Editing Configurations of P Systems

Erzsébet Csuhaj-Varjú[1], Antonio Di Nola[2], Gheorghe Păun[3,4], Mario J. Pérez–Jiménez[4], György Vaszil[1]

[1] Computer and Automation Institute
Hungarian Academy of Sciences
Kende utca 13–17, H-1111 Budapest, Hungary
E-mail: {csuhaj, vaszil}@sztaki.hu
[2] Department of Mathematics and Computer Science
University of Salerno
84081 Baronissi, Salerno, Italy
E-mail: adinola@unisa.it
[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
E-mail: george.paun@imar.ro
[4] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {gpaun, marper}@us.es

Summary. This paper proposes and preliminarily investigates the possibility of transforming a configuration (membrane structure and multisets of symbol-objects present in the compartments of this membrane structure) of a P system into another configuration, by means of a given set of rules acting both on the membranes and on the multisets of objects. Although such a transformation can be obtained during a computation of a P system, we consider it as a goal *per se*, as a pre-computation phase, when the system itself is built. In this framework, several important topics appear, such as the edit-distance between configurations (with respect to a given set of editing rules; actually, this is a weak metric, because it is not necessarily symmetric), normal forms, reachability, existence of single configurations from which a given family of configurations can be constructed, etc. We investigate here only a few of these questions; the paper is mainly devoted to formulating problems in the new framework, calling attention to the possible extensions and usefulness of the present approach.

1 Introduction

Membrane computing aims to abstract computing models from the cell structure and functioning, [14], [15]. With such a goal, the main research topics of the domain concern the computing power (comparing the power of the models obtained with

inspiration from the cell biology with the power of Turing machines and of their restrictions) and the computing efficiency (solving computationally hard problems in a feasible time, by making use of a time-space trade-off which is made possible by various ways of producing an exponential working space in a linear time). The investigations were rather successful from these points of view – see details in [15] and in the web page from `http://psystems.disco.unimib.it`.

Roughly speaking, a cell-like P system consists of a membrane structure (a hierarchical arrangement of membranes), in the compartments of which one places multisets of symbol-objects; these two elements, the membrane structure and the multisets of objects present in its compartments, form a *configuration* of a system at a given time. In the compartments or associated with the membranes there also are sets of rules, according to which the objects and the membrane structure evolve; otherwise stated, by using these rules we obtain transitions among configurations. A sequence of transitions is called a computation. A computation is successful only if it halts, and with a halting computation we associate a result, e.g., in the form of the vector describing the multiplicity of objects from a given compartment of the halting configuration.

Here we switch the focus from computations to configurations, and we consider the problem of passing from a configuration to another configuration with the help of a given set of rules. This is similar to looking for transitions which link the two configurations, but the interest is different: we do not care about the computation itself (halting/non-halting) or about its result, while the rules we consider are mainly devoted to handling membrane structures. In some sense, our approach is directed to founding a "membrane calculus", as attempted also in [4], in another context.

Actually, there are several motivations for this kind of investigation. For instance, as already mentioned in [6] (the present paper can also be considered as a continuation of [6]), if a good (weak) metric related to the time of passing from a configuration to another configuration, with respect to the rules of a given P system, can be found, then it can be useful in a heuristic strategy to solving hard problems, based on the A^* algorithm from [13]. Then, taking into account that a P system is nothing else than an initial configuration and given sets of rules associated with membranes, constructing the initial configuration is a way to construct the system itself; otherwise stated, we can consider a specific set of rules for the pre-computing case, when the computing model itself is build, and other rules for the computation. This can have interesting consequences, for instance, in building a family of P systems associated with a decidability problem, in order to solve it (the particular systems from the family solve particular instances of the problem – see details, e.g., in [16]); in the standard computational complexity theory, this construction is done in polynomial time by a Turing machine, but the problem was formulated several times to have the P systems solving a problem constructed by another P system, so that the whole procedure is "uniformly bio-inspired". Links with other areas (such as the theory of abstract families of languages, graph theory, or evolutionary computing) will be mentioned below.

However, as already said, the present paper is only a preliminary exploration of the "membrane calculus" we propose, with several results and much more research topics formulated.

2 Preliminary Definitions

We start by fixing some notation and terminology. For related/further details (from membrane computing), we refer to [15] and to the papers available in the web page mentioned above.

An *alphabet* is a finite and non-empty set of abstract symbols. For an alphabet A we denote by A^* the set of all strings of symbols from A, including the empty string, denoted by λ; the set $A^* - \{\lambda\}$, of non-empty strings over A, is denoted by A^+. The length of a string $w \in A^*$ is denoted by $|w|$.

A *multiset* over an alphabet A is a mapping from A to $\mathbf{N}$, the set of natural numbers. We represent the multisets by strings from A^*; the number of occurrences of a symbol $a \in A$ in a string w, denoted by $|w|_a$, represents the multiplicity of a in the multiset represented by w (hence all strings obtained by permuting symbols in a string w represent the same multiset). Because of this direct correspondence between multisets over A and strings over A, we will use the terms "multiset" and "string" interchangeably (for instance, we speak about the "length" of a multiset, with the obvious meaning that this is the total multiplicity of elements in the multiset, equal with the length of the string which represents the multiset).

In one of the following sections we need the notion of a (non–deterministic) *register machine*. Such a device consists of a given number of registers each of which can hold an arbitrarily large non-negative integer number, and a set of labelled instructions which specify how the numbers stored in registers can change and which instruction should follow after any used instruction.

Definition 1. *A (non–deterministic) register machine is a construct* $M = (m, Lab, l_0, l_h, P)$*, where* $m \geq 1$ *is the number of registers,* Lab *is a nonempty finite set (whose elements are called instruction labels),* $l_0 \in Lab$ *is the start label,* $l_h \in Lab - \{l_0\}$ *is the halt label (assigned to instruction* `HALT`*), and* P *is a finite set of (labelled) instructions of one of the following three forms:*

- $l_1 : (\texttt{ADD}(r), l_2, l_3)$*, for* $l_1, l_2, l_3 \in Lab, 1 \leq r \leq m$,
- $l_1 : (\texttt{SUB}(r), l_2, l_3)$*, for* $l_1, l_2, l_3 \in Lab, 1 \leq r \leq m$,
- $l_h : \texttt{HALT}$ *(the halt instruction).*

Each label from Lab *labels at most one instruction of* P*, and* l_0 *labels one instruction of* P*.*

A *state* of a register machine $M = (m, Lab, l_0, l_h, P)$ is an m-tuple $(a_1, \ldots, a_m)$ of natural numbers. We interpret the number a_i as the content of register i. A *snapshot* or *instantaneous description* of a register machine $M = (m, Lab, l_0, l_h, P)$

is a pair (l, s), where $l \in Lab$ is a label, and s is a state of M. Intuitively, the label l indicates what is the instruction which is about to be executed.

A snapshot $\sigma = (l, s)$ of a register machine M is called *initial* if $l = l_0$ and $s = (0, 0, \ldots, 0)$, and it is called *halting* if $l = l_h$.

Definition 2. *Let $M = (m, Lab, l_0, l_h, P)$ be a register machine, and $\sigma = (l, s)$ with $s = (a_1, \ldots, a_m)$ a non–halting snapshot of M. A successor of σ is a snapshot $\sigma' = (l', s')$ defined as follows:*

1. *If l labels an instruction $(\mathtt{ADD}(r), l_2, l_3)$, then $s' = (a_1, \ldots, a_r + 1, \ldots, a_m)$, and $l' = l_2$ or $l' = l_3$.*
2. *If l labels an instruction $(\mathtt{SUB}(r), l_2, l_3)$, then $s' = (a_1, \ldots, a_r - 1, \ldots, a_m)$ and $l' = l_2$ if $a_r > 0$, and $s' = s$ and $l' = l_3$ if $a_r = 0$.*

That is, the execution of an instruction $(\mathtt{ADD}(r), l_2, l_3)$ adds 1 to register r and then go to one of the instructions with labels l_2 and l_3, non-deterministically chosen. In the execution of an instruction $(\mathtt{SUB}(r), l_2, l_3)$, if register r is non-empty, then we subtract 1 from it and continue with the instruction labelled with l_2, otherwise we continue with the instruction labelled with l_3.

A *computation* of a register machine $M = (m, Lab, l_0, l_h, P)$ is a (finite or infinite) sequence $\sigma_0, \sigma_1, \ldots, \sigma_k$ of snapshots of M such that: (a) σ_0 is the initial snapshot, (b) σ_{i+1} is a successor snapshot of σ_i, for $i = 0, \ldots, k - 1$, and (c) if $k \in \mathbf{N}$ then σ_k is halting (in this case, we say that the computation is halting).

The set $N(M)$ of natural numbers computed by a register machine M is the set of numbers $x \in \mathbf{N}$ such that there exists a halting computation such that if $\sigma = (l_h, (a_1, \ldots, a_m))$ is its halting snapshot, then $a_1 = x$.

That is, a register machine M computes a set $N(M)$ of numbers in the following way: we start with all registers empty (hence storing the number zero) with the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number stored at that time in register 1 is said to be computed by M and hence it is introduced in $N(M)$ (because of the non-determinism in choosing the continuation of the computation in the case of ADD instructions, $N(M)$ can be an infinite set.)

It is known (see, e.g., [12]) that in this way we can compute all sets of numbers which are Turing computable (even with machines with a small number of registers, but this detail is not of interest here).

Theorem 1. *If Q is a Turing computable set, then there exists a register machine M, such that $N(M) = Q$.*

Without loss of generality, we may assume that when halting, all registers are empty, with the exception of register 1, which contains the generated/computed number.

3 Configurations

Informally speaking, a membrane structure is a 3D arrangement of vesicles, placed in a unique external membrane, called the skin membrane, without other relations taken into consideration than the inclusion (child-parent) relation. Mathematically, this corresponds to a rooted tree structure, with the root associated with the skin. As usual in membrane computing, we represent the membrane structures by strings of labelled parentheses (the labels are associated with the membranes, hence also with the nodes of the underlying tree) or, graphically, as Euler-Venn diagrams (of a particular type: no intersection is allowed and there is a unique external membrane, the skin). Figure 1 contains both the graphical and the tree representation of the following membrane structure

$$[_1[_2[_4[_7\,]_7[_8[_{10}\,]_{10}]_8[_9\,]_9]_4]_2[_3[_5\,]_5[_6\,]_6]_3]_1.$$

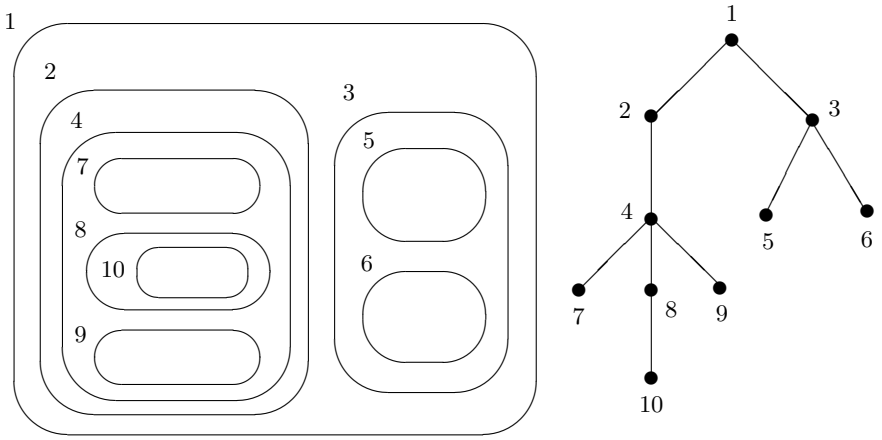

Fig. 1. Example of a membrane structure.

The number of membranes from a membrane structure μ is called the *degree* of μ, and the number of levels of the associated tree (with the root in level 1) is called the *depth* of μ. Sometimes, we denote the degree with $deg(\mu)$ and the depth with $dep(\mu)$. In the case of the membrane structure from Figure 1, we have the degree equal to 10 and the depth equal to 5.

In the example above, the labels are associated in a one-to-one way to membranes. In the case of membrane structures which evolve during a (pre)computation, e.g., by membrane division, we allow multiple membranes to be labelled with the same label. In such a case, the identification of membranes through labels will be no longer possible, that is why we use a double labelling for membranes, by means of pairs (i, j), where i is the "real" label, and j identifies

the copy of membrane i in the membrane structure. Thus, if we use labels from a set H, then the actual labels are from $H \cup (H \times \mathbf{N})$, with elements of H used alone when they label unique membranes and in couples when they label several membranes at the same time. This *extended* set of labels, $H \cup (H \times \mathbf{N})$, is denoted by H_e. The set of labels from H_e which actually appear in a membrane structure μ is denoted by $H_e(\mu)$; clearly, this set is finite, as we work here only with finite membrane structures.

A systematic, recursive procedure for constructing the extended labels can be based on an encoding $\langle i_1, j_2 \rangle$ of numbers j_1, j_2, for instance, using Cantor pair function: we start with labels h interpreted as $(h, 0)$; when dividing such a membrane, we label the resulting membranes by $(h, \langle 0, 1 \rangle)$ and $(h, \langle 0, 2 \rangle)$; the membranes obtained by dividing the first of these membranes are labelled by $(h, \langle \langle 0, 1 \rangle, 1 \rangle)$ and $(h, \langle \langle 0, 1 \rangle, 2 \rangle)$, while the membranes obtained by dividing the second membrane are labelled by $(h, \langle \langle 0, 2 \rangle, 1 \rangle)$ and $(h, \langle \langle 0, 2 \rangle, 2 \rangle)$, and so on and so forth. Because also h is a number, we can encode also h (it can be uniquely recovered from a code), but we prefer to keep it "visible", in pair-labels as above.

However, the following *convention* is made: we use natural numbers as labels of membranes, the skin membrane is always labelled with 1, and no other membrane has this label (hence the skin membrane does not need a double labelling).

The unique directly upper membrane of each membrane i, except the case of the skin ($i = 1$), which has no membrane above it, is called the parent of i in μ and denoted by $par_\mu(i)$; conversely, the directly inner membranes placed in membrane i are called the children of i and their set is denoted by $chd_\mu(i)$. Of course, the labels mentioned here are either from H or from H_e, depending on the circumstances. For an elementary membrane h we have $chd_\mu(h) = \emptyset$.

If in each compartment (we also say region) i of a membrane structure μ we place a multiset w_i, of objects from a given alphabet O, then we obtain a *configuration*. Note that a multiset can be empty (and then it is represented by λ).

Definition 3. *A configuration C over a set of objects O and with membranes labelled with elements of some set H is a pair $C = (\mu, M)$, where μ is a membrane structure with labels from H_e and $M : H_e(\mu) \longrightarrow O^*$ is a mapping which associates multisets over O with the regions of μ.*

4 Rules for Processing Configurations

Because a configuration means both a membrane structure and the associated multisets, we need rules for processing membranes and multisets of objects.

The types of rules we consider here are indicated in Table 1 (in all cases, a, b, c, d are objects from an alphabet O (in rules of type (10), c plays the role of a catalyst), h, h', h'' are labels from a set H, and u, v are multisets of objects over the alphabet O).

Table 1. Types of rules for configuration editing

Nr.	Identification	Form of the rule	Action
1	div	$[_h a]_h \to [_{h'} b]_{h'} [_{h''} c]_{h''}$	divide
2	mer	$[_h a]_h [_{h'} b]_{h'} \to [_{h''} c]_{h''}$	merge
3	endo	$[_h a]_h [_{h'} b]_{h'} \to [_h [_{h'} d]_{h'} c]_h$	endocytosis
4	exo	$[_h a [_{h'} b]_{h'}]_h \to [_h c]_h [_{h'} d]_{d'}$	exocytosis
5	cre	$a \to [_h b]_h$	create
6	dis	$[_h a]_h \to b$	dissolve
7	in	$a[_h\ ]_h \to [_h b]_h$	move in
8	out	$[_h a]_h \to [_h\ ]_h b$	move out
9	ncoo	$[_h a \to v]_h$	non-cooperative objects evolution
10	cat	$[_h ca \to cv]_h$	catalytic objects evolution
11	coo	$[_h u \to v]_h$	cooperative objects evolution

Note that the first four pairs of rules are one the inverse of the other: the operation of merging two membranes is the inverse of the operation of dividing a membrane, endocytosis is the inverse of exocytosis, creating a membrane is the inverse of dissolving a membrane, moving an object inside a membrane is the inverse of moving it outside a membrane. The cooperative rules for objects evolution can be considered as their own inverse.

When applying a rule $[_h a]_h \to [_{h'} b]_{h'} [_{h''} c]_{h''}$, the contents of membrane h, objects and membranes alike, are replicated and introduced in both membranes h' and h'', with the exception of object a, which is replaced by b in the first membrane and by c in the second membrane. Conversely, when merging the membranes h', h'', by using the rule $[_{h'} b]_{h'} [_{h''} c]_{h''} \to [_h a]_h$, then the contents of both membranes is accumulated (in the multiset sense) in membrane h, with the objects b, c replaced by a. Similarly, in the exocytosis/endocytosis rules, the whole contents of the moved membrane is moved together with the membrane, with the objects a, b replaced by c, d, respectively.

We stress the important fact that the labels appearing in these rules are from H, not from $H_e - H$; that is, a rule associated with a membrane with label h is applied to all copies of this membrane. Then, because O and H are finite sets, the number of rules of types (1) – (8) is finite; imposing a bound on the total multiplicity of multisets u and v (on the length of strings u and v) from rules of types (9), (10), (11), we can also ensure that the number of rules of these types is finite.

Rules for membrane division, dissolution and creation, for sending objects inside or outside a membrane are currently used in P systems with active membranes; non-cooperative and catalytic rules for objects evolution are also customary. Rules for merging membranes and for moving a whole membrane (together with its con-

tents) inside or outside another membrane were also occasionally used – e.g., in [3], [1], [11]. In many cases, variants of rules of these types are considered, for instance, in what concerns the labels of the involved membranes. In the case of division rules, the basic variant does not allow the change of the labels (which is allowed for division rules in Table 1 above); then, we can also allow the change of the label for in/out rules, and this extension was used, e.g., in [2].

These rules are used as standard in membrane computing, choosing the membranes and the objects in a non-deterministic manner, in such a way that the application of rules is maximally parallel. Each object and each membrane can be involved in the application of only one rule, with the mentioning that a rule of the types (9), (10), (11) is not considered as using the membrane h, but only the objects a, c and those from the multiset u, respectively. Thus, any membrane can be involved in only one rule of types (1) – (8), but the number of rules of types (9), (10), (11) which refer to a membrane h does not matter. In all rules of types (1) – (8) there are involved objects (the rules themselves indicate how the membrane structure is changed, under the influence of objects); these objects cannot be used at the same time also by rules of types (9), (10), (11). The use of rules of any type in a membrane structure is done in a bottom-up manner. For instance, if an elementary membrane h is divided (by a rule of type (1)), or moved into another membrane (by a rule of type (3)), then first all its objects different from the object involved in the division or in the move should evolve through object evolution rules, and, when divided or moved, the membrane will replicate or move the objects obtained by the maximally parallel use of rules for object evolution. Similarly, if a membrane contains inside other membranes, then first all lower level membranes evolve and then the upper level membranes evolve, in the same step.

It is important to note that we have not restricted here the use of rules of any type to elementary membranes, although this might be an interesting restriction to investigate.

The previous list contains those rules which we consider as basic, both biologically motivated and mathematically elegant, but, of course, other rules can be considered. This research topic, related to the set of rules to use, is left to the reader, and this is an attractive direction of investigation. Here we have mentioned these eleven types, but we will examine in some details only some combinations of rules, as an illustration of the type of problems which we want to raise in this paper.

We close this section by mentioning that by creating membranes and by endocytosis we can increase the depth of a membrane structure, while by means of the converse operations (by dissolving membranes and exocytosis) we can decrease the depth of a membrane structure. In turn, the membrane division increases the "width" of a membrane structure (hence it can increase the out-degree of the tree describing it). Rules of types (7) – (11) do not modify the membrane structure, they only move objects across membranes or handle objects inside compartments.

5 The Edit-Distance Among Configurations

We now introduce one of the basic notions of our approach, the edit-distance among two configurations with respect to a given set of rules.

Let us fix an alphabet of objects, O, and a set of labels, H. Let k be a natural number. The family of all configurations $C = (\mu, M)$, with the membrane structure of degree at most n and depth at most d, with labels in H_e and multisets of objects over O, such that each multiset w present in a region of μ has $|w| \leq k$, is denoted by $CFG_{n,d}(O, H, k)$.

Because both the degree of the membrane structure and the total multiplicity of elements in the multisets are bounded, the family $CFG_{n,d}(O, H, k)$ is finite for all $n, d, k \geq 1$.

Then, we define

$$CFG(O, H, k) = \bigcup_{n \geq 1, d \geq 1} CFG_{n,d}(O, H, k).$$

This is the family of all configurations with membranes labelled with elements of H_e and multisets (each of them of total multiplicity at most k) over O. Because we allow membranes with the same label from H, this family is infinite.

Let us now consider a set R of *rules*, for instance, of some of the types (1) – (11). This can be a set of specific rules, which are listed explicitly, or can be a set of *types* of rules (that is, the set of *all* rules of the given types). If we use all possible rules of given types, then we write the list of rule identifications after R; for instance, $R(div, in, coo_k)$ (resp., $R(div, in, cat_k)$) is the set of all rules over a given alphabet and a given set of labels, of types (div), (in), (coo) (resp., (div), (in), (cat)), with the left-hand and the right-hand member of cooperating rules of length at most k (resp., using at most k catalysts); the alphabet and the set of labels follow from the context where the rules are used. If the size of object processing rules is not relevant (or not known), then we omit mentioning it, and we write *coo* instead of coo_k. Sometimes we also write $ncoo_k$, indicating the fact that we use non-cooperating rules $a \to u$ with $|u| \leq k$.

If for each rule of type (1), (3), (5), (7) we also use the reverse rule of types (2), (4), (6), (8), respectively, and for each object evolution rule (of types (9), (10), (11)) $[_h u \to v]_h$ we also have the rule $[_h v \to u]_h$ in the set R, then the set R is said to be *reversible*. If to a set R of rules we add all rules reverse to the rules from R, then we obtain a reversible set of rules, denoted by cR and called the *reversible completion* of R. (Clearly, if R contains rules of a type (1), (3), (5), (7), then cR also contains rules of types (2), (4), (6), (8), respectively, and, if R contain rules of any type (ncoo), (cat), then cR may contain rules of type (coo). For instance, $cR(div, in, cat) = R(div, mer, in, out, coo)$, and $cR(cre, ncoo) = R(cre, dis, coo)$.)

For two configurations $C_1, C_2 \in CFG(O, H, k)$ we write $C_1 \Longrightarrow_R C_2$ if we can pass from C_1 to C_2 by using rules from R (this is a transition, in the customary meaning in membrane computing, with the rules used in the maximally parallel manner). We denote by $\Longrightarrow_R^*$ the reflexive and transitive closure of the relation $\Longrightarrow_R$.

Definition 4. (Edit-distance) *We define the mapping*

$$edit_R : CFG(O, H, k) \times CFG(O, H, k) \longrightarrow \mathbf{N} \cup \{\infty\},$$

as follows: for $C_1, C_2 \in CFG(O, H, k)$*, we take*

$$edit_R(C_1, C_2) = \begin{cases} \min\{r \mid C_1 \Longrightarrow_R C_{i_1} \Longrightarrow_R C_{i_2} \Longrightarrow_R \ldots \Longrightarrow_R C_{i_r} = C_2\}, \\ \qquad \textit{if } C_2 \textit{ can be reached from } C_1 \textit{ by using rules from } R, \\ \infty, \qquad \textit{otherwise.} \end{cases}$$

Of course, a minimal request about the mapping $edit_R$ would be its effective computability, but, unfortunately, this does not hold in general, it essentially depends on the type of rules from the set R. Specifically, for many classes of P systems which are known to compute all Turing computable sets of natural numbers (and for which the proof is based on simulating register machines), the reachability of a given configuration from the initial configuration of the system can be arranged (this depends on the system at hand) to imply the answer to the question whether or not a given number belongs to the set computed by the system; in turn, this is equivalent with the membership to the respective set of numbers, and this is not decidable for arbitrary Turing computable sets of numbers.

Three cases where this reasoning works are given in the next theorems.

Theorem 2. *Given* n, d, O, H, k *as above and the set of rules* $R(cat_2)$*, the mapping* $edit_{R(cat_2)} : CFG_{n,d}(O, H, k) \times CFG_{n,d}(O, H, k) \longrightarrow \mathbf{N} \cup \{\infty\}$ *is not effectively computable. Moreover, even for given configurations* $C_1, C_2 \in CFG_{n,d}(O, H, k)$*, we cannot effectively compute the value of* $edit_{R(cat_2)}(C_1, C_2)$*.*

Proof. The assertion follows from the proof of the main result from [7], where it is proved that catalytic P systems with two catalysts, using rules of the form $[_h ca \to cv]_h$ or $[_h a \to v]_h$, can generate all Turing computable sets of natural numbers. The proof is based on simulating a register machine and the simulation uses one membrane. One starts with the membrane containing only the initial label l_0 of the register machine – and this is the configuration C_1 from the statement of the theorem. In the end of a computation, in the membrane there remain only the two catalysts, as well as one given object a, whose multiplicity indicates the number generated by the computation – this is the configuration C_2 from the theorem statement. Thus, the membership of a given number to the generated set is equivalent with the reachability of the corresponding configuration from the initial configuration, and this is not decidable for arbitrary recursively enumerable sets of numbers. (See more precise constructions of this type in the proofs of the next theorems.) □

We present now two non-computability results with full details. The first one deals with the case when cooperative rules are used; such rules are more powerful than catalytic rules, but we can impose an additional restriction, on the total length of rules (which is smaller than in the case of catalytic rules, where we had five symbols in rules of the form $[_h ca \to cbd]_h$).

Theorem 3. *If R is a set of rules which can contain rules $[_h u \to v]_h$ of type (coo) with $|uv| \geq 3$, then the mapping $edit_R$ is not computable.*

Proof. It is suffices to prove that if $C_1, C_2 \in CFG_{n,d}(O, H, k)$ for given n, d, O, H, k, then the value of $edit_R(C_1, C_2)$ is not necessarily computable.

For an arbitrary non-deterministic register machine $M = (n, Lab, l_0, l_h, P)$, let us consider the alphabet

$$O = \{a_i \mid 1 \leq i \leq n\} \cup \{l, l', l'', l''' \mid l \in Lab\} \cup \{f\},$$

the configurations $C_1 = ([_1\,]_1, l_0), C_2 = ([_1\,]_1, \lambda)$, and the following set of rules:

$$\begin{aligned} R = {} & \{[_1 l_1 \to l_2 a_r]_1,\ [_1 l_1 \to l_3 a_r]_1 \mid \text{for all } l_1 : (\texttt{ADD}(r), l_2, l_3) \in P\} \\ & \cup \{[_1 l_1 a_r \to l_2]_1, \\ & \quad [_1 l_1 \to l_1' l_1'']_1,\ [_1 l_1' a_r \to f]_1,\ [_1 l_1'' \to l_1''']_1,\ [_1 l_1' l_1''' \to l_3]_1 \mid \\ & \quad \text{for all } l_1 : (\texttt{SUB}(r), l_2, l_3) \in P\} \\ & \cup \{[_1 f \to f]_1,\ [_1 l_h \to \lambda]_1\}. \end{aligned}$$

Starting from C_1, these rules simulate the register machine M. The simulation of an `ADD` instruction is obvious, while the simulation of a `SUB` instruction $l_1 : (\texttt{SUB}(r), l_2, l_3)$ is done as follows. When l_1 is present, we either use the rule $[_1 l_1 a_r \to l_2]_1$, which corresponds to the case when the register r is non-empty, or we use the rule $[_1 l_1 \to l_1' l_1'']_1$. In the latter case, if any copy of a_r is present, then in the next step we have to use the rule $[_1 l_1' a_r \to f]_1$, simultaneously with the rule $[_1 l_1'' \to l_1''']_1$; if no a_r is present, then the former rule cannot be used, hence in the third step we can introduce the label l_3 by means of the rule $[_1 l_1' l_l''' \to l_3]_1$, which completes the simulation. If the symbol f was introduced, hence the choice of the rule $[_1 l_1 \to l_1' l_1'']_1$ was "wrong", because the subtraction was possible, then the rule $[_1 f \to f]_1$ will be used forever, hence we never reach the configuration C_2. Thus, $C_1 \Longrightarrow_R^* C_2$ if and only if $0 \in N(M)$. Because M is arbitrary, the membership of 0 to $N(M)$ is not decidable, hence the reachability of C_2 is not decidable; this implies that we cannot compute $edit_R(C_1, C_2)$. □

The next result deals with non-cooperative rules, but instead it uses two types of membrane manipulating rules.

Theorem 4. *The mapping $edit_{R(div,exo,ncoo_2)}$ is not computable.*

Proof. It is suffices to prove that if $C_1, C_2 \in CFG_{n,d}(O, H, k)$ for given n, d, O, H, k, then the value of $edit_{R(div,exo,ncoo_2)}(C_1, C_2)$ is not necessarily computable.

We start again from an arbitrary non-deterministic register machine $M = (n, Lab, l_0, l_h, P)$, and consider the alphabet

$$O = \{a_i \mid 1 \leq i \leq n\} \cup \{l, l', l'' \mid l \in Lab\} \cup \{d\},$$

the configurations $C_1 = ([_1[_2\ ]_2]_1, d, l_0), C_2 = ([_1[_2\ ]_2]_1, d, \lambda)$, and the following set of rules:

$$
\begin{aligned}
R = {} & \{[_2 l_1 \to l_2 a_r]_2,\ [_2 l_1 \to l_3 a_r]_2 \mid \text{for all } l_1 : (\texttt{ADD}(r), l_2, l_3) \in P\} \\
& \cup \{[_2 l_1]_2 \to [_{(r)} l_2]_{(r)} [_3 d]_3, \\
& \quad [_{(r)} a_r]_{(r)} \to [_2 d]_2 [_3 d]_3, \\
& \quad [_{(r)} l_2]_{(r)} \to [_4 d]_4 [_4 d]_4, \\
& \quad [_2 l_1]_2 \to [_{(r')} l_3']_{(r')} [_3 d]_3, \\
& \quad [_{(r')} l_3' \to l_3'']_{(r')}, \\
& \quad [_{(r')} l_r'']_{(r')} \to [_2 l_3]_2 [_3 d]_3, \\
& \quad [_{(r')} a_r]_{(r')} \to [_4 d]_4 [_4 d]_4 \mid \\
& \quad \text{for all } l_1 : (\texttt{SUB}(r), l_2, l_3) \in P\} \\
& \cup \{[_1 d[_3 d]_3]_1 \to [_1 d]_1 [_3 d]_3, \\
& \quad [_4 d \to d]_4, \\
& \quad [_2 d \to \lambda]_2, \\
& \quad [_2 l_h \to \lambda]_2\}.
\end{aligned}
$$

Starting from configuration C_1, these rules simulate the computations of the register machine in the following way.

The ADD instructions are simply simulated by evolution rules in the inner membrane.

If the label l_1 of a SUB instruction $l_1 : (\texttt{SUB}(r), l_2, l_3)$ appears in membrane 2, then we non-deterministically use one of the rules $[_2 l_1]_2 \to [_{(r)} l_2]_{(r)} [_3 d]_3$ and $[_2 l_1]_2 \to [_{(r')} l_3']_{(r')} [_3 d]_3$; the first one tries to follows the path towards label l_2 (when at least one copy of a_r is present), the latter one corresponds to the case when no a_r is present.

Assume that we have used the first rule. In the next step we can use the rule $[_{(r)} l_2]_{(r)} \to [_4 d]_4 [_4 d]_4$, and this leads to an endless computation which never reaches the configuration C_2; the only way to avoid this is to use instead the rule $[_{(r)} a_r]_{(r)} \to [_2 d]_2 [_3 d]_3$, provided that there is a copy of a_r in the membrane. In this way, membrane (r) is used, and divided into a membrane with label 2 and one with label 3. All membranes with label 3 (and object d inside) are thrown out of the configuration, by the exocytosis rule $[_1 d[_3 d]_3]_1 \to [_1 d]_1 [_3 d]_3$. Therefore, we return to a configuration with two membranes, l_2 inside the inner membrane, and the number of objects $a_i, 1 \leq i \leq n$, corresponding to the contents of the registers (note that the object d produced by the division operation instead of the subtracted a_r was erased from membrane 2). That is, the continuation has a chance to go to configuration C_2 only if the simulation of this case of the SUB instruction was correct (otherwise, further objects and membranes remain forever in the configurations we reach).

If we apply first the rule $[_2 l_1]_2 \to [_{(r')} l'_3]_{(r')} [_3 d]_3$, then in the next step we have to use the object evolution rule $[_{(r')} l'_3 \to l''_3]_{(r')}$. Simultaneously (because the rule above does not "keep busy" the membrane), if any occurrence of a_r is present, then we have to also use the rule $[_{(r')} a_r]_{(r')} \to [_4 d]_4 [_4 d]_4$, which introduces the trap membrane 4, again preventing the reach of configuration C_2. If no occurrence of a_r is present, then the membrane remains unchanged, hence in the next step we can use the rule $[_{(r')} l''_r]_{(r')} \to [_2 l_3]_2 [_3 d]_3$, which both returns the label of the membrane to 2 and introduces the object l_3. This means that also this sequence of operations correctly simulates the SUB instruction, in the sense that we remain "on the way towards C_2" if and only if l_3 is introduced and no a_r is present.

We continue in this way. The only way to reach C_2 is to reach the label l_h, which can be erased. This means that $0 \in N(M)$, which is undecidable for an arbitrary register machine M, and this completes the proof. □

In the previous theorems, the configurations C_1, C_2 are taken from the finite set $CFG_{n,d}(O, H, k)$, but the reachability is not restricted to paths through configurations which belong to the same $CFG_{n,d}(O, H, k)$, but arbitrarily large configurations, as the multisets contained, are allowed; if we restrict to paths through $CFG_{n,d}(O, H, k)$, then, of course, the reachability is decidable, because we only have to search a finite set of configurations.

Remark 1. The previous proofs also give the universality of P systems using rules of types (coo_2), and (div), (exo), (ncoo_2), respectively; the first result is known – see Theorem 3.3.3 from [15] – except that the proof from [15] use rules $u \to v$ with $|u| \leq 3$ and $|v| \leq 4$; the latter result reminds a similar result from [2] (our division rules do not use polarizations, as usual in P systems with active membranes, but they allow label change, a feature also used in [2]), with the mentioning that the result from [2] is obtained in a more complicated manner (the proof is based on simulating matrix grammars with appearance checking), instead of exocytosis uses (out) rules, and the used rules for object evolution are of the form $a \to v$ with $|v| \leq 3$. A universality result for P systems using (div), (exo), (ncoo) rules appears also in [11], but using endocytosis rules (the proof is based on a different idea: simulating another class of P systems, which is known to be universal).

Research topic: Find sets R of rules for which the mapping $edit_R$ is computable on $CFG(O, H, k) \times CFG(O, H, k)$.

Still, from a mathematical point of view, the mapping $edit_R$ can be considered as a good estimation of the distance from C_1 to C_2:

Lemma 1. *For any R, $edit_R$ is a weak-metric.*

Proof. The fact that $edit_R(C_1, C_2) = 0$ if and only if $C_1 = C_2$ is obvious. The triangle inequality is also obvious: if we can pass from C_1 to C_3 and from C_3 to C_2, this also gives a path from C_1 to C_2, which should be at least as long as the shortest path from C_1 to C_2; that is, $edit_R(C_1, C_2) \leq edit_R(C_1, C_3) + edit_R(C_3, C_2)$. □

The symmetry condition does not hold in general, but it is entailed by the reversibility of R:

Lemma 2. *If R is reversible, then $edit_R(C_1, C_2) = edit_R(C_2, C_1)$ for all $C_1, C_2 \in CFG(O, H, k)$.*

Proof. Each transition performed by a reversible rule can be performed also in the reverse direction, using the reverse operations made possible by the rules, hence any path from C_1 to C_2 corresponds to a path from C_2 to C_1, with the same length. Thus, not only C_2 is reachable from C_1 if and only if C_1 is reachable from C_2, but also the minimal number of transitions necessary in any of the two directions is the same. □

This lemma makes possible the definition of a plain metric among configurations:

Lemma 3. *For a reversible set R of rules, $edit_R$ is a metric.*

Proof. Combine Lemmas 1 (weak metric) and 2 (the symmetry condition). □

6 A Case Study: Rules of Types (cre) and (ncoo)

Somewhat expected, by using membrane creation rules (as well as non-cooperative rules for handling the objects), we can create any configuration from a configuration of a very reduced form, a sort of "seed-of-configurations". This result is of a particular interest, having several consequences.

Let us consider the simplest non-trivial configuration, $C_0 = ([_1\ ]_1, a)$ (one membrane, one single object inside).

Theorem 5. *For every configuration $C \in CFG_{n,d}(O, H, k)$, there is a finite set R of rules of types (cre) and (ncoo), over an alphabet $O' \supset O$, such that $edit_R(C_0, C) \leq 2d - 1$.*

Proof. Let us assume that $C = (\mu, M)$, with $deg(\mu) = n$ and $dep(\mu) = d$. For each label $g \in H_e(\mu)$ of a membrane in μ we consider the objects $\langle g \rangle$ and $\langle g \rangle'$. That is,

$$O' = O \cup O_H, \text{ where } O_H = \{\langle g \rangle,\ \langle g \rangle' \mid g \in H_e(\mu)\}.$$

Now, we consider the following rules, over O':

1. $[_1 a \to M(1)\langle j_1 \rangle \langle j_2 \rangle \dots \langle j_{r_1} \rangle]_1$, where $\{j_1, j_2, \dots, j_{r_1}\} = chd_\mu(1)$.
2. $\langle g \rangle \to [_g \langle g \rangle']_g$,
   $[_g \langle g \rangle' \to M(g)\langle h_1 \rangle \langle h_2 \rangle \dots \langle h_{r_g} \rangle]_g$, where $\{h_1, h_2, \dots, h_{r_g}\} = chd_\mu(g)$ and $g \in H_e(\mu)$.

It is obvious that by using these rules we can construct C_2 level by level, with two steps necessary for each level which is not the first one (of the root). Thus, in at most $2d-1$ steps (if $d=1$ and $C \neq C_0$, then we pass from C_0 to C in one step) we reach the configuration C. That is, $edit_R(C_0, C) \leq 2d-1$, where R is the set of rules considered above. □

If we add to R also the reverse rules, then we can pass in the same number of steps also from C to C_0. Thus, $edit_{cR}(C_0, C) \leq 2d-1$.

Note the interesting fact that the distance from C_0 to C does not depend on the degree of C, but only on its depth.

Of course, in the theorem above we can take R as containing *all* rules of types (cre) and (ncoo) over O' and H, with bounded object evolution rules), that is, $R(cre, ncoo_{k+r})$, where $r = \max\{card(chd_\mu(g) \mid g \in H_e(\mu)\}$. This dependence of the size of object processing rules (those of type (ncoo)) from R on k and on the out-degree of μ is not very attractive. At the expense of a linear slowdown, this dependence can be avoided, and we can consider rules $[_g a \to v]_g$ with $|v| \leq 2$ – hence the set $R(cre, ncoo_2)$.

Indeed, each rule $[_g \alpha \to \alpha_1\alpha_2 \dots \alpha_k]_g$ can be replaced with the sequence of rules

$$\begin{aligned}
&[_g \alpha \to \alpha_1\langle\alpha_2 \dots \alpha_k\rangle]_g,\\
&[_g \langle\alpha_i \dots \alpha_k\rangle \to \alpha_i\langle\alpha_{i+1} \dots \alpha_k\rangle]_g,\ 2 \leq i \leq k-2,\\
&[_g \langle\alpha_{k-1}\alpha_k\rangle \to \alpha_{k-1}\alpha_k]_g,
\end{aligned}$$

where $\langle\alpha_i \dots \alpha_k\rangle,\ 2 \leq i \leq k$, are new symbols, constructed for all $\alpha, \alpha_i \in O'$.

In this way, the size of rules from R becomes independent of C, but the construction of C does no longer proceed level by level and, moreover, it can last $d \cdot (k+r) - 1$ steps, where r is the maximal out-degree of μ: instead of one rule $[_g \langle g\rangle' \to M(g)\langle h_1\rangle \dots \langle h_r\rangle]_g$ we use now $k+r-1$ rules, to which we have to add the use of the rule $\langle g\rangle \to [_g \langle g\rangle']_g$. This result is rather interesting, hence it is worth stating as a theorem.

Theorem 6. *If $C \in CFG_{n,d}(O, H, k)$, then $edit_{R(cre,ncoo_2)}(C_0, C) < d\cdot(n+k)-1$.*

Proof. Besides the previous discussion, we have to use the fact that the maximal out-degree of μ is strictly smaller than the degree of μ, which is n. □

If we want to use a construction as in the proof of Theorem 5 for obtaining the initial configuration of a P system, hence we also have sets of rules associated with the compartments of μ, acting on the membranes and on the multisets of C in a computation which is different from the pre-computation done by using the rules of R, then there appears a problem here: in general, when dealing with classes of membrane systems where we can create new membranes, the rules associated with membranes are supposed to be given as soon as the membrane is created (and they are identified by the label of the membrane). That is, in the construction above,

after introducing a multiset $M(g)$ in a membrane g, the rules associated with g can start working, although the membranes from lower levels are not yet present. This leads to the de-synchronization of the system, which is not correct.

Still, this difficulty can be overcome, by introducing the multisets only in the last moment of the construction. Because this is a rather important aspect, we give here the full details, modifying the construction from the proof of Theorem 5 for this case.

Theorem 7. *For every configuration $C = (\mu, M) \in CFG_{n,d}(O, H, k)$, there is a finite set R of rules of types (cre) and (ncoo), over an alphabet $O' \supset O$, such that $edit_R(C_0, C) \leq 2d - 1$ and the multisets of C are available only in the last step of the passage from C_0 to C. That is, there exist configurations $C_1, \ldots, C_p \in CFG_{n,d}(O, H, k)$ such that $p \leq 2d - 1$, $C_0 \Longrightarrow_R C_1 \Longrightarrow_R C_2 \Longrightarrow_R \ldots \Longrightarrow_R C_p = C$, and for all $i \in H_e(\mu)$, $M(i) \in C - (C_0 \cup C_1 \cup \ldots \cup C_{p-1})$.*

Proof. This time, for each label $g \in H_e(\mu)$ of a membrane in μ we consider the objects $\langle g \rangle, \langle g \rangle'$ indexed with the step when such symbols are supposed to evolve. We proceed in the same way with the multisets $M(i)$. Specifically, we take

$$O' = O \cup \{\langle g \rangle_s,\ \langle g \rangle'_s \mid g \in H_e(\mu), 2 \leq s \leq 2d-1\}$$
$$\cup \{\langle M(g) \rangle_s \mid g \in H_e(\mu), 2 \leq s \leq 2d-1\},$$

and the following rules in R:

1. $[_1 a \to \langle M(1) \rangle_2 \langle j_1 \rangle_2 \langle j_2 \rangle_2 \ldots \langle j_{r_1} \rangle_2]_1$, where $\{j_1, j_2, \ldots, j_{r_1}\} = chd_\mu(1)$.
2. $\langle g \rangle_{2s} \to [_g \langle g \rangle'_{2s+1}]_g$,
   $[_g \langle g \rangle'_{2s+1} \to \langle M(g) \rangle_{2s+2} \langle h_1 \rangle_{2s+2} \langle h_2 \rangle_{2s+2} \ldots \langle h_{r_g} \rangle_{2s+2}]_g$, where
   $\{h_1, h_2, \ldots, h_{r_g}\} = chd_\mu(g)$ and $g \in H_e(\mu)$ labels a membrane in a level s of μ with $s < d$.
3. $\langle g \rangle_{2d-2} \to [_g \langle g \rangle'_{2d-1}]_g$,
   $[_g \langle g \rangle'_{2d-1} \to M(g)]_g$, where g is a membrane in level d of μ.
4. $[_g \langle M(g) \rangle_j \to \langle M(g) \rangle_{j+1}]_g$, for all $g \in H_e(\mu)$ and $1 \leq j \leq 2d-2$,
   $[_g \langle M(g) \rangle_{2d-1} \to M(g)]_g$, for all $g \in H_e(\mu)$.

This time, each multiset $M(g)$ is introduced in the "protected" form $\langle M(g) \rangle_s$, with s indicating the step of the computation (pre-computation, if we take into account the fact that we want to construct the initial configuration of a P system); only in the last step the symbols $\langle M(g) \rangle_{2d-1}$ are replaced with the actual multisets $M(g)$, all of them at the same time, hence from now on the computation can start, synchronously, in all compartments of μ. □

The remark made after Theorem 5, about the possibility of bounding the multiset v from rules $[_g a \to v]_g$, is valid also for the construction from the proof of Theorem 7, but this time we have to be careful with the synchronization, hence every symbol from a multiset should be "protected", such that it becomes active only in the last step of the pre-computation. This can be again achieved by indexing

each object from O with counters which increase with the steps of the computation. Because the length of the pre-computation depends on the out-degree of the tree of μ, we have to count in the indices of symbols $\langle M(i)\rangle_s$ for each level exactly $r+1$ steps, where r is the maximal out-degree of μ, even if the out-degree of the node where we work might be smaller than r. Thus, after $(r+1)d-1$ steps, we pass to producing the symbols of multisets $M(i)$, one by one, during $k-1$ steps, where k is the maximal length of a multiset. That is, we un-protect the symbols, passing from objects of type $\langle a\rangle_s$ to a only after $(r+1)d-1+(k-1)$ steps. The technical details, although clear from the previous discussion, are cumbersome, hence we omit them.

We illustrate the construction from the proof of Theorem 7 (without restrictions on the size of rules of type (ncoo)) with the case of the configuration C with the membrane structure as in Figure 1 and multisets $M(i), 1 \leq i \leq 10$, which we do not specify. The set R contains the rules given in Table 2 (we present the rules in the order they are used in the 9 steps of the passage from C_0 to C). The reader is asked to follow the work of these rules, on the way from C_0 to C.

Table 2. Rules of types (cre), (ncoo) for obtaining the configuration from Figure 1

Step	Rules
1	$[_1 a \to \langle M(1)\rangle_2\langle 2\rangle_2\langle 3\rangle_2]_1$
2	$\langle 2\rangle_2 \to [_2\langle 2\rangle'_3]_2$, $\langle 3\rangle_2 \to [_3\langle 3\rangle'_3]_3$, $[_1\langle M(1)\rangle_2 \to \langle M(1)\rangle_3]_1$
3	$[_2\langle 2\rangle'_3 \to \langle M(2)\rangle_4\langle 4\rangle_4]_2$, $[_3\langle 3\rangle'_3 \to \langle M(3)\rangle_4\langle 5\rangle_4\langle 6\rangle_4]_3$, $[_1\langle M(1)\rangle_3 \to \langle M(1)\rangle_4]_1$
4	$\langle 4\rangle_4 \to [_4\langle 4\rangle'_5]_4$, $\langle 5\rangle_4 \to [_5\langle 5\rangle'_5]_5$, $\langle 6\rangle_4 \to [_6\langle 6\rangle'_5]_6$, $[_1\langle M(1)\rangle_4 \to \langle M(1)\rangle_5]_1$, $[_2\langle M(2)\rangle_4 \to \langle M(2)\rangle_5]_2$, $[_3\langle M(3)\rangle_4 \to \langle M(3)\rangle_5]_3$
5	$[_4\langle 4\rangle'_5 \to \langle M(4)\rangle_6\langle 7\rangle_6\langle 8\rangle_6\langle 9\rangle_6]_4$, $[_5\langle 5\rangle'_5 \to \langle M(5)\rangle_6]_5$, $[_6\langle 6\rangle'_5 \to \langle M(6)\rangle_6]_6$, $[_1\langle M(1)\rangle_5 \to \langle M(1)\rangle_6]_1$, $[_2\langle M(2)\rangle_5 \to \langle M(2)\rangle_6]_2$, $[_3\langle M(3)\rangle_5 \to \langle M(3)\rangle_6]_3$
6	$\langle 7\rangle_6 \to [_7\langle 7\rangle'_7]_7$, $\langle 8\rangle_6 \to [_8\langle 8\rangle'_7]_8$, $\langle 9\rangle_6 \to [_9\langle 9\rangle'_7]_9$, $[_i\langle M(i)\rangle_6 \to \langle M(i)\rangle_7]_i$, for $i = 1,2,3,4,5,6$
7	$[_7\langle 7\rangle'_7 \to \langle M(7)\rangle_8]_7$, $[_8\langle 8\rangle'_7 \to \langle M(8)\rangle_8\langle 10\rangle_8]_8$, $[_9\langle 9\rangle'_7 \to \langle M(9)\rangle_8]_9$, $[_i\langle M(i)\rangle_7 \to \langle M(i)\rangle_8]_i$, for $i = 1,2,3,4,5,6$
8	$\langle 10\rangle_8 \to [_{10}\langle 10\rangle'_9]_{10}$, $[_i\langle M(i)\rangle_8 \to \langle M(i)\rangle_9]_i$, for $i = 1,2,3,4,5,6,7,8,9$
9	$[_{10}\langle 10\rangle'_9 \to M(10)]_{10}$, $[_i\langle M(i)\rangle_9 \to M(i)]_i$, for $i = 1,2,3,4,5,6,7,8,9$

Theorem 6 has a series of interesting consequences.

For an alphabet O and a set H of labels, let O_H be the alphabet defined in the proof of Theorem 5. Let $R(cre, ncoo_2)$ be the set of all rules of types (cre) and (ncoo) with objects in $O \cup O_H$ and labels in H. Clearly, this is a finite set.

Corollary 1. *If $C_1 \in CFG_{n_1,d_1}(O,H,k_1)$ and $C_2 \in CFG_{n_2,d_2}(O,H,k_2)$, then $edit_{cR(cre,ncoo_2)}(C_1,C_2) < d_1 \cdot (n_1+k_1) + d_2 \cdot (n_2+k_2) - 2$.*

Proof. We just write the triangle inequality for C_1, C_2, C_0 and we use the result from Theorem 6. Because we work with the reversible completion of the set $R(cre, ncoo_2)$, we can pass both from C_0 to C_1, and C_2, and conversely, thus having a path from C_1 to C_2. □

This means that any two configurations from $CFG(O, H, k)$ can be reached from each other, they are at a finite distance with respect to the set $cR(cre, ncoo_2)$ of rules.

The previous corollary gives only an upper bound on the distance among the considered configurations. Can this estimation be improved? This seems to be always possible, at least with a (small) constant: when passing directly from C_1 to C_2 we can "save" that part of the membrane structure of C_1 which can be found also in C_2 (for instance, the skin membrane). The question which is the largest improvement in general, or for two configurations with certain "similarities" remains as a research topic.

7 Generators for Families of Configurations

Another interesting consequence of Theorem 6 is the fact that the configuration C_0 can be seen as a generator of the whole family $CFG(O, H, k)$, with respect to rules from $R(cre, ncoo_2)$, or, in terms of P systems, as a normal form for all systems which can use rules for membrane creation (the "minimality" of the configuration C_0 is somewhat surprising and, of course, pleasant: any recursively enumerable set of numbers can be generated by a system with the initial configuration equal to C_0). We place this idea in a more general framework, resembling the one of AFL (abstract family of languages) theory, see [8].

Definition 5. *Let $\mathcal{C} \subseteq CFG(O, H, k)$ be a family of configurations and R a set of rules. We define*

$$gen_R(\mathcal{C}) = \{C \in CFG(O, H, k) \mid edit_R(C', C) < \infty \text{ for some } C' \in \mathcal{C}\}.$$

That is, $gen_R(\mathcal{C})$ is the family of configurations which can be reached, in a finite number of steps, when starting from configurations from $\mathcal{C}$, by using rules from R.

Definition 6. *For $\mathcal{C}_1, \mathcal{C}_2 \subseteq CFG(O, H, k)$ and a set R of rules, we say that:*

1. *$\mathcal{C}_1$ is an R-generator for $\mathcal{C}_2$ if $gen_R(\mathcal{C}_1) = \mathcal{C}_2$;*
2. *$\mathcal{C}_1$ is an R-cover for $\mathcal{C}_2$ if $\mathcal{C}_2 \subseteq gen_R(\mathcal{C}_1)$.*

Of course, any set of configurations is an R-cover for itself with respect to any set of rules, hence of interest are *minimal* generators and covers, in particular, singleton generators and covers.

Definition 7. *A family $\mathcal{C} \subseteq CFG(O, H, k)$ of configurations is said to be principal with respect to a set R of rules if there is a configuration $C \in CFG(O, H, k)$ such that $gen_R(\{C\}) = \mathcal{C}$. In such a case we say that C is an R-generator for $\mathcal{C}$.*

In this framework, Theorem 6 directly implies the following interesting result:

Theorem 8. *The family $CFG(O, H, k)$ is principal with respect to the set of rules $R(cre, ncoo_2)$, and C_0 is an $R(cre, ncoo_2)$-generator for $CFG(O, H, k)$.*

Because in a principal family each configuration can be reached from any given generator, if the respective set of rules is reversible, then it also follows that any configuration can be reached from any other configuration (passing through the given generator), hence we have:

Corollary 2. *In the family $CFG(O, H, k)$, any configuration is a generator with respect to $cR(cre, ncoo_2) = R(cre, dis, coo_2)$.*

8 Using Rules of Types (div), (endo), and (ncoo)

The rules of type (cre) and (ncoo) are not the only ones for which we can obtain results as above. The use of rules of type (ncoo) (or (cat), (coo)) cannot be avoided, because these rules are the only ones (in the basic set considered here) which can increase the number of objects in the configurations, but membranes can be created also by division. However, as we have noticed already, by division we cannot increase the depth of the membrane structure. This can be done by endocytosis rules, hence a combination of rules of type (div) and (endo) can simulate rules of type (cre).

For instance, if we want to create a new membrane, with label g, inside an existing membrane h, we can first pass from $[_h a]_h$ to $[_h b]_h [_g c]_g$ (the object a is supposed to exist in membrane h), by using a division rule, then to $[_h [_g c']_g b']_h$, by an endocytosis rule.

Still, one more difficulty arises: we cannot divide the skin membrane, hence we cannot start from the configuration C_0 used in the previous sections, we need at least two membranes in the beginning.

On the other hand, by using endocytosis/exocytosis rules (as well as rules of type (coo)), any configuration with n membranes can be transformed into any configuration with the same number of membranes. Then, if exocytosis is allowed also with respect to the skin membrane, then we can expel membranes from the configurations, thus decreasing the degree of the membrane structures. Of course, by using membrane dissolving rules (but not membrane creation rules) we can again decrease the degree of configurations.

Therefore, several combinations of rules can be considered which leads to results like those from Sections 6 and 7 (of course, with edit-distances among configurations different from those in sections above).

In what follows we consider only the case when we use rules of types (div), (endo), and (ncoo), again with a bound on the length of rules of type (ncoo), without permitting the exocytosis for the skin membrane (consequently, the number of membranes cannot be decreased). That is, we use

the set $R(div, endo, ncoo_2)$ of rules. Consider also the configurations $C_0^m = ([_1[_2 \ldots [_m]_m \ldots]_2]_1, a_1, a_2, \ldots, a_m)$, for all $m \geq 1$, where $a_1, a_2, \ldots, a_m$ are new symbols, not in the alphabet O, with each object a_i present in region i, $1 \leq i \leq m$.

We mention here, without a proof, only the result corresponding to Theorem 8. For $m \geq 1$, let us denote

$$CFG_{\geq m}(O, H, k) = \bigcup_{n \geq d \geq m} CFG_{n,d}(O, H, k),$$

hence this is the family of all configurations $C = (\mu, M)$ with membrane structures of degree at least m.

Theorem 9. *The family $CFG_{\geq m}(O, H, k)$ is principal with respect to the set of rules $R(div, endo, ncoo_2)$, and C_0^m is an $R(div, endo, ncoo_2)$-generator for this family.*

It is also worth noting the result corresponding to Theorem 7, which provides a further normal form for systems which are allowed to use rules for the division of membranes and endocytosis, and non-cooperative rules for object evolution: any system of this type of degree at least m is equivalent with a system having the initial configuration equal to C_0^m.

If we consider a set of rules containing rules of types (div), (endo), (exo), (ncoo), without restricting the exocytosis, or if we allow dissolving rules, then we can again generate $CFG(O, H, k)$ starting from C_0, hence we have a general normal form theorem, for P systems of any degree.

In order to have an idea about the way the path from C_0^m to an arbitrary configuration $C \in CFG_{\geq m}(O, H, k)$ proceeds, we illustrate the construction for a result like that from Theorem 5 – hence without the complications which appear if we want to protect the multisets of objects, in view of a proof of a result like Theorem 7, and without limiting the size of object processing rules.

We consider again a configuration C with the membrane structure μ from Figure 1 and multisets $M(i)$ placed in the compartments $i = 1, 2, \ldots, 10$ of μ. The configuration C can be generated from C_0^2, even if we use no exocytosis rule, and we consider here this case.

The set of rules used is given in Table 3, which, like in case of Table 2, presents the rules in the order of the steps when they are used. The symbols b, c, d, e, f used in the rules are chosen in an *ad hoc* manner; in order to generalize the construction to an arbitrary configuration, these symbols should be related in a systematic manner to the membranes whose evolution they control and to the step of the construction (which is also related to the level of the tree where we work). Such technicalities are left to the reader.

9 A Plethora of Research Topics

Several research topics were already mentioned above, and many others can be considered. We formulate here only some of them, without any intention to be

Table 3. Rules of types (div), (endo), (ncoo) for obtaining the configuration from Figure 1

Step	Rules
1	$[_1 a_1 \to M(1)]_1,\ [_2 a_2]_2 \to [_2 b_1]_2 [_3 b_2]_3$
2	$[_2 b_1]_2 \to [_2 c_1]_2 [_4 c_2]_4,\ [_3 b_2]_3 \to [_3 c_1]_3 [_5 c_2]_5$
3	$[_2 c_1]_2 [_4 c_2]_4 \to [_2 [_4 c_2']_4 c_1']_2,\ [_3 c_1]_3 [_5 c_2]_5 \to [_3 [_5 c_2']_5 c_1']_3$
4	$[_2 c_1' \to M(2)]_2,\ [_3 c_1' \to M(3)]_3,$ $[_4 c_2']_4 \to [_4 d_1]_4 [_7 d_2]_7,\ [_5 c_2']_5 \to [_5 d_1]_5 [_6 d_2]_6$
5	$[_4 d_1]_4 [_7 d_2]_7 \to [_4 [_7 d_2']_7 d_1']_4,\ [_5 d_1 \to M(5)]_5,\ [_6 d_2 \to M(6)]_6$
6	$[_4 d_1' \to M(4)]_4,\ [_7 d_2']_7 \to [_7 e_1]_7 [_8 e_2]_8$
7	$[_7 e_1 \to M(7)]_7,\ [_8 e_2]_8 \to [_8 e_2']_8 [_9 e_3]_9$
8	$[_9 e_3 \to M(9)]_9,\ [_8 e_2']_8 \to [_8 f_1]_8 [_{10} f_2]_{10}$
9	$[_8 f_1]_8 [_{10} f_2]_{10} \to [_8 [_{10} f_2']_{10} f_1']_8$
10	$[_8 f_1' \to M(8)]_8,\ [_{10} f_2' \to M(10)]_{10}$

exhaustive (and without knowing which of these topics are difficult and which are straightforward).

In the previous sections we have investigated only some very particular cases: the family $CFG(O, H, k)$, the set of rules $R(cre, ncoo_2)$, the generator C_0, as well as the family $CFG_{\geq 2}(O, H, k)$, the set of rules $R(div, endo, ncoo_2)$, and the generator C_0^m. Consider other families of configurations. For instance, what about $CFG_{n,d}(O, H, k)$? What about families using particular types of membrane structures (described by linear trees, binary trees, trees of depth 2, etc)? Then, there are families of P systems – hence of configurations – which appear in a natural way in various contexts; such a context is that of solving **NP**-complete problems in polynomial time, using a family of P systems constructed in an uniform manner, starting from the problem to solve (see details in [16] and in a series of other papers). Can such a family be generated from a given unique configuration, using a suitable set of rules?

Many problems are suggested by the parallelism with the AFL theory. Define a sort of "abstract family of configurations", with "nice" closure properties (under operations defined by evolution rules, but also under set-theoretical operations, such as union and intersection). Consider sets of rules R and families of configurations $\mathcal{C}$ such that $gen_R(\mathcal{C}) \neq CFG(O, H, k)$; which properties has such a family?

A large panoply of problems is related to the set of rules taken into account. Any set of rules of types belonging to a subset of the set $\{(1), \ldots, (11)\}$ can be considered, arbitrary or reversible, with specific restrictions on the size of multisets used in rules for object evolution. Also, other types of rules can be considered.

Find families of configurations and sets of rules for which the minimal generators are not singletons (that is, find families of configurations which are not principal with respect to a given set of rules). Is the property of being a generator decidable for a given family of configurations, a given set of rules, and a given candidate generator? (The answer should be negative in general, because of the Turing completeness of most classes of P systems.) Providing that we know that a family is principal, how can we find a generator? Is this possible in an effective, algorithmic way? Which is the complexity of such an operation?

The decidability and complexity issues were already mentioned for the question whether a configuration can be reached from another configuration by means of rules in a given set, and this is of a crucial importance (e.g., for the possible applications of our approach). Again, the answer depends in an essential manner on the set of rules – but how exactly this happens it remains to be discovered. Connections with previous investigations of reachability issues will probably be useful, especially for cases where the reachability can be decided – see, e.g., [9], [10].

A series of interesting questions concerns the possible relations of the investigations above with other areas of computer science, of mathematics in general. Processing configurations means to process the underlying trees, too. Can any connection with tree/graph grammars be established? Then, we can naturally define a graph on the set $CFG(O, H, k)$, with the configurations as nodes and the arcs indicating the transitions among configurations with respect to a given set of rules. How this link can be used? Looking for (the number of) connected components in this graph seems to be directly related to the cardinality of generator sets. Furthermore: what about the topological properties of the metric space defined by a distance $edit_R$? Which is the relevance of such connections for membrane computing?

On the same line, interesting connections seems to be possible with evolutionary computing. Assume that we have a criterion according to which some configurations can be considered "better" than others (and that this criterion can be checked in an efficient manner). Then we can try to "improve" the configurations as usual in evolutionary computing: start from an initial population of configurations, apply a given number of rules, randomly, to these configurations, so that a new population is obtained; select from this population only part of configurations, in the decreasing order of their quality; repeat these steps until a satisfactory configuration is obtained (or a given number of steps have been performed). Any application of this strategy?

We close this section with an "exotic" question, that of self-reproduction: by sending membranes out of the skin membrane (by exocytosis – rules of type (a)), we can send a whole configuration/system into the environment. How can this be done in such a way to start from a given configuration (*any* given configuration?) and to send out a copy of it? This is interesting also as an intrinsic problem, and it can be useful both for creating exponential space for solving hard problems and

for creating further and further agents capable of "exploring the extraterrestrial space" in the speculative framework sketched in [5].

10 Final Remarks

We conclude these notes with the belief that the investigations started here deserve further efforts, both because of the mathematical appeal of the many questions raised in this framework, but also because of the interest for membrane computing in general, and for possible applications in particular (e.g., in finding heuristic algorithms for addressing certain decidability or optimization problems).

We hope to return to this topic in a forthcoming paper.

Acknowledgement

The work of Gh. Păun was partially done during a stay as a visiting scientist at MTA SZTAKI EU Center of Excellence, Budapest.

References

1. A. Alhazov, T.-O. Ishdorj: Membrane operations in P systems with active membranes. *Proc. Second Brainstorming Week on Membrane Computing*, Sevilla, February 2004, TR 01/04 of Research Group on Natural Computing, Sevilla University, 2004, 37–44.
2. A. Alhazov, L. Pan, Gh. Păun: Trading polarizations for labels in P systems with active membranes. *Acta Informatica*, 41, 2-3 (2005), 111–144.
3. G. Bel Enguix, M.D. Jiménez-Lopez: Linguistic membrane systems and applications. In *Applications of Membrane Computing* (G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.), Springer-Verlag, Berlin, 2005.
4. L. Cardelli: Brane calculi. Interactions of biological membranes. *Proc. Computational Methods in Systems Biology*, 2004, Springer-Verlag, to appear.
5. C. Calude, Gh. Păun: Bio-steps beyond Turing. *BioSystems*, 77 (2004), 175–194.
6. A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: Weak metrics on configurations of a P system. *Proc. Second Brainstorming Week on Membrane Computing*, Sevilla, February 2004, TR 01/04 of Research Group on Natural Computing, Sevilla University, 2004, 139–151.
7. R. Freund, L. Kari, M. Oswald, P. Sosik: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330, 2 (2005), 251–266.
8. S. Ginsburg: *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland, Amsterdam, 1975.
9. O.H. Ibarra: On the computational complexity of membrane computing systems. *Theoretical Computer Science*, 320, 1 (2004), 98–109.
10. O.H. Ibarra, Z. Dang, O. Egecioglu: Catalytic membrane systems, semilinear sets, and vector addition systems. *Theoretical Computer Science*, 312, 2-3 (2004), 378–400.

11. S.N. Krishna, Gh. Păun: P systems with mobile membranes. *Natural Computing*, to appear.
12. M. Minsky: *Computation. Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
13. N.J. Nilsoon: *Artificial Intelligence. A New Synthesis*. The Morgan Kaufmann Series in Artificial Intelligence, San Francisco, 1998.
14. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
15. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
16. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: *Teoría de la Complejidad en Modelos de Computatión Celular con Membranas*. Editorial Kronos, Sevilla, 2002.