules can remain in the same region where they have appeared, or can pass through the membranes delimiting this space, selectively. Some reactions not only modify molecules, but also break membranes. (We may imagine that certain chemicals are produced which break/dissolve the membrane.) When a membrane is broken, the molecules previously placed inside it will remain free in the larger space newly created, but the evolution rules of the former membrane are lost. The assumption is that the reaction conditions from the previous membrane are modified by the disparition of the membrane and in the newly created space only the rules specific to this space can act. Of course, when the external membrane is broken, then the organism ceases to exist, its organs fall apart.

The following **example** will (hopefully) clarify the definition of a transition in a (cooperative) super-cell system.

Consider the super-cell system of degree 4:

$$\begin{aligned}
\Pi &= (V, \mu, M_1, \dots, M_4, (R_1, \rho_1), \dots, (R_4, \rho_4), 4), \\
V &= \{a, b, c, d\}, \\
\mu &= [{}_1[{}_2[{}_3]_3]_2[{}_4]_4]_1, \\
M_1 &= \{aac\}, \\
M_2 &= \{a\}, \\
M_3 &= \{cd\}, \\
M_4 &= \emptyset, \\
R_1 &= \{r_1 : c \rightarrow (c, in_4), r_2 : c \rightarrow (b, in_4), \\
&\quad r_3 : a \rightarrow (a, in_2)b, dd \rightarrow (a, in_4)\}, \\
\rho_1 &= \{r_1 > r_3, r_2 > r_3\}, \\
R_2 &= \{a \rightarrow (a, in_3), ac \rightarrow \delta\}, \\
\rho_2 &= \emptyset, \\
R_3 &= \{a \rightarrow \delta\}, \\
\rho_3 &= \emptyset, \\
R_4 &= \{c \rightarrow (d, out), b \rightarrow b\}, \\
\rho_4 &= \emptyset.
\end{aligned}$$

The system and the configurations obtained after two possible transitions are represented in Figure 1.

In the initial configuration we can apply a rule in membrane 1 and one in membrane 2. If in membrane 1 we use the rule  $c \rightarrow (b, in_4)$ , then the computation will never halts: the rule  $b \rightarrow b$  can be applied for ever in membrane 4. Thus, we will not use the rule  $c \rightarrow (b, in_4)$ , but the rule  $c \rightarrow (c, in_4)$ . Because both these rules can be applied and they have priority over the rule  $a \rightarrow (a, in_2)b$ , this latter rule cannot be used. Thus, a symbol  $c$  is sent from membrane 1 to membrane 4 and at the same time a symbol  $a$  is sent from membrane 2 to membrane 3.

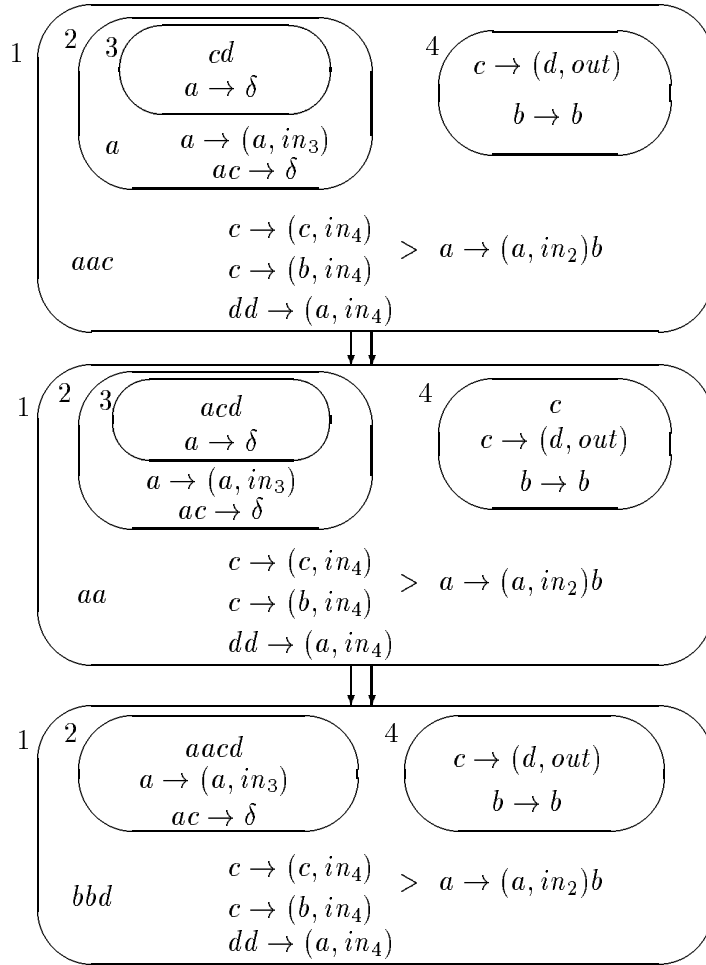
Now, no  $c$ -rule in membrane 1 can be applied, hence the rule  $a \rightarrow (a, in_2)b$  can be used. It has to be used for both copies of  $a$  in membrane 1, hence two copies of  $a$  will be sent to membrane 2 and two copies of  $b$  will remain in membrane 1. At the same time, the rule  $a \rightarrow \delta$  will be used in membrane 3, dissolving it, and the rule  $c \rightarrow (d, out)$  will be used in membrane 4, sending a copy of  $d$  to membrane 1. As a result of these operations, membrane 1 will contain the multiset (we write it as a string)  $bbd$ , membrane 2 will contain  $aacd$ , while membrane 4 is empty; membrane 3 does no longer exist (hence the rule  $a \rightarrow (a, in_3)$  in membrane 2 is useless from now on).

Two more transitions can be performed. First, the rule  $ac \rightarrow \delta$  can be

used in membrane 2, dissolving it and releasing the remaining objects  $ad$ . Thus, membrane 1 will contain the multiset  $abbdd$ , which makes possible for the first time the use of the rule  $dd \rightarrow (a, in_4)$  from membrane 1. It consumes the two copies of  $d$  and sends a copy of  $a$  to membrane 4. No further rule can be applied, the “life” of the super-cell stops here.

The computing flavour of such a game is obvious: we start from an initial configuration of our system provided with evolution rules and we get a sequence of transitions.

A sequence of transitions in a super-cell system  $\Pi$ , starting from the initial configuration  $C_0$ , is called a *computation* with respect to  $\Pi$ .



**Figure 1.** An example of transitions in a super-cell system.

A computation  $C_0 \Rightarrow C_1 \Rightarrow \dots \Rightarrow C_m$ ,  $m \geq 0$ , is *successful* if and

only if both of the following assertions are true:

1. There is no rule in  $C_m$  which can be applied to the objects present in  $C_m$ .
2. The membrane  $i_0$  appears in  $C_m$ , namely, as an elementary membrane of it.

Reversing these statements, a computation as above is unsuccessful in each of the following two cases:

- It can continue, that is, there exists a configuration  $C_{m+1}$  such that  $C_m \Rightarrow C_{m+1}$ . Note that it is not necessary to have  $C_m \neq C_{m+1}$ .
- No rule can be applied, but either there is no membrane labeled with  $i_0$  (it has been dissolved by a symbol  $\delta$ ), or there is such a membrane, but it is not an elementary membrane in  $C_m$ .

In this way, a super-cell system  $\Pi$  can be seen as a device which generates multisets: start from the initial configuration of  $\Pi$  and let the system evolve. If a successful computation is found, then we say that the multiset contained by the membrane labeled with  $i_0$  is *generated* by  $\Pi$ .

We can also consider the super-cell systems as devices which generate numbers: work as above and say that the size of the membrane  $i_0$  (remember that the size is the sum of multiplicities of objects in a membrane) is the generated number. In what follows, we consider this latter possibility. We denote by  $N(\Pi)$  the set of natural numbers generated by  $\Pi$  in the previous sense.

A generalization is to use a super-cell system  $\Pi$  for generating *relations*. For instance, we can specify in advance certain objects  $a_{i_1}, \dots, a_{i_k}$ ; if at the end of a successful computation the output membrane contains  $n_1, \dots, n_k$  occurrences of objects  $a_{i_1}, \dots, a_{i_k}$ , respectively, then we say that  $(n_1, \dots, n_k)$  belongs to the relation generated by  $\Pi$ .

It is also possible to interpret a super-cell system  $\Pi$  as a device *recognizing* a multiset (that initially placed in a distinguished elementary *input* membrane), or a number (the size of an input elementary membrane), or a relation (the number of occurrences of certain objects placed in a specified input membrane), or even as a device *computing* a partial mapping from natural numbers to sets of natural numbers (give a number as an input, codified in the size of a distinguished elementary membrane, and collect all numbers obtained as outputs at the end of successful computations – if any). We will exemplify these possibilities in the next section.

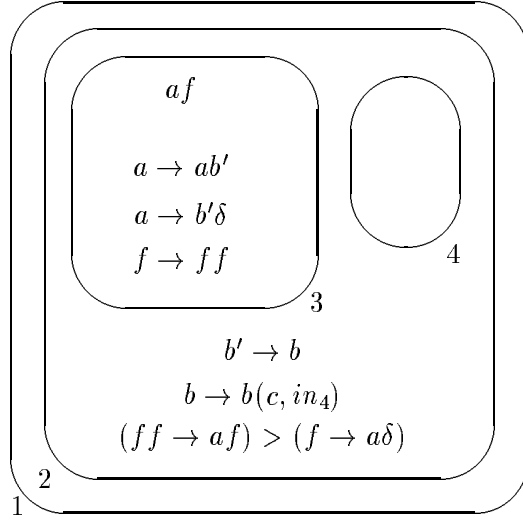
## 6 Examples

Before going to investigate the power of super-cell systems, we will examine some examples.

**Example 1.** Consider the super-cell system of degree 4

$$\begin{aligned}
\Pi_1 &= (V, \mu, M_1, \dots, M_4, (R_1, \rho_1), \dots, (R_4, \rho_4), 4), \\
V &= \{a, b, b', c, f\}, \\
\mu &= [{}_1[{}_2[{}_3[{}_4]_4]_2]_1], \\
M_1 &= \emptyset, R_1 = \emptyset, \rho_1 = \emptyset, \\
M_2 &= \emptyset, R_2 = \{b' \rightarrow b, b \rightarrow b(c, in_4), r_1 : ff \rightarrow af, r_2 : f \rightarrow a\delta\}, \\
\rho_2 &= \{r_1 > r_2\}, \\
M_3 &= \{af\}, R_3 = \{a \rightarrow ab', a \rightarrow b'\delta, f \rightarrow ff\}, \rho_3 = \emptyset, \\
M_4 &= \emptyset, R_4 = \emptyset, \rho_4 = \emptyset.
\end{aligned}$$

(For the sake of simplicity, we have labeled only the rules which appear in the priority relation.) The system is presented in Figure 2.



**Figure 2.** A super-cell system generating  $n^2, n \geq 1$ .

No object is free in membrane 2, hence no rule can be applied here. The only possibility is to start in membrane 3, using the free objects  $a, f$ , present in one copy each. Using the rules  $a \rightarrow ab', f \rightarrow ff$ , in parallel for all occurrences of  $a$  and  $f$  currently available, after  $n$  steps,  $n \geq 0$ , we get  $n$  occurrences of  $b'$  and  $2^n$  occurrences of  $f$ . In any moment, instead of  $a \rightarrow ab'$  we can use  $a \rightarrow b'\delta$  (note that we always have only one copy of  $a$ ). In that moment we have  $n + 1$  occurrences of  $b'$  and  $2^{n+1}$  occurrences of  $f$  and we dissolve membrane 3. The obtained configuration is

$$\begin{aligned}
&[{}_1[{}_2b'^{n+1}f^{2^{n+1}}, b' \rightarrow b, b \rightarrow b(c, in_4), \\
&\quad r_1 : ff \rightarrow af, r_2 : f \rightarrow a\delta, r_1 > r_2, [{}_4]_4]_2]_1.
\end{aligned}$$

(We have used again the more compact string notation,  $\alpha^i$ , instead of the multiset notation  $(\alpha, i)$ .)

The rules of the former active membrane 3 are lost, the rules of membrane 2 are now active. Due to the priority relation, we have to use the rule  $ff \rightarrow af$  as much as possible. In one step, we pass from  $b^{n+1}$  to  $b^{n+1}$ , while the number of  $f$  occurrences is divided by two. In the next step, from  $b^{n+1}$ ,  $n+1$  occurrences of  $c$  are introduced in membrane 4 (each occurrence of the symbol  $b$  introduces one occurrence of  $c$ ). At the same time, the number of  $f$  occurrences is divided again by two. We can continue. At each step, further  $n+1$  occurrences of  $c$  are introduced in the output membrane. This can be done  $n+1$  steps:  $n$  times when the rule  $ff \rightarrow af$  is used (thus diminishing the number of  $f$  occurrences to one), and one when using the rule  $f \rightarrow a\delta$  (it may now be used). In this moment, membrane 2 is dissolved, which entails the fact that its rules are removed. No further step is possible. The obtained configuration is

$$[_1 a^{2^{n+1}} b^{n+1}, [_4 c^{(n+1)^2}]_4]_1.$$

Consequently,

$$N(\Pi) = \{m^2 \mid m \geq 1\}.$$

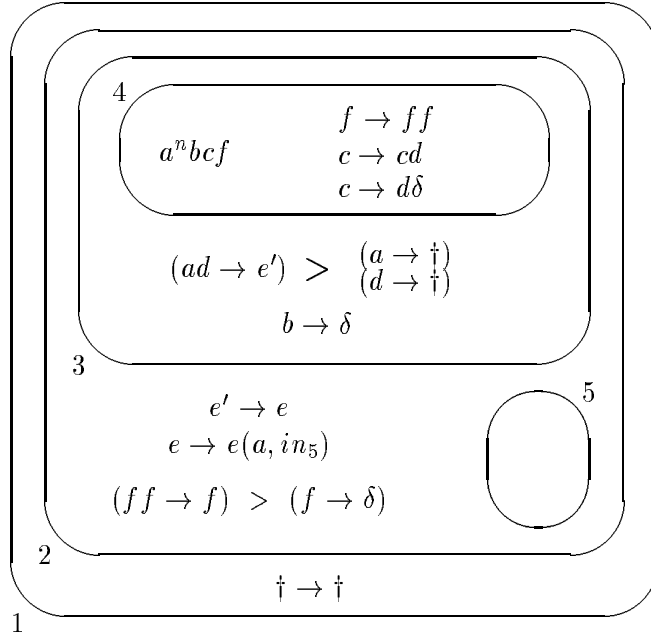
If we omit membrane 4 (then the rule  $b \rightarrow b(c, in_4)$  is replaced by  $b \rightarrow bc$ ) and consider membrane 1 as the output membrane, then we can generate the set of numbers  $\{2^n + n^2 + n \mid n \geq 1\}$  (all objects ever used contribute to the output). Furthermore, if we also distinguish the occurrences of  $b$  from those of  $c$ , then we generate the relation  $\sigma = \{(n, m) \mid n \text{ is the square root of } m\}$ .

Note that the super-cell system  $\Pi_1$  is propagating and it has only one cooperative rule.

**Example 2.** The previous super-cell system is a generative one: it starts from a unique initial configuration and, because of the nondeterministic evolution, it collects in its output membrane different values of  $n^2$ ,  $n \geq 1$ . A variant of interest could be a super-cell system just computing  $n^2$  for a given  $n$ . Such a system is the next one (it has the degree 5; in order to have only propagating rules, we can add a dummy object to the right hand member of each rule which diminishes the number of objects occurrences, but we do not deal here with this detail):

$$\begin{aligned} \Pi_2 &= (V, \mu, M_1, \dots, M_5, (R_1, \rho_1), \dots, (R_5, \rho_5), 5), \\ V &= \{a, c, d, e, e', f, \dagger\}, \\ \mu &= [_1 [_2 [_3 [_4 ]_4]_3]_5]_2]_1, \\ M_1 &= \emptyset, R_1 = \{\dagger \rightarrow \dagger\}, \rho_1 = \emptyset, \end{aligned}$$

$$\begin{aligned}
M_2 &= \emptyset, R_2 = \{e' \rightarrow e, e \rightarrow e(a, in_5), r_1 : ff \rightarrow f, r_2 : f \rightarrow \delta\}, \\
\rho_2 &= \{r_1 > r_2\}, \\
M_3 &= \emptyset, R_3 = \{r_3 : ad \rightarrow e', r_4 : a \rightarrow \dagger, r_5 : d \rightarrow \dagger, b \rightarrow \delta\}, \\
\rho_3 &= \{r_3 > r_4, r_3 > r_5\}, \\
M_4 &= \{a^n bcf\}, R_4 = \{f \rightarrow ff, c \rightarrow cd, c \rightarrow d\delta\}, \rho_4 = \emptyset, \\
M_5 &= \emptyset, R_5 = \emptyset, \rho_5 = \emptyset.
\end{aligned}$$



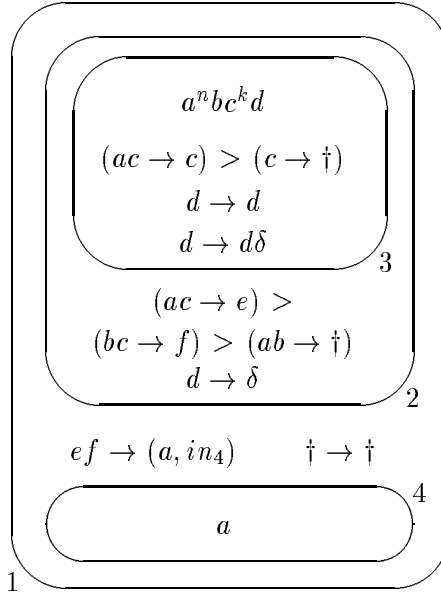
**Figure 3.** A super-cell system computing  $f(n) = n^2$ .

The system is represented in Figure 3 and it works as follows. In membrane 4 (this is the only region of the initial configuration with applicable rules) we produce  $m$  copies of  $d$  simultaneously with  $2^m$  copies of  $f$ . In any moment (but with  $m \geq 1$ ), the membrane is dissolved. Thus, in the next membrane, 3, we have  $n$  copies of  $a$ ,  $m$  copies of  $d$ ,  $2^m$  copies of  $f$ , and one copy of  $b$ . The occurrences of  $a$  and  $d$  are checked for equality by the rule  $ad \rightarrow e'$ . (Due to parallelism, this is done in one step only.) If  $n \neq m$ , then the trap-object  $\dagger$  is introduced and the computation will never halt: this object will eventually arrive in region 1, where the rule  $\dagger \rightarrow \dagger$  can then be used for ever. If  $n = m$ , then we can continue: the membrane is dissolved and we reach membrane 2 with  $n$  copies of  $e'$  and  $2^n$  copies of  $f$ . The copies of  $f$  are diminished by a factor of two at each iteration; at the same time,  $n$  occurrences of  $a$  are sent to the output membrane. Consequently,  $n^2$  copies of  $a$  are collected in the output membrane (the last use of an  $f$ -rule,  $f \rightarrow \delta$ ,

corresponds to the use of the rule  $e' \rightarrow e$ , when no object  $a$  is produced). At the end, membrane 2 is dissolved, so no more rule is available, the computation stops; this also makes possible that the trap-object, if present, can reach the skin and make the computation continue for ever. Thus, we compute the passing from  $n$  to  $n^2$ .

If we replace the rule  $e' \rightarrow e$  of membrane 2 with the rule  $e' \rightarrow e(a, in_5)$ , then we compute the function  $f(n) = n^2 + n$ . The reader can consider other possibilities.

**Example 3.** Let us now consider a super-cell system which has a decidability task: we introduce in the input configuration two numbers,  $n$  and  $k$ , and ask whether or not  $n$  is a multiple of  $k$ . In the affirmative case, we will finish with one object in the output membrane; in the negative case we will have two objects in the output membrane.



**Figure 4.** A super-cell system deciding whether  $k$  divides  $n$ .

The system is the following (of degree 4):

$$\begin{aligned}
\Pi_3 &= (V, \mu, M_1, \dots, M_4, (R_1, \rho_1), \dots, (R_4, \rho_4), 4), \\
V &= \{a, b, c, d, e, f, \dagger\}, \\
\mu &= [{}_1[{}_2[{}_3]_3]_2[{}_4]_4]_1, \\
M_1 &= \emptyset, R_1 = \{ef \rightarrow (a, in_4), \dagger \rightarrow \dagger\}, \rho_1 = \emptyset, \\
M_2 &= \emptyset, R_2 = \{r_1 : ac \rightarrow e, r_2 : bc \rightarrow f, r_3 : ab \rightarrow \dagger, d \rightarrow \delta\}, \\
\rho_2 &= \{r_1 > r_2, r_2 > r_3\},
\end{aligned}$$

$$\begin{aligned}
M_3 &= \{a^n b c^k d\}, R_3 = \{r_4 : ac \rightarrow c, r_5 : c \rightarrow \dagger, d \rightarrow d, d \rightarrow d\delta\}, \\
\rho_3 &= \{r_4 > r_5\}, \\
M_4 &= \{a\}, R_4 = \emptyset, \rho_4 = \emptyset.
\end{aligned}$$

The structure of  $\Pi_3$  is better seen in Figure 4. In membrane 3 we subtract  $k$  from  $n$ , repeatedly (by the rule  $ac \rightarrow c$ : at each step,  $k$  copies of  $a$  disappear, while  $c$  is reproduced). At any time, the symbol  $d$  will introduce  $\delta$ , and this membrane is dissolved.

We distinguish several cases.

If  $n$  is a multiple of  $k$ , then in membrane 3 we can exhaust the occurrences of  $a$ ; no rule involving this object will be applicable in membrane 2, hence no symbol  $a$  is introduced in the output membrane. The symbol  $\dagger$  is not produced, so the computation ends correctly. If  $n$  is a multiple of  $k$ , but in membrane 3 we do not exhaust the occurrences of  $a$ , but we leave exactly  $k$  copies, then these copies are consumed in membrane 2 by the rule  $ac \rightarrow e$ , of the highest priority (thus, the rule  $bc \rightarrow f$  cannot be used) and again nothing is produced in the output membrane (and the computation stops correctly). If more than  $k$  copies of  $a$  remain, then the rule  $ab \rightarrow \dagger$  can be used in membrane 2 (after using  $ac \rightarrow e$  for  $k$  pairs of  $a$  and  $c$ , at least one occurrence of  $a$  remains and can be paired with  $b$ ), which will make the computation continue for ever; we do not get a wrong answer (in fact, we get no answer).

If  $n$  is not a multiple of  $k$ , and we dissolve membrane 3 with more than  $k$  occurrences of  $a$  which are left unconsumed, then again the rule  $ab \rightarrow \dagger$  can be used in membrane 2: the  $k$  copies of  $c$  are paired with  $k$  copies of  $a$  and at least one remaining copy of  $a$  can be paired with  $b$ . If  $n$  is not a multiple of  $k$  and we leave membrane 3 with less than  $k$  occurrences of  $a$  (that is, with the remainder of dividing  $n$  by  $k$ , which is a number between 1 and  $k$ , strictly smaller than  $k$ ), then in membrane 2 we can introduce both the symbol  $e$  (maybe several times, as many occurrences of  $a$  we have) and the symbol  $f$  (the trap-object is not produced, because of the priority relation). In this way, in the skin membrane we can use the rule  $ef \rightarrow (a, in_4)$ , that is, a second occurrence of  $a$  is introduced in the output membrane. The computation stops.

In conclusion, if the computation is correctly finished, then the output membrane contains two objects if and only if  $n$  is not a multiple of  $k$  (in the opposite case, we have here only one object).

All super-cells considered above were cooperative systems and always the rules were either propagating or easy to modify in order to obtain propagating rules. We have insisted on the intricate behaviour of the super-cell systems and not on their parallelism. This parallelism appears at two levels: the objects in each membrane evolve in parallel, while the membranes themselves evolve in parallel. The influence of the parallelism on the complexity

of computing the output (in comparison with other computing models) is one of the main research topics left open.

## 7 The Power of Transition Super-cell Systems

The transition super-cell systems are computationally complete, systems of a simple structure can compute all recursively enumerable sets of natural numbers. In the proof of this result we need the notion of a *matrix grammar with appearance checking*.

Such a grammar is a construct  $G = (N, T, S, P, F)$ , where  $N, T$  are disjoint alphabets,  $S \in N$ ,  $P$  is a finite set of sequences of the form  $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$ ,  $n \geq 1$ , 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  $P$  (we say that  $N$  is the nonterminal alphabet,  $T$  is the terminal alphabet,  $S$  is the axiom, while the elements of  $P$  are called matrices).

For  $w, z \in (N \cup T)^*$  we write  $w \implies z$  if there is a matrix  $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$  in  $P$  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  $A_i \rightarrow x_i$  appears in  $F$ . (The rules of a matrix are applied in order, possibly skipping the rules in  $F$  if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If  $F = \emptyset$ , then the grammar is said to be without appearance checking (and  $F$  is no longer mentioned).

We denote by  $\implies^*$  the reflexive and transitive closure of the relation  $\implies$ . The language generated by  $G$  is defined by  $L(G) = \{w \in T^* \mid S \implies^* w\}$ . The family of languages of this form is denoted by  $MAT_{ac}$ . When the set  $F$  is empty, hence all rules have to be applied effectively (the grammars are said to be *without appearance checking*), the obtained family is denoted by  $MAT$ .

We denote by  $REG$ ,  $CF$ ,  $CS$ ,  $RE$  the basic families in the Chomsky hierarchy: of regular, context-free, context-sensitive, and recursively enumerable languages, respectively. When dealing with numbers,  $RE$  denotes the family of recursively enumerable sets of natural numbers.

It is known that  $CF \subset MAT \subset MAT_{ac} = RE$ . Further details about matrix grammars can be found in [6] and in [16].

We also consider here *EOL systems*, which are constructs of the form  $G = (V, T, w, P)$ , where  $V$  is an alphabet,  $T \subseteq V$ ,  $w \in V^*$ , and  $P$  is a finite set of context-free rules  $a \rightarrow x$  over  $V$ ; for each  $a \in V$  there is at least one rule  $a \rightarrow x$  in  $P$  (we say that  $P$  is *complete*). For  $y, z \in V^*$  we write  $y \implies z$  iff  $y = a_{i_1} \dots a_{i_k}$ ,  $z = x_{i_1} \dots x_{i_k}$ , for  $a_{i_j} \rightarrow x_{i_j} \in P$ ,  $1 \leq j \leq k$ . The language generated by  $G$  is  $L(G) = \{z \in T^* \mid w \implies^* z\}$ . We denote by  $EOL$  the family of these languages and by  $Ls(EOL)$  the family of length



*Proof.* The inclusions between *TSC* families are obvious from the definitions. The inclusion  $TSC(Coo, \delta) \subseteq RE$  can be proved in a straightforward manner (or we can invoke the Church-Turing thesis). The inclusions  $Ls(E0L) \subseteq TSC_1(nCoo)$  and  $RE \subseteq TSC_2(Cat)$  are proved in the following lemmas. ■

**Lemma 1.**  $Ls(E0L) \subseteq TSC_1(nCoo)$ .

*Proof.* Consider an E0L system  $G = (V, T, w, P)$ . For each symbol  $a \in V$  we consider a new symbol  $a'$ . Let  $V'$  be the set of these symbols and  $h$  the morphism defined by  $h(a) = a'$ , for  $a \in V$ . Assume that  $P$  contains  $m$  rules,  $p_i : a_i \rightarrow x_i$ ,  $1 \leq i \leq m$ .

We construct the transition super-cell system of degree 1

$$\Pi = (V \cup V' \cup \{d, e, \dagger\}, [1]_1, \{dh(w)\}, (R_1, \rho_1), 1),$$

with the following rules:

1.  $r_1 : d \rightarrow d$ ,
2.  $r_2 : d \rightarrow e$ ,
3.  $r'_i : a' \rightarrow h(x_i)$ , for  $i = 1, 2, \dots, m$ ,
4.  $r_3 : e \rightarrow (e, out)$ ,
5.  $r_a : a' \rightarrow a$ , for  $a \in T$ ,
6.  $r'_a : a' \rightarrow \dagger$ , for  $a \in V - T$ ,
7.  $r_\infty : \dagger \rightarrow \dagger$ ,

and the priority relations

$$\begin{aligned} r_1 > r_a, r_1 > r'_a, r_2 > r_a, r_2 > r'_a, \text{ for all possible } a, \\ r_3 > r'_i, 1 \leq i \leq m. \end{aligned}$$

The system works as follows. To a multiset (represented here by a string)  $dh(z)$  we can apply the rule  $r_1$  and nothing is changed; this forbids the use of rules  $r_a, r'_a$ . As long as  $d$  is present, each symbol  $a'$  present in the current string should evolve by using a rule  $r'_i$  associated with the corresponding rule in  $P$ . In this way, we simulate the derivations in  $G$ , using sentential forms composed of primed symbols. At any moment we can use the rule  $d \rightarrow e$ . Because  $r_3$  is now applicable, no rule  $r'_i$  can be used. However, the rules  $r_a, r'_a$  are now applicable. If the obtained string is terminal with respect to  $G$ , then all primed symbols are replaced by their non-primed versions and the computation stops. If a symbol  $a'$  is present, with  $a \in V - T$ , then the trap-object  $\dagger$  is introduced and the computation will continue for ever.

Consequently,  $Ls(L(G)) = N(\Pi)$ . ■

**Lemma 2.** (The Computational Completeness Lemma for Transition Super-cell Systems)  $RE \subseteq TSC_2(Cat)$ .

*Proof.* Clearly, each set  $Q \subseteq \mathbb{N}$  can be identified with the language  $L(Q) = \{a^n \mid n \in Q\}$  and  $Q$  is recursively enumerable if and only if  $L(Q)$  is recursively enumerable. Take a matrix grammar with appearance checking,  $G = (N, \{a\}, S, M, F)$  generating the language  $L(Q)$ , for a given recursively enumerable set  $Q$  of numbers.

According to Lemma 1.3.7 in [6], without loss of generality we may assume that  $N = N_1 \cup N_2 \cup \{S, \dagger\}$ , with these three sets mutually disjoint, and that the matrices in  $P$  are of one of the following forms:

1.  $(S \rightarrow XA)$ , with  $X \in N_1, A \in N_2$ ,
2.  $(X \rightarrow Y, A \rightarrow x)$ , with  $X, Y \in N_1, A \in N_2, x \in (N_2 \cup \{a\})^*$ ,
3.  $(X \rightarrow Y, A \rightarrow \dagger)$ , with  $X, Y \in N_1, A \in N_2$ ,
4.  $(X \rightarrow x_1, A \rightarrow x_2)$ , with  $X \in N_1, A \in N_2$ , and  $x_1, x_2 \in \{a\}^*$ .

Moreover, there is only one matrix of type 1 (we use then to write its rule in the form  $S \rightarrow X_0A_0$ , in order to stress the fact that these symbols are fixed) and  $F$  consists exactly of all rules  $A \rightarrow \dagger$  appearing in matrices of type 3. The symbols in  $N_1$  are mainly used to control the use of rules of the form  $A \rightarrow x$  with  $A \in N_2$ , while  $\dagger$  is a trap-symbol; once introduced, it is never removed. A matrix of type 4 is used only once, at the last step of a derivation.

Assume that all matrices of forms 2, 3, 4 are labeled in a one-to-one manner, by  $m_1, m_2, \dots, m_k$ .

We construct the following transition super-cell system with catalysts:

$$\Pi = (V, \{c\}, [{}_1[{}_2]_2]_1, \{X_0A_0cZ\}, \emptyset, (R_1, \rho_1), (\emptyset, \emptyset), 2),$$

where

$$V = N_1 \cup N_2 \cup \{c, D, \dagger, Z\} \cup \{X_i, X'_i, X''_i \mid X \in N_1, 1 \leq i \leq k\},$$

and the set  $R_1$  contains the following rules ( $h$  is the morphism defined by  $h(\alpha) = \alpha$ ,  $\alpha \in N_2$ , and  $h(a) = (a, in_2)$ ):

1.  $X \rightarrow X_i$ , for all  $X \in N_1$  and  $1 \leq i \leq k$ .
2.  $X_i \rightarrow Y'$ , for  $m_i : (X \rightarrow Y, A \rightarrow x)$  a matrix of type 2 in  $P$ .
3.  $cA \rightarrow c h(x)D$ , for  $m_i : (X \rightarrow Y, A \rightarrow x)$  a matrix of type 2 in  $P$ .

4.  $cD \rightarrow c$ .
5.  $cZ \rightarrow c\ddagger$ .
6.  $\ddagger \rightarrow \ddagger$ .
7.  $Y' \rightarrow Y$ , for all  $Y \in N_1$ .
8.  $cX_i \rightarrow cY$ , for  $m_i : (X \rightarrow Y, A \rightarrow \ddagger)$  a matrix of type 3 in  $P$ .
9.  $A \rightarrow \ddagger$ , for all  $A \in N_2$ .
10.  $X_i \rightarrow X'_i$ , for  $m_i : (X \rightarrow x_1, A \rightarrow x_2)$  a matrix of type 4 in  $P$ .
11.  $cA \rightarrow c h(x_2)D$ , for  $m_i : (X \rightarrow x_1, A \rightarrow x_2)$  a matrix of type 4 in  $P$ .
12.  $X'_i \rightarrow X''_i$ , for  $m_i : (X \rightarrow x_1, A \rightarrow x_2)$  a matrix of type 4 in  $P$ .
13.  $Z \rightarrow \lambda$ .
14.  $X''_i \rightarrow h(x_1)$ , for  $m_i : (X \rightarrow x_1, A \rightarrow x_2)$  a matrix of type 4 in  $P$ .

The priorities are the following (at the same time, we give explanations on the work of  $\Pi$ ):

- each rule of type 1 has priority over all rules of other types;  
(In the presence of a symbol from  $N_1$  no rule can be used, excepting a rule of type 1, which specifies a matrix to be simulated by the subscript of the symbol  $X$ .)
- each rule  $X_i \rightarrow Y'$  of type 2 has priority over all rules of type 3 associated to matrices  $m_j$  with  $j \neq i$ , as well as over all rules of types 5, 9, 11, 13;  
(If a symbol  $X_i$  is present, identifying a matrix  $m_i : (X \rightarrow Y, A \rightarrow x)$  of type 2 from  $P$ , then the only rules which can be applied are  $X_i \rightarrow Y'$ , because only  $X_i$  is present, and  $cA \rightarrow c h(x)D$ , because all other rules are either of a lower priority than  $X_i \rightarrow Y'$ , or do not have symbols to which they can be applied; note that always we have exactly one occurrence of the catalyst, hence the rule  $cA \rightarrow c h(x)D$  can be used at most once; by using this rule, one occurrence of the symbol  $D$  is introduced.)
- the rule of type 4 has priority over the rule of type 5;  
(This is a very important point of the construction, making a full use of the catalyst: if there is no occurrence of  $D$  in the multiset, then the rule  $cZ \rightarrow c\ddagger$  can be applied, introducing the trap-object  $\ddagger$  which will evolve for ever by the rule  $\ddagger \rightarrow \ddagger$ . Thus, at the same time with  $X_i \rightarrow Y'$  we have to use the corresponding rule  $cA \rightarrow c h(x)D$ , which means that the use of the matrix  $m_i$  is correctly simulated. Note that

the rule  $cZ \rightarrow c\ddagger$  cannot be used at the previous step, because of the priority of  $X_i \rightarrow Y'$  over it.)

- each rule  $Y' \rightarrow Y$  of type 7, for  $Y \in N_1$ , has priority over all rules of types 3, 9, 11, 13;  
(At the same time with the rule  $cD \rightarrow c$ , providing that  $D$  is present, we can use the rule  $Y' \rightarrow Y$ ; no rule associated with a rule appearing in the second position in a matrix can be applied, the simulation of the matrix  $m_i$  is completed.)
- each rule  $cX_i \rightarrow cY$  of type 8 has priority over all rules  $cA \rightarrow c h(x)D$  associated with matrices of types 2 and 4, over all rules  $A \rightarrow \ddagger$  associated with matrices  $m_j$  of type 3 with  $j \neq i$ , as well as over rules of types 5, 11, 13;  
(When the symbol  $X_i$  points to a matrix  $m_i$  of type 3, then the catalyst is “kept busy” by the rule  $cX_i \rightarrow cY$ , in order not to use the rule  $cZ \rightarrow c\ddagger$ ; no rule for evolving a symbol from  $N_2$  can be used, because of the priority; if, however, the symbol  $A$  from  $m_i : (X \rightarrow Y, A \rightarrow \ddagger)$  appears in the current multiset, then the corresponding rule of type 9 should be used and the trap-object is introduced. In this way, we simulate the use of this rule in the appearance checking mode.)
- each rule  $X_i \rightarrow X'_i$  of type 10 has priority over all rules of type 3, of type 11 associated with matrices  $m_j$  with  $j \neq i$ , as well as over all rules of types 5, 9, 13;  
(When simulating the use of a matrix of type 4, at the last step of a derivation in  $G$ , we proceed as for matrices of type 2, with the difference that at the end we have also to introduce a terminal string instead of the “control symbol”  $X$  and also we have to remove the primed successors of  $X$ .)
- each rule  $X'_i \rightarrow X''_i$  of type 12 has priority over all rules of types 3, 11, 13;  
(After introducing  $X_i$  we replace it with  $X'_i$  and, at the same time, we use the corresponding rule  $cA \rightarrow c h(x_2)D$ . At the next step, we check whether or not  $D$  is introduced, that is, whether or not the simulation is correct. The symbol  $Z$  is still present, but it is not used, because of the priorities mentioned above. At the same time, we check whether or not any nonterminal symbol from  $N_2$  is still present: the rules  $A \rightarrow \ddagger$  are available and no other rule using symbols from  $N_2$  can be used; if any rule  $A \rightarrow \ddagger$  can be applied, then it has to be applied.)
- each rule of type 14 has priority over  $cZ \rightarrow c\ddagger$ ;  
(If a symbol  $X''_i$  is present, then this means that the computation is finished; we replace this symbol with the corresponding string  $h(x_1)$ )

and we remove the “semi-trap” object  $Z$ ; the rule  $cZ \rightarrow c\ddagger$  cannot be used.)

From the previous explanations, it is easy to see that each derivation in  $G$  can be simulated by a computation in  $\Pi$  and, conversely, each computation in  $\Pi$  corresponds to a derivation in  $G$ . It is worth mentioning that this is possible because we deal with a language over the one-letter alphabet, hence the order of symbols appearing in a sentential form of  $G$  is not important, only their presence matters (exactly as in a multiset). Moreover, at each moment when an occurrence of  $a$  is introduced, it is introduced directly into the output membrane. Nothing else can reach the output membrane. If the derivation is not correctly simulated or it is not terminal, then at least a rule can be further applied, in particular, the rule  $\ddagger \rightarrow \ddagger$  if this symbol was produced. Thus, we can conclude that, because  $L(G) = L(Q)$ , we have  $N(\Pi) = Q$ . ■

In the previous construction we have paid no attention to the propagating feature, but this can be easily done: just add a dummy object  $\#$  which never evolves (and does not arrive in the output membrane) to the right hand member of each rule which diminishes the number of objects:  $cD \rightarrow c$  and  $Z \rightarrow \lambda$ , as well as to rules  $X_i'' \rightarrow h(x_2)$  of type 14, if  $x_2 = \lambda$ . Note also that we never dissolve a membrane, hence this feature is useless in this case.

It is also easy to see that we can generate recursively enumerable relations with transition super-cell systems of degree 2 as those used above: a relation  $Q \subseteq \mathbf{N}^k$  is characterized by the language  $P(Q)$  obtained as the permutation closure of the language  $\{a_1^{n_1} \dots a_k^{n_k} \mid (n_1, \dots, n_k) \in Q\}$ ; starting from a matrix grammar with appearance checking for  $P(Q)$ , the construction above gives a transition super-cell system for  $Q$  (the important observation is again that the order of symbols in the strings of  $P(Q)$  is not relevant, hence we can work with multisets instead of strings).

We do not know which of the inclusions in the diagram in Figure 5 are proper (but at least one should be, because  $Ls(E0L)$  is strictly included in  $RE$ ).

## 8 Super-cell Systems Based on Rewriting

Transition super-cell systems can be interpreted as using no data structure for codifying the information: the numbers are encoded as the cardinality of multisets, hence they are represented in the base one. This can be adequate to a biochemical implementation, but it looks inefficient from a classic point of view. Moreover, in this way we can deal only with problems on numbers, not (directly, without a number codification) with symbolic computations. That is why we look now for representing information by using a data structure of a standard type, *strings*.

Thus, from now on, instead of objects of an atomic type (i.e., without “parts”), we consider objects which can be described by finite strings over a given finite alphabet. The evolution of an object will then correspond to a transformation of the string. In this section we consider transformations in the form of rewriting steps, as usual in formal language theory.

Consequently, the evolution rules are given as rewriting rules.

Assume that we have an alphabet  $V$ . A usual rewriting rule is a pair  $(u, v)$  of words over  $V$  (we give it in the form  $u \rightarrow v$ ). For  $x, y \in V^*$  we write  $x \Longrightarrow y$  iff  $x = x_1 u x_2$  and  $y = x_1 v x_2$ , for some strings  $x_1, x_2 \in V^*$ .

Here, the rules are also provided with indications on the target membrane of the produced string (we do no longer consider the membrane dissolving action, because, similarly to the case of Theorem 1, it will not be necessary in order to obtain computational completeness; of course, if for other purposes it will be useful/necessary to use this action, then it can be introduced in the same way as in the transition super-cell systems). Always we use only context-free rules. Thus, the rules are of the form

$$X \rightarrow v(\text{tar}),$$

where  $\text{tar} \in \{\text{here}, \text{out}, \text{in}_m\}$  (“tar” comes from “target”,  $m$  is the label of a membrane), with the obvious meaning: the string produced by using this rule will go to the membrane indicated by  $\text{tar}$ .

Note the important difference from the way the transition super-cell systems work: a string is now a unique object, hence it passes through membranes as a unique entity, its symbols do not follow different itineraries, as it was possible for the objects in a multiset; of course, in the same region we can have several strings at the same time.

In this way, we obtain a language generating mechanism of the form

$$\Pi = (V, \mu, M_1, \dots, M_n, (R_1, \rho_1), \dots, (R_n, \rho_n), i_0),$$

where  $V$  is an alphabet,  $\mu$  is a membrane structure,  $M_1, \dots, M_n$  are finite languages over  $V$ ,  $R_1, \dots, R_n$  are finite sets of context-free evolution rules,  $\rho_1, \dots, \rho_n$  are partial order relations over  $R_1, \dots, R_n$ , and  $i_0$  is the output membrane.

We call such a system a *rewriting super-cell system*.

The language generated by a system  $\Pi$  is denoted by  $L(\Pi)$  and it is defined as explained in Section 5, with the differences specific to an evolution based on rewriting: we start from an initial configuration of the system and proceed iteratively, by transition steps done by using the rules in parallel, to all strings which can be rewritten, obeying the priority relations, and collecting the strings generated in a designated membrane, the output one.

Note that each string is processed by one rule only, the parallelism refers here to processing simultaneously all available strings by all applicable rules. If several rules can be applied to a string, maybe in several places each, then

we take only one rule and only one possibility to apply it and consider the obtained string as the next state of the object described by the string. It is important to have in mind the fact that the evolution of strings is not independent to each other, but interrelated in two ways: (1) if we have priorities, a rule  $r_1$  applicable to a string  $x$  can forbid the use of another rule,  $r_2$ , for rewriting another string,  $y$ , which is present at that time in the same membrane; after applying the rule  $r_1$ , if  $r_1$  is not applicable to  $y$  or to the string  $x'$  obtained from  $x$  by using  $r_1$ , then it is possible that the rule  $r_2$  can now be applied to  $y$ ; (2) even without priorities, if a string  $x$  can be rewritten for ever, in the same membrane or on an itinerary through several membranes, and this cannot be avoided, then all strings are lost, because the computation never stops, irrespective of the strings collected in the output membrane and which cannot evolve further.

**Remark 2.** It is worth noting the similarities and, mainly, the differences between rewriting super-cell systems and parallel communicating grammar systems with the communication by queries ([14]) or by command ([4]). Both kinds of systems are distributed parallel devices, making an essential use of communication. In the grammar systems case, the component grammars work synchronously and send to each other sentential forms. Here, the synchronization is not obligatory, a component membrane can wait if its rules, if any, are not applicable. More important: the components of a grammar system are always the same, are arranged in the same level, and they can communicate to each other without restrictions (a total graph is available as a communication graph); here the components can be hierarchically arranged in a specified architecture and they can disappear during the computation. Still, the two mechanisms meet each other in the generative power: also the parallel communicating grammar systems characterize the recursively enumerable languages, both when communicating by queries ([5]) and by command ([4], [9]). As a common conclusion we can state the fact that communication is very powerful, irrespective of the ways it is performed.

We denote by  $RSC_n(Pri)$  the family of languages generated by rewriting super-cell systems of degree at most  $n$ ,  $n \geq 1$ , using priorities; when priorities are not used, we replace  $Pri$  with  $nPri$ ; the union of all families  $RSC_n(\alpha)$  is denoted by  $RSC(\alpha)$ ,  $\alpha \in \{Pri, nPri\}$ .

Because we will use below the notion of an ET0L system, we briefly introduce it: such a system is a construct  $G = (V, T, w, P_1, \dots, P_n)$ , such that each  $(V, T, w, P_i)$ ,  $1 \leq i \leq n$ , is an E0L system. One step (parallel) derivation with respect to  $P_i$  is denoted by  $\Longrightarrow_i$  and defined as for E0L systems. The language generated by  $G$  is  $L(G) = \{z \in T^* \mid w \Longrightarrow_{i_1} w_1 \Longrightarrow_{i_2} \dots \Longrightarrow_{i_k} w_k = z, \text{ for some } 1 \leq i_j \leq n, 1 \leq j \leq k\}$ . The family of languages generated by ET0L systems is denoted by  $ET0L$ .

By *ORD* we denote the family of languages generated by context-free ordered grammars (that is, context-free grammars with a partial order relation on the set of rules; a rule can be applied only when no rule of a higher priority can be used).

It is known that  $ET0L \subset ORD \subset RE$ .

**Theorem 2.** *The relations in the diagram in Figure 6 hold, where the arrows indicate inclusions which are not necessarily proper; the inclusion  $CF \subset RSC_2(nPri)$  is proper.*

*Proof.* The inclusions between the *RSC* families follow from the definitions.

The equality  $CF = RSC_1(nPri)$  can be proved in the following way:

For a context-free grammar  $G = (N, T, S, P)$ , we construct the rewriting super-cell system

$$\Pi = (N \cup T, [ ]_1, \{S\}, (P \cup \{A \rightarrow A \mid A \in N\}, \emptyset), 1).$$

A computation is finished only when no rule  $A \rightarrow A$  is applicable, which means that no nonterminal symbol is present in the obtained string, hence the computation corresponds to a terminal derivation in  $G$ .

Conversely, let  $\Pi$  be a rewriting super-cell system of degree one over some alphabet  $V$ . Let  $P$  be the set of all rules appearing in  $\Pi$  and  $L_0$  be the finite set of all strings initially present in the system. Denote by  $T$  the set of symbols  $a \in V$  such that no rule  $a \rightarrow x$  is in  $P$  and by  $N$  the set  $(V - T) \cup \{S\}$ , where  $S$  is a new symbol. A symbol  $A \in V - T$  for which there is no derivation with respect to  $P$  of the form  $A \Longrightarrow^* x$  with  $x \in T^*$  is said to be *poisoned* (there are rules  $a \rightarrow x$  for these symbols, but they never lead to a string of terminals). If there is a string in  $L_0$  which contains a poisoned symbol, then  $L(\Pi) = \emptyset$ . In the opposite case, the context-free grammar  $G = (N, T, S, \{S \rightarrow x \mid x \in L_0\} \cup P)$  clearly generates the language  $L(\Pi)$  (all strings in  $L_0$  leads to strings in  $T^*$ ).

By adding a partial order relation, we obtain in the same way the equality  $ORD = RSC_1(Pri)$  (the set of poisoned symbols does not depend on the order relation among rules: if a rule cannot be applied because a rule with a higher priority is applicable to a non-poisoned symbol, it will be applied later, when the symbol is replaced by a terminal one).

The inclusions  $RE \subseteq RSC_3(Pri)$  and  $MAT \subseteq RSC(nPri)$  are proved in the following two lemmas.

The fact that the family  $RSC_2(nPri)$  contains non-context-free languages is proved by the following rewriting super-cell system:

$$\begin{aligned} \Pi &= (\{A, B, a, b, c\}, [ ]_1 [ ]_2, \emptyset, \{AB\}, (R_1, \emptyset), (R_2, \emptyset), 2), \\ R_1 &= \{B \rightarrow cB(in_2)\}, \\ R_2 &= \{A \rightarrow aAb(out), A \rightarrow ab, B \rightarrow c\}. \end{aligned}$$



$$\begin{aligned}
M_1 &= \{X_0A_0E\}, \text{ while the two other multisets are empty,} \\
R_1 &= \{r_\alpha : \alpha \rightarrow \alpha \mid \alpha \in V - T, \alpha \neq E\} \\
&\cup \{r_0 : E \rightarrow \lambda(in_2), \dagger \rightarrow \dagger\} \\
&\cup \{X \rightarrow Y_i(in_2) \mid \\
&\quad m_i : (X \rightarrow Y, A \rightarrow x) \text{ is a matrix of types 2 or 4}\} \\
&\cup \{X \rightarrow Y_i(in_3) \mid m_i : (X \rightarrow Y, A \rightarrow \dagger) \text{ is a matrix of type 3}\} \\
&\cup \{X \rightarrow X'_i x_1(in_2), X'_i \rightarrow \lambda \mid \\
&\quad m_i : (X \rightarrow X'x_1, A \rightarrow x_2) \text{ is a matrix of type 4}'\} \\
&\cup \{Y_i \rightarrow Y, Y'_i \rightarrow Y \mid Y \in N_1, 1 \leq i \leq k\}, \\
\rho_1 &= \{r_\alpha > r_0 \mid \alpha \in V - T, \alpha \neq E\}, \\
R_2 &= \{r_i : Y_i \rightarrow Y_i, r'_i : A \rightarrow x(out) \mid \\
&\quad m_i : (X \rightarrow Y, A \rightarrow x) \text{ is a matrix of types 2 or 4}\} \\
&\cup \{r_i : X'_i \rightarrow X'_i, r'_i : A \rightarrow x_2(out) \mid \\
&\quad m_i : (X \rightarrow X'x_1, A \rightarrow x_2) \text{ is a matrix of type 4}'\} \\
\rho_2 &= \{r_i > r'_j \mid i \neq j, \text{ for all possible } i, j\}, \\
R_3 &= \{p_i : Y_i \rightarrow Y'_i, p'_i : Y'_i \rightarrow Y_i, p''_i : A \rightarrow \dagger(out) \mid \\
&\quad m_i : (X \rightarrow Y, A \rightarrow \dagger) \text{ is a matrix of type 3}\} \\
&\cup \{p_0 : E \rightarrow E(out)\}, \\
\rho_3 &= \{p''_i > p_i, p_i > p_0 \mid \text{for all possible } i\}.
\end{aligned}$$

The system works as follows. Observe first that the rules  $\alpha \rightarrow \alpha$  from membrane 1 change nothing, can be used for ever, and prevent the use of the rule  $E \rightarrow \lambda$ , which sends the string to membrane 2, the output one.

Assume that in membrane 1 we have a string of the form  $XwE$  (initially, we have here the string  $X_0A_0E$ ). In membrane 1 one chooses the matrix to be simulated,  $m_i$ , and one simulates its first rule,  $X \rightarrow Y$ , by introducing  $Y_i$ ; the string is sent to membrane 2 if we deal with a matrix of types 2 or 4 (without a rule which has to be applied in the appearance checking mode), and to membrane 3 if we have to simulate a matrix of type 3.

In membrane 2 we can use the rule  $Y_i \rightarrow Y_i$  for ever. The only way to quit this membrane is by using the rule  $A \rightarrow x$  appearing in the second position of a matrix of type 2 or 4. Due to the priority relation, this matrix should be exactly  $m_i$  as specified by the subscript of  $Y_i$ . Therefore, we can continue the computation only when the matrix is correctly simulated.

The process is similar in membrane 3: The rules  $Y_i \rightarrow Y'_i, Y'_i \rightarrow Y_i$  can be used for ever. We can quit the membrane either by using a rule  $A \rightarrow \dagger(out)$  or by using the rule  $E \rightarrow E(out)$ . In the first case the computation will never halts. Because of the priority relation, such a rule must be used if the corresponding symbol  $A$  appears in the string. If this is not the case, then the rule  $Y_i \rightarrow Y'_i$  can be used. If we now use the rule  $Y'_i \rightarrow Y_i$ , then we get

nothing. If we use the rule  $E \rightarrow E(out)$ , and this is possible because  $Y_i$  is no longer present, then we send out a string of the form  $Y_i'wE$ .

In membrane 1 we replace  $Y_i$  or  $Y_i'$  by  $Y$ , and thus the process of simulating the use of matrices of types 2, 3, 4 can be iterated.

A slightly different procedure is followed for the matrices of type 4'; they are of the form  $m_i : (X \rightarrow X'x_1, A \rightarrow x_2)$ . In membrane 1 we use  $X \rightarrow X'x_1(in_2)$ , which already introduces the substring  $x_1$ , and the string arrives in membrane 2. Again the only way to leave this membrane is by using the associated rule  $A \rightarrow x_2(out)$ . In membrane 1 we have to apply  $X_i' \rightarrow \lambda$ . If no symbol different from  $E$  and terminals is present, then we can apply the rule  $E \rightarrow \lambda(in_2)$ . Thus, a terminal string is sent to membrane 2, where no rewriting can be done, the computation stops. If any nonterminal symbol is still present, then the computation will never halt, because of the rules  $\alpha \rightarrow \alpha$  from membrane 1.

Therefore, we collect in the output membrane exactly the terminal strings generated by the grammar  $G$ , that is  $L(G) = L(\Pi)$ . ■

**Lemma 4.**  $MAT \subseteq RSC(nPri)$ .

*Proof.* Let  $M = (N, T, S, P)$  be a matrix grammar without appearance checking. Assume that  $G$  is in the normal form used also in the proofs above (this time, the matrices of type 3 are missing), with the matrices labeled in a one-to-one manner. We construct a rewriting super-cell system  $\Pi$  in the following way. In the skin membrane we introduce the initial string  $X_0A_0E$ , where  $E$  is a new symbol, and rules of the form  $X \rightarrow z(in_i)$  for each matrix  $m_i : (X \rightarrow z, A \rightarrow x)$  in  $P$ , terminal or not. A membrane with the label  $i$  will contain the unique rule  $A \rightarrow x(out)$ . In this way, the use of matrices is correctly simulated by the way the strings are circulated among membranes. Note that if we do not use the rule of membrane  $i$ , then we cannot leave the membrane, hence the output membrane will remain empty. A special membrane, labeled with  $O$ , is the output one; in this membrane we put all the rules  $A \rightarrow A$ , for  $A \in N_1 \cup N_2$ . In the skin membrane we also consider the rule  $E \rightarrow \lambda(in_O)$ . It can send a string to the output membrane in any moment, but the computation halts only if the string is terminal. The details of this construction are left to the reader. It is obvious that we have  $L(\Pi) = L(G)$ . ■

Several problems are *open* in this area: Is the hierarchy  $RSC_n(nPri)$  an infinite one? Is the result  $RE \subseteq RSC_3(Pri)$  optimal? Is the inclusion  $MAT \subseteq RSC(nPri)$  proper? (The difficulty in proving that  $RSC(nPri) \subseteq MAT$  lies in the dependence between the evolution of the words initially placed in a rewriting super-cell system: even if a string has reached the output membrane and it cannot further evolve, in order to accept it we have to make sure that no other string present in the system can further evolve. This can easily be controlled in a matrix grammar with appearance checking, but we see no way to do it without appearance checking.)

## 9 Splicing Super-cell Systems

We now relate the idea of computing with membranes to another important natural computing area, that of DNA computing. Specifically, we consider super-cell systems with objects in the form of strings and with the evolution rules based on the splicing operation introduced in [8]. We first define this operation.

Consider an alphabet  $V$  and two symbols  $\#, \$$  not in  $V$ . A *splicing rule* over  $V$  is a string  $r = u_1\#u_2\$u_3\#u_4$ , where  $u_1, u_2, u_3, u_4 \in V^*$ . For such a rule  $r$  and for  $x, y, w \in V^*$  we define

$$(x, y) \vdash_r w \quad \text{iff} \quad \begin{aligned} x &= x_1u_1u_2x_2, \quad y = y_1u_3u_4y_2, \\ w &= x_1u_1u_4y_2, \quad \text{for some } x_1, x_2, y_1, y_2 \in V^*. \end{aligned}$$

(One cuts the strings  $x, y$  in between  $u_1, u_2$  and  $u_3, u_4$ , respectively, and one concatenates the “first half” of  $x$  with the “second half” of  $y$ .) We say that we *splice* the strings  $x, y$  at the *sites*  $u_1u_2, u_3u_4$ , respectively. When  $r$  is understood, we write  $\vdash$  instead of  $\vdash_r$ . For clarity, we usually indicate by a vertical bar the place of splicing:  $(x_1u_1|u_2x_2, y_1u_3|u_4y_2) \vdash x_1u_1u_4y_2$ .

Based on this operation, language generating devices were introduced: start from a given set of strings and apply to them iteratively the splicing rules in a given set. We obtain what is called an *H system*. If a terminal alphabet is considered, then we obtain an *extended H system*. It is known that extended H systems with finite sets of axioms and finite sets of rules characterize the regular languages and that systems with certain controls on the use of rules or with certain distributed architectures characterize the recursively enumerable languages. Comprehensive details can be found in [12].

Because we need a string-to-string transformation, we shall consider here a variant of the relation  $\vdash$ , as a binary relation.

Specifically, with each splicing rule  $r : u_1\#u_2\$u_3\#u_4$  over a given alphabet  $V$  we associate a string  $z \in V^*$ . For  $x, y \in V^*$  we write

$$x \Longrightarrow_{(r,z)} y \quad \text{iff either } (x, z) \vdash_r y \text{ or } (z, x) \vdash_r y.$$

When  $(r, z)$  is understood, we write simply  $\Longrightarrow$  instead of  $\Longrightarrow_{(r,z)}$ .

Such transformations can be used for defining transitions in super-cell systems with string-objects.

A *splicing super-cell system* (over a given alphabet  $V$ ) is a super-cell system  $\Pi$  with strings as objects, with evolution rules given in the form  $(r, z)tar$ , where  $r$  is a splicing rule over  $V$ ,  $z \in V^*$ , and  $tar$  is a target indication for the resulting string, one of *here*, *out*, *in<sub>m</sub>*. (As usual, the

indication *here* will be omitted when writing a system.) With respect to such a rule we define a relation  $x \Longrightarrow_{(r,z)} y(\text{tar})$  as mentioned above. Using this relation, we define the transition between configurations, taking into consideration also a possible priority relation among evolution rules. (We do not provide the membrane dissolving action, because it is again not necessary for obtaining computational completeness.) A computation is correctly finished in the same conditions as in the previous sections: no further move is possible, one elementary membrane is designated as the output one. The language generated by  $\Pi$  is the set of strings placed in the output membrane at the end of correctly finished computations.

We denote by  $SSC_n(Pri)$  the family of languages generated by splicing super-cell systems of degree at most  $n$ ,  $n \geq 1$ , using priorities; when priorities are not used, we replace  $Pri$  with  $nPri$ ; the union of all families  $SSC_n(\alpha)$  is denoted by  $SSC(\alpha)$ ,  $\alpha \in \{Pri, nPri\}$ .

We also denote by  $EH$  the family of languages generated by extended splicing systems and by  $EH(Ord)$  the family of languages generated by extended splicing systems with a priority on the set of rules (we use a splicing rule for splicing two strings only if no rule with a higher priority can be used for splicing these strings).

It is known that  $REG = EH$  and  $EH(Ord) = RE$ .

Rather expected, in view of the results in the previous sections and in [12], we have the following result:

**Theorem 3.** *The relations in the diagram in Figure 7 hold, where the arrows indicate inclusions which are not necessarily proper; the family  $SSC_2(nPri)$  contains non-regular languages, while  $SSC_3(nPri)$  contains languages which are not in the family MAT.*

*Proof.* The inclusions follow from the definitions. The equality  $EH(Ord) = RE$  (see the proof of Theorem 8.3 in [12], for checking that the halt condition from splicing super-cell systems is fulfilled by the ordered extended splicing system constructed for simulating a given type-0 grammar) implies the collapse of the hierarchy  $SSC_n(Pri)$  at its first level.

The inclusion  $RE \subseteq SSC_4(nPri)$  is proved in the next lemma.

In order to show that  $SSC_2(nPri)$  contains non-regular languages, let us consider the system:

$$\begin{aligned} \Pi &= (\{a, b, d\}, [_1[_2]_2]_1, \{dabd\}, \emptyset, (R_1, \emptyset), (R_2, \emptyset), 2), \\ R_1 &= \{(d\#a\$da\#Z, daZ)in_2\}, \\ R_2 &= \{(b\#d\$Z\#bd, Zbd)out, (b\#d\$Z\#b, Zb)\}. \end{aligned}$$

It is easy to see that strings of the form  $da^n b^n d$  (initially, we have  $n = 1$ ) get one more occurrence of  $a$  in the skin membrane, are passed to the output

membrane, here one more occurrence of  $b$  is added, the process is iterated, and, at any moment, in the output membrane we can use the rule  $b\#d\$Z\#b$  and the right occurrence of the symbol  $d$  is removed. No further splicing is possible, hence the computation is correctly finished. Consequently, we get

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

This is not a regular language.

$$\begin{array}{c}
EH(Ord) = SSC_1(Pri) = SSC(Pri) \\
= RE = SSC_4(nPri) = SSC(nPri) \\
\uparrow \\
SSC_3(nPri) \\
\uparrow \\
SSC_2(nPri) \\
\uparrow \\
SSC_1(nPri)
\end{array}$$

**Figure 7.** The hierarchy of families  $SSC_n(\alpha)$ .

For the similar assertion  $SSC_3(nPri) - MAT \neq \emptyset$  we use the following system:

$$\begin{aligned}
\Pi &= (V, \mu, \{XabY\}, \emptyset, \emptyset, (R_1, \emptyset), (R_2, \emptyset), (R_3, \emptyset), 3), \\
V &= \{a, b, c, X, Y, Z\}, \\
\mu &= [_1[_2]_2[_3]_3]_1, \\
R_1 &= \{(X\#Z\$Xa\#, XZ)in_2, (X\#Z\$Xb\#, XZ)in_3, \\
&\quad (c\#Z\$Xb\#, cZ)in_3, (c\#Z\$c\#, cZ)\}, \\
R_2 &= \{(\#Y\$Z\#aaY, ZaaY)out\}, \\
R_3 &= \{(\#Y\$Z\#bY, ZbY)out\}, (\#Y\$Z\#c, Zc), (X\#Z\$X\#, XZ)\}.
\end{aligned}$$

Assume that we have a string of the form  $Xa^i ba^j Y$  in membrane 1; initially, we have  $i = 1, j = 0$ . If  $i \geq 1$ , then we have to use the rule  $X\#Z\$Xa\#$  and the string  $Xa^{i-1} ba^j Y$  is sent to membrane 2. There is only one rule here, hence we get the string  $Xa^{i-1} ba^{j+2} Y$ , which is sent back to membrane 1. In this way, we will eventually obtain the string  $Xba^{j+2i} Y$ . If we use the rule  $X\#Z\$Xb\#$ , then we obtain the string  $Xa^{j+2i} Y$  which is sent to membrane 3. If we use the rule  $\#Y\$Z\#c$ , then we obtain the string

$Xa^{j+2i}c$  which will remain in membrane 3 and can be processed for ever by using the rule  $X\#Z\$X$ . The only way to continue in such a way that the computation will be successfully finished is to use the rule  $\#Y\$Z\#bY$ ; the obtained string  $Xa^{j+2i}bY$  is returned to membrane 1 and the process is iterated. In this way, the number of occurrences of  $a$  is doubled at each passing of the string through membrane 3.

When in membrane 1 we have a string  $Xba^jY$ , then we can also use the rule  $c\#Z\$Xb\#$ , the marker  $X$  is replaced with  $c$  and the string is sent to membrane 3. If we apply here the rule  $\#Y\$Z\#bY$ , then the string  $ca^jY$  arrives in membrane 1 and it will be processed for ever by using the rule  $c\#Z\$c\#$ . Thus, in order to halt, in membrane 3 we have to use the rule  $\#Y\$Z\#c$ , that is we get the string  $ca^jY$ . No further rule can be applied.

In conclusion, we obtain the language  $L(\Pi) = \{ca^{2^n}c \mid n \geq 1\}$ , which is not in the family  $MAT$ : each one-letter language in  $MAT$  is regular, see [7], and  $L(\Pi)$  can be mapped to the non-regular language  $\{a^{2^n} \mid n \geq 1\}$  by a morphism;  $MAT$  is closed under morphisms, see [6]. ■

**Lemma 5.** (The Computational Completeness Lemma for Splicing Super-cell Systems)  $RE \subseteq SSC_4(nPri)$ .

*Proof.* Let  $G = (N, T, S, P)$  be a type-0 Chomsky grammar. Assume that  $N \cup T = \{D_1, \dots, D_n\}$  and take a further symbol,  $B$ , also denoted by  $D_{n+1}$ .

We construct the following splicing super-cell system (of degree 4 and without priorities):

$$\begin{aligned}
\Pi &= (V, \mu, M_1, M_2, M_3, M_4, (R_1, \emptyset), (R_2, \emptyset), (R_3, \emptyset), (R_4, \emptyset), 4), \\
V &= \{N \cup T \cup \{Z, B\} \cup \{X_i, Y_i \mid 0 \leq i \leq n+1\}, \\
\mu &= [{}_1[{}_2]{}_2[{}_3]{}_3[{}_4]{}_4]{}_1, \\
M_1 &= M_3 = M_4 = \emptyset, \\
M_2 &= \{XSBY\}, \\
R_1 &= \{(X_i D_i \# Z \$ X \#, X_i D_i Z)in_2 \mid 1 \leq i \leq n+1\} \\
&\cup \{(X_{i-1} \# Z \$ X_i \#, X_{i-1} Z)in_2 \mid 2 \leq i \leq n+1\} \\
&\cup \{(X \# Z \$ X_1 \#, X Z)in_3, (\#BY\$Z\#BY', ZBY')in_2, \\
&\quad (\#Y''\$Z\#, Z)in_4, (\#\dagger\$Z\#\dagger, Z\dagger)\}, \\
R_2 &= \{(\#uY\$Z\#vY, ZvY) \mid u \rightarrow v \text{ is a rule from } P\} \\
&\cup \{(\#D_i Y \$ Z \# Y_i, ZY_i)out, (\#Y_i \$ Z \# Y_{i-1}, ZY_{i-1})out \mid \\
&\quad 1 \leq i \leq n+1\} \\
&\cup \{(\#D_i Y \$ Z \# Y'_i, ZY'_i)out \mid D_i \in T \cup \{B\}, 1 \leq i \leq n+1\} \\
&\cup \{(\#Y'_i \$ Z \# Y'_{i-1}, ZY_{i-1})out \mid 1 \leq i \leq n+1\} \\
&\cup \{(\#Y_0 \$ Z \#\dagger, Z\dagger)out\},
\end{aligned}$$

$$\begin{aligned}
R_3 &= \{(\#Y_0\$Z\#Y, ZY)out, \\
&\quad (\#Y_0\$Z\#Y', ZY')out, (\#BY_0\$Z\#Y'', ZY'')out\} \\
&\cup \{(\#Y_i\$Z\#\dagger, Z\dagger)out, (\#Y_i'\$Z\#\dagger, Z\dagger)out \mid 1 \leq i \leq n+1\}, \\
R_4 &= \{(\#Z\$X\#, Z)\}.
\end{aligned}$$

The system works as follows. There is only one string in the initial configuration (namely, in membrane 2),  $XBSY$ , which introduces the axiom of  $G$ , together with the new symbol  $B$ , and the end markers  $X, Y$ . Therefore, always we will have only one string in the system.

Assume that we have a string of the form  $XwY$  in membrane 2. If we apply a splicing rule  $\#uY\$Z\#vY$ , then we simulate the use of a rule from  $P$  at the end of the string,  $Xw'uY \implies Xw'vY$ , and this corresponds to  $w'u \implies w'v$  (modulo an occurrence of  $B$  in the string  $w'$ ) in  $G$ . The string remains in membrane 2.

If we perform a splicing

$$(Xw'|D_iY, Z|Y_i) \vdash Xw'Y_i,$$

for some  $i = 1, 2, \dots, n+1$ , then the string exits the membrane. In the skin membrane, we have only the possibility to use a splicing rule of the form  $X_jD_j\#Z\$X\#$ . In this way, we get a string  $X_jD_jw'Y_i$ , which is again passed to membrane 2. Here, we have to decrease by one the subscript of the right end marker. The obtained string,  $X_jD_jw'Y_{i-1}$  is sent to the skin membrane, where the subscript of the left end marker is decreased by one; the obtained string is sent to membrane 2 and the process is repeated. When in membrane 2 we get  $X_kD_jw'Y_0$ , this string is again passed to the skin membrane; if  $k = 1$ , then the rule  $X\#Z\$X_1\#$  is used here and the resulting string is passed to membrane 3. Only strings with the subscript of  $Y$  equal to zero can be processed in membrane 3 without introducing the trap-symbol  $\dagger$  in the currently produced string (once introduced, this symbol will be processed for ever in the skin membrane by the rule  $\#\dagger\$Z\#\dagger$ , hence the computation can never be finished; note that the string is of the form  $Xw\dagger$ , hence no other rule but  $\#\dagger\$Z\#\dagger$  can be used in the skin membrane; using this rule, we do not change the string:  $(Xw|\dagger, Z|\dagger) \vdash Xw\dagger$ ). If the string  $X_kD_jw'Y_0$  is passed from membrane 2 to the skin membrane and  $k \geq 2$ , then the rule  $X_{k-1}\#Z\$X_k\#$  is used and the string  $X_{k-1}D_jw'Y_0$ , with  $k-1 \geq 1$ , is sent to membrane 2. The only rule in membrane 2 which can be applied is  $\#Y_0\$Z\#\dagger$ , which introduces the trap-object  $\dagger$ . The computation is never terminated.

If a string  $X_1D_jw'Y_k$  with  $k \geq 1$  is produced in membrane 2 and sent to membrane 1, here we can only apply the rule  $X\#Z\$X_1\#$  and the string  $XD_jw'Y_k$  is sent to membrane 3. No rule but  $\#Y_i\$Z\#\dagger$  is here applicable, hence the computation never terminates.

Consequently, in order to finish correctly the computation, the subscripts of the end markers have to reach the value zero at the same time, that is,

$i = j$ . This means that the symbol  $D_i$  which was cut from the right hand end of the string has been reproduced in the left end of the string. Note that the symbol  $B$  can be moved from an end of the string to the other one like any symbol from  $N \cup T$ .

In this way, the string is circularly permuted, making possible the simulation of rules of  $G$  in any position. In each moment, there is exactly one occurrence of the symbol  $B$ , indicating the beginning of the sentential form of  $G$  simulated by our system: if in  $\Pi$  we have generated the string  $Xw_1Bw_2Y$ , then the string  $w_2w_1$  is a sentential form of  $G$ , and conversely.

When the rule  $\#BY\$Z\#BY'$  is used in the skin membrane, the resulting string, of the form  $XwBY'$  is sent to membrane 2; no simulation of a rule in  $P$  is possible from now on, but only circular permutations. Such circular permutations can be performed as above, using the primed right end markers  $Y'_i$  instead of  $Y_i$ ,  $0 \leq i \leq n + 1$ . However, it is important to note that only symbols which are terminal with respect to  $G$  can be moved. In any moment when in membrane 3 we have a string of the form  $XwBY'_0$  (received from the skin membrane), we can choose to replace  $BY'_0$  by  $Y''$ . The fact that  $B$  is in the right hand end of the string tells us that  $w$  is a sentential form of  $G$  (in a non-permuted form). Moreover, because  $B$  is again in the right hand end, this implies that at least one circular permutation of the string  $wB$  has been done since  $Y'$  has been introduced, that is, the string  $w$  is terminal. The obtained string,  $Xw$ , is sent to membrane 4, where the left marker is removed. No rule can now be applied, the computation stops with the output  $w$ . Because we know that this string can be generated by  $G$ , we have the equality  $L(G) = L(\Pi)$ . ■

We do not know whether or not the degree of the system in this lemma can be further decreased.

If we provide a splicing super-cell system  $\Pi$  with a terminal alphabet  $T$  and we define the language generated as consisting of all strings over  $T$  collected in the output membrane at the end of halting computations, then systems of degree three can characterize the recursively enumerable languages: in the proof of Lemma 5, membrane 4 is no longer necessary, while membrane 3 can be considered the output one (a rule able to remove  $X_1B$  in the skin membrane and a rule for removing  $Y_0$  in the output membrane should be considered). We can formulate this observation also in the following form:

**Corollary 1.** *Each recursively enumerable language  $L \subseteq T^*$  can be written in the form  $L = L' \cap T^*$ , where  $L' \in SSC_3(nPri)$ .*

## 10 Final Remarks

We have introduced a new computability model, called a super-cell system, based on the evolution of objects in a membrane structure. The objects can be single symbols, or strings of symbols; in the latter case, the evolution is defined in terms of string transformations. We have considered here rewriting and splicing as underlying operations with strings. In all three cases, basic (transition) super-cell systems, rewriting super-cell systems, and splicing super-cell systems, we obtain computational completeness, characterizations of recursively enumerable sets of natural numbers (of relations, too) and of recursively enumerable languages by systems of a rather simple form.

From the proofs of these results we can draw an important observation: the computational completeness is obtained without making use of the parallel synchronized evolution of objects and membranes. Synchronization, in the sense of an universal clock, is assumed in the definition of transitions in super-cell systems, but it does not appear in the proofs of the three computational completeness lemmas: in the case of transition super-cell systems we have only one working membrane, in the case of rewriting and splicing super-cell systems we always have only one string in the system, hence the synchronization has no object.

Many open problems and research topics are formulated.

Many further questions naturally arise in this framework. We only mention some of them. Of definite interest is to consider *deterministic* super-cell systems, having in each moment at most one possible transition. This might be important in the case that such devices would be implemented in “reliable” media, which behave deterministically. Of course, in biochemical-like media, where a huge parallelism is possible, the nondeterminism could be useful, because by using it we can simulate the deterministic parallelism (with a high probability, working nondeterministically on a large number of “processors” we can get the result of a parallel exploration of the search space).

We have considered above that when a membrane is dissolved, only the objects survive, the rules of the former membrane are lost. This, of course, can be changed. Moreover, in the same way as the objects evolve, we may assume that also the rules evolve. Still more: also the membranes can evolve, not only by disappearing under the influence of certain object evolution rules, by also in other modes. Creating new membranes can be done either by usual action rules (instead of a symbol  $\delta$ , consider a symbol  $\tau_X$ , with the meaning “create a new membrane, labeled with  $X$ ”) or by membrane evolution rules (duplication, separation in two distinct membranes, etc.) Some technical problems appear here, concerning the contents of the new membranes, the objects and the rules to be put into them (certain “inheritance principles” should be considered). A small jungle of variants can be produced in this way.

We finish by stressing again the importance of parallelism in super-cell systems, appearing at two levels in transition and splicing super-cell systems and, possibly, at three levels in rewriting super-cell systems: we can also use the rules in parallel in the sense of Lindenmayer systems (each symbol of a string which can be rewritten should be rewritten; then, *all* strings are rewritten simultaneously, in *all* membranes of the system). The influence of parallelism on the time complexity of computations in super-cell systems is a question of a basic interest. It is highly conceivable that when rules for producing new membranes are provided, by creating an exponential number of membranes, an essential speed-up can be obtained (perhaps, even polynomial time computations, done in parallel, of solutions to exponential problems).

An important problem, not mentioned in the formal framework above, concerns the possibility of implementing super-cell systems, either in electronic media or in biological media. A related, double, problem is (1) to find specific computing problems which can be solved in this way, and (2) to find natural processes (for instance, biological) which can be considered as counterparts of membrane structures we used here, or, at least, similar to the operations used in our super-cell systems (for instance, moving objects through membranes in a well specified manner, dissolving membranes – producing “holes” in them, etc.). The answer to such questions can direct the theoretical studies to the most promising and practically relevant direction.

**Note.** This work was supported by the Academy of Finland, Project 137358. Useful discussions with J. Dassow, H. Fernau, M. Holzer, A. Ma-teescu, I. Petre, K. Reinhardt, G. Rozenberg, and A. Salomaa are gratefully acknowledged.

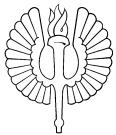
## References

- [1] L. M. Adleman, On constructing a molecular computer, in *DNA Based Computers*, Proc. of a DIMACS Workshop, Princeton, 1995, Amer. Math. Soc., 1996 (R. J. Lipton, E. B. Baum, eds.), 1–22.
- [2] M. Amos, *DNA Computing*, PhD Thesis, Univ. of Warwick, Dept. of Computer Sci., 1997.
- [3] E. Csuhaj-Varju, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.
- [4] E. Csuhaj-Varju, J. Kelemen, Gh. Păun, Grammar systems with WAVE-like communication, *Computers and AI*, 15, 5 (1996), 419–436.

- [5] E. Csuhaj-Varju, G. Vaszil, Context-free PC grammar systems are computationally complete, *Theoretical Computer Sci.*, 1999.
- [6] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
- [7] D. Hauschildt, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, *Acta Inform.*, 31 (1994), 719–728.
- [8] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology*, 49 (1987), 737–759.
- [9] L. Ilie, A. Salomaa, 2-testability and relabeling produce everything, *J. of Computer and System Sciences*, 56 (1998), 253–262.
- [10] R. J. Lipton, Speeding up computations via molecular biology, in *DNA Based Computers*, Proc. of a DIMACS Workshop, Princeton, 1995 (R. J. Lipton, E. B. Baum, eds.), Amer. Math. Soc., 1996, 67–74.
- [11] V. Manca, String rewriting and metabolism: A logical perspective, in *Computing with Bio-Molecules. Theory and Experiments* (Gh. Păun, ed.), Springer-Verlag, Singapore, 1998, 36–60.
- [12] Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Heidelberg, 1998.
- [13] Gh. Păun, A. Salomaa, eds., *Grammatical Models of Multi-Agent Systems*, Gordon and Breach, London, 1998.
- [14] Gh. Păun, L. Sântean, Parallel communicating grammar systems: the regular case, *Ann. Univ. Buc., Matem.-Inform. Series*, 38, 2 (1989), 55–63.
- [15] G. Rozenberg, A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.
- [16] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.

**Turku Centre for Computer Science**  
**Lemminkäisenkatu 14**  
**FIN-20520 Turku**  
**Finland**

<http://www.tucs.abo.fi>



**University of Turku**  
• Department of Mathematical Sciences



**Åbo Akademi University**  
• Department of Computer Science  
• Institute for Advanced Management Systems Research



**Turku School of Economics and Business Administration**  
• Institute of Information Systems Science