Membrane Systems with Marked Membranes

Robert Brijder[a,1] Matteo Cavaliere[b,c,2]
Agustín Riscos-Núñez[b,3] Grzegorz Rozenberg[a,4]
Dragoş Sburlan[b,d,5]

[a] *Leiden Institute of Advanced Computer Science (LIACS)*
Leiden University
Leiden, The Netherlands

[b] *Department of Computer Science and Artificial Intelligence*
University of Seville
Seville, Spain

[c] *Microsoft Research – University of Trento*
Centre for Computational and Systems Biology
Trento, Italy

[d] *Faculty of Mathematics and Informatics*
Ovidius University of Constantza
Constantza, Romania

Abstract

Membrane computing is a biologically inspired computational paradigm. Motivated by brane calculi we investigate membrane systems which differ from conventional membrane systems by the following features: (1) biomolecules (proteins) can move through the regions of the systems, and can attach onto (and de-attach from) membranes, and (2) membranes can evolve depending on the attached molecules. The evolution of membranes is performed by using rules that are motivated by the operation of pinocytosis (the pino rule) and the operation of cellular dripping (the drip rule) that take place in living cells.
We show that such membrane systems are computationally universal. We also show that if only the second feature is used then one can generate at least the family of Parikh images of the languages generated by programmed grammars without appearance checking (which contains non-semilinear sets of vectors).
If, moreover, the use of pino/drip rules is non-cooperative (i.e., not dependent on the proteins attached to membranes), then one generates a family of sets of vectors that is strictly included in the family of semilinear sets of vectors.
We also consider a number of decision problems concerning reachability of configurations and boundness.

Keywords: Membrane Computing, Brane Calculi

[1] Email: rbrijder@liacs.nl
[2] Email: matteo.cavaliere@msr-unitn.unitn.it
[3] Email: ariscosn@us.es
[4] Email: rozenber@liacs.nl
[5] Email: dsburlan@univ-ovidius.ro

1 Introduction

Membrane computing is a biologically inspired computational paradigm introduced by Gh. Păun in 1998, [10]. The model is based on a hierarchical structure of nested membranes, inspired by the structure of living cells. In each region (enclosed by a membrane) some objects are present, modeling the presence of molecules inside the compartments of living cells. Moreover, each region has an associated set of multiset rewriting rules. These rules are motivated by chemical reactions that occur inside the regions of living cells. Membranes play a crucial role in living cells: the cell membrane separates, and hence protects the cell from its environment and the inner membranes delimit the structure of various organelles of the cell, e.g., the nuclear membrane separates the nucleus from the rest of the cell.

Membranes are not only "containers" but they also regulate the flow of molecules into and out of the cell. This is facilitated by proteins that are embedded in membranes and which provide channels for the transport of molecules through membranes.

In brane calculi, presented in [3], several operations (*pino, exo, phago, mate, drip, bud*) involving membranes with embedded proteins are considered and formalized in the framework of process calculi. The important difference with membrane computing is that the evolution of the system happens *on* the membranes and *not inside* the compartments (regions) delimited by them. The computational power of several brane calculi operations has been investigated in [2] where universality has been obtained for systems using *phago* and *exo*. In [4] these operations from brane calculi have been represented in the membrane computing framework and then studied by using tools from formal language theory.

In this paper we investigate operations involving membranes with embedded proteins, but we also add the ability of proteins to attach/de-attach to/from the membranes, and also to move through the membranes. Hence, in our case, the evolution of the system takes place both on the membranes and inside the regions, which is natural from a biological point of view.

More specifically, we consider *protein-membrane rules* – rules that modify the structure of (the membranes of) the system where the modifications are based on the multisets of proteins embedded in the membranes (we say that such multisets *mark* the membranes). In particular, we consider the *pino* and *drip* rules inspired by the operation of *pinocytosis* and the operation of cellular *dripping*, respectively. Both pinocytosis and dripping split off a membrane from another membrane, however, in pinocytosis, this new (empty) membrane is found inside the original membrane, while in dripping, this new membrane is found outside the original membrane. We also use *protein movement rules*, that model the attachment, de-attachment and movement of the proteins. Also these rules are applied according to the proteins marking the involved membranes. The protein movement rules do not change the membrane structure of the system, but they can change the multisets of embedded proteins marking the membranes of the system.

The paper is structured in the following way. In Section 2 we provide preliminaries concerning formal languages, recalling in particular the definition of programmed grammars often used in the proofs. In Section 3 we recall the formal definition of pino and drip rules, and introduce the protein movement rules, and in Section 4 we introduce membrane systems based on these rules which – the model is called *membrane system with marked membranes, protein-membrane rules, and protein movement rules*, abbreviated as P_{pp} system.

In Section 5, we investigate the computational power of P_{pp} systems which use only protein movement rules, and in Section 6 of P_{pp} systems using only pino (or drip) rules. In Section 7, we discuss P_{pp} systems using both types of rules. In Section 8 we prove several decidability results concerning reachability of configurations and boundness of P_{pp} systems with pino, drip rules, and protein movement rules. In the last section we discuss the results obtained in this paper and formulate a number of research directions.

This paper does not contain any proofs – all the proofs can be found in the full version of this paper [1].

2 Preliminaries

We will briefly recall the main notions and results of formal language theory used in this paper. For more details the reader can consult standard books, such as [8], [13], [7], and the handbook [12].

Given a set A, we denote by $|A|$ its cardinality and by $\mathbb{P}(A)$ the power set of A. The empty set is denoted by $\varnothing$.

As usual, an *alphabet* V is a finite set of symbols. By V^* we denote the set of all strings over V. The empty string is denoted by λ. The *length* of a string $w \in V^*$ is denoted by $|w|$, while the number of occurrences of $a \in V$ in w is denoted by $|w|_a$. For a language $L \subseteq V^*$, the set $length(L) = \{|w| \mid w \in L\}$ is called the *length set* of L. Given a string w, a string u is a *subword* of w if there exist two strings x, y, possibly empty, such that $w = xuy$. The string u is a *scattered subword* of w if and only if there exist strings $x_1, \ldots, x_k$, and $y_0, \ldots, y_k$, possibly empty, such that $u = x_1 \cdots x_k$, and $w = y_0 x_1 y_1 \cdots x_k y_k$. We use $Sub(w)$ to denote the set of all subwords of w, while $Scub(w)$ denotes the set of the scattered subwords of w.

Given an alphabet $V = \{a_1, a_2, \ldots, a_n\}$, with every string $w \in V^*$ we can associate the *Parikh vector* $\Psi_V(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})$, where the ordering $(a_1, \ldots, a_n)$ of V is assumed. Given a language $L \subseteq V^*$, the *Parikh image* of L is defined as $\Psi_V(L) = \{\Psi_V(w) \mid w \in L\}$.

If FL is a family of languages, then $PsFL$ denotes the family of Parikh images of languages in FL (w.r.t. a given alphabet V), and NFL denotes the family of length sets of languages in FL. Note that each $L \in PsFL$ is a set of vectors with a fixed dimension. We denote by FIN, REG, CF, CS, and RE the family of finite, regular, context-free, context-sensitive, and recursively enumerable languages, respectively. Accordingly, the family of Parikh images of languages in RE is denoted by $PsRE$ (this is the family of all recursively enumerable sets of vectors of natural numbers). The family of all recursively enumerable sets of natural numbers is denoted by

NRE. As usual, two language generating/accepting devices are called *equivalent* if they generate/accept the same language.

A context-free *programmed grammar with appearance checking* is a construct $G = (N, T, S, P)$, where N (T, resp.) is a finite set of nonterminals (terminals, resp.), $S \in N$ is the start symbol, and P is a finite set of productions of the form $(b : A \to x, E_b, F_b)$, where b is a label, $A \to x$ with $A \in N$ and $x \in (N \cup T)^*$ is a context-free production, and E_b, F_b are two sets of labels of productions of G (E_b is called the *success field* and F_b the *failure field* of the production). A production $(b : A \to x, E_b, F_b)$ is applied as follows: if A is present in the sentential form, then the production $A \to x$ is applied and the next production is chosen from those with the labels in E_b, otherwise, the sentential form remains unchanged and we choose the next production from the set of productions labeled by some element of F_b. A derivation step is denoted by $\Rightarrow$ while $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$. If no failure field is given for any of the productions, then we obtain a programmed grammar *without appearance checking*.

By PR we denote the family of languages generated by programmed grammars without appearance checking, and by PR_{ac} we denote the family of languages generated by programmed grammars with appearance checking. Proofs of the following results can be found in [7].

Lemma 2.1 $CF \subset PR \subset PR_{ac} = RE$.

We assume the reader to be familiar with the basic notions of membrane computing, see, e.g., [11].

3 Membrane Operations with Marked Membranes

In [3] several membrane operations involving membranes and embedded proteins have been modeled in the framework of process calculi. In [4] these operations have been expressed in the framework of membrane systems.

We will briefly recall these operations, however in a slightly modified form: while in [3] and [4] a region (enclosed by a membrane) can contain other membranes but not objects, we allow a region to contain objects.

As usual in membrane computing, a membrane is represented by a pair of square brackets, $[\,]$. To each membrane $[\,]$ we associate a multiset u (over a certain alphabet V) and this is denoted by $[\,]_u$. We say that the membrane is *marked* with u (u is called a *marking*). The objects of V are called *proteins* or, simply, *objects*. The contents of a membrane can consist of proteins and/or other membranes.

The *protein-membrane rules* over V are of the following form (the subscript i stands for *internal*, e for *external*):

$$pino_i : [\,]_{uav} \to [\,[\,]_{ux}]_v,$$
$$pino_e : [\,]_{uav} \to [\,[\,]_v]_{ux},$$
$$drip : [\,]_{uav} \to [\,]_{ux}[\,]_v.$$

where $a \in V$, and $u, x, v \in V^*$ (thus the restriction of having the right-hand sides of the rules non-empty, as in [4], has been relaxed here). If $uv = \lambda$, then we have a *non-cooperative rule*; we add the prefix $(ncoo)$ to denote it. Thus $(ncoo)pino_i$: $[\,]_a \rightarrow [\,[\,]_x]$ is a non-cooperative $pino_i$ rule.

The described rules are applicable to any membrane whose marking *includes* the multiset indicated in the left hand side of the rules; all the proteins not specified in the rules are not affected by the use of the rules, but they are *randomly distributed* between the two resulting membranes. When using any rule of any type, we say that the membrane from its left hand side is *involved* in the rule; the membrane involved is "consumed" while the membranes from the right hand side of the rule are "produced". Similarly, the protein a specified in the left hand side of the rules is consumed, and it is replaced by the multiset of proteins x (that might be empty).

After the application of a $pino_i$ or $pino_e$ rule, the contents of the consumed membrane is moved into the region of the created external membrane (thus, membrane $[\,]_v$ for $pino_i$ and membrane $[\,]_{ux}$ for $pino_e$), and after the application of a *drip* rule, the contents of the consumed membrane is moved into the region of the produced membrane $[\,]_v$.

We also define rules that can attach/de-attach proteins to/from the membranes, and rules to move the proteins through the membranes of the system. The *protein movement rules* over V can have one of the following forms (the subscript i stands for *inside*, o for *outside*):

$$attach_i : [\,a]_u \rightarrow [\,]_{ua},\ \ attach_o : [\,]_u a \rightarrow [\,]_{ua},$$
$$de{-}attach_i : [\,]_{ua} \rightarrow [\,a]_u,\ \ de{-}attach_o : [\,]_{ua} \rightarrow [\,]_u a,$$
$$move_{out} : [a]_u \rightarrow [\,]_u a,$$
$$move_{in} : [\,]_u\ a \rightarrow [a]_u,$$

with $a \in V$, $u \in V^*$.

The effect of the rules $attach_i$ and $attach_o$ is to attach the protein a to the corresponding membrane if the marking of the membrane *includes* u.

The rules $move_{out}$ ($move_{in}$) move the protein a outside (inside, resp.) if the marking of the corresponding membrane *includes* u. We use *prot* to denote the set of protein movement rules.

4 Membrane Systems with Marked Membranes

In this section we define membrane systems (also called P systems) having membranes marked with multisets of proteins, and using the protein-membrane rules and the protein movement rules introduced in Section 3.

Formally, a *membrane system with marked membranes, protein-membrane rules, and protein movement rules, in short P_{pp} system*, is a construct

$$\Pi = (V, \mu, u_1, \ldots, u_m, R, F),$$

- V is a finite, nonempty alphabet of proteins;
- μ is a membrane structure with $m \geq 1$ membranes;

- $u_1, \cdots, u_m \in V^*$ are the markings of the m membranes of μ at the beginning of the computation (the *initial markings* of Π);
- R is a finite set of protein-membrane rules and protein movement rules over the alphabet V;
- $F \subseteq V$ is the set of *protein-flags*, simply called *flags* (marking the *output membranes*).

We will also use V_Π, μ_Π, R_Π, and F_Π to denote V, μ, R, and F respectively.

A *configuration* of Π consists of a membrane structure, the markings of the membranes, and the multisets of proteins present inside the regions. In what follows, configurations are denoted by writing the markings as subscripts of the right hand parentheses which identify the membranes, e.g., $[\,[\;]_{ab}[aaa]_b[\;]_{bb}]_a$ is an example of a configuration.

We suppose that in the *initial configuration* the regions are empty, thus the initial configuration is defined by μ and $u_1, \ldots, u_m$.

As standard for membrane systems, we assume the existence of a global clock which marks the timing of steps (single *transitions*) for the whole system.

A single transition of Π from a configuration to a new one is performed by applying, *to each membrane of the system, either* (i) *the protein movement rules in the nondeterministic maximally parallel manner, or* (ii) *one of the protein-membrane rules.*

The choice between using protein movement rules or using a protein-membrane rule, for each membrane, is done in a nondeterministic way if both types of rules can be applied for a given membrane. A membrane remains unchanged (only) if no rules can be applied to it.

The application in the nondeterministic maximally parallel manner of the protein movement rules means that, for the chosen membrane, the proteins (the ones marking the membrane and those present in the enclosed region) are assigned with the rules in such a way that, after the assignment is done, no other protein movement rule is applicable to the proteins that have no rules assigned to them. If a protein can be used by several rules, then it is assigned to one of them in a nondeterministic way.

As usual, a sequence of transitions forms a *computation.* A computation which starts from the initial configuration is *successful* if it halts, that is, it reaches a *halting configuration*, i.e., a configuration where no rule can be applied, anywhere in the system. In the halting configuration we consider the *output membranes* – these are membranes whose markings contain at least one flag from F.
Then, the *result* of a successful computation is the set of vectors describing the multiplicities of proteins present in the markings of the output membranes. Because of the non-determinism in the choice of rules, one can get a set of (successful) computations, and thus a set of results.
Collecting all the results, for all possible successful computations, we get *the set of vectors generated* by Π, and denoted by $Ps(\Pi)$.

Note that in a P_{pp} system one computation can deliver a finite family of vectors as its output because several membranes can be “flagged” as output membranes. This differs from assigning the output in “standard” membrane systems. However since the set of vectors $Ps(\Pi)$ generated by a P_{pp} system is taken over the union of results of all successful computations, this difference “disappears” when we consider $Ps(\Pi)$.

We denote by $PP_m(\alpha, prot)$, with $\alpha \in \{pino_i, pino_e, drip, (ncoo)pino_i, (ncoo)pino_e, (ncoo)drip\}$, $m \geq 1$, the class of P_{pp} systems using protein-membrane rules of type α, protein movement rules, and at most m membranes (α or *prot* are removed if the corresponding rules are not used). Therefore $PsPP_m(\alpha, prot)$ is the family of sets of vectors generated by P_{pp} systems from $PP_m(\alpha, prot)$ (α or *prot* are removed if the corresponding rules not used). If m is substituted by $*$ then the number of membranes considered is arbitrary.

Since one cannot mark the empty multiset by a flag, we consider the equality of families of multisets modulo the empty multiset, i.e., if two families differ only by the empty multiset, then we consider them to be equal.

A configuration of a P_{pp} system Π that can be reached by a (possibly empty) sequence of transitions, starting from the initial configuration, is called *reachable*. A multiset w of proteins is a *reachable marking* for Π if there exists a reachable configuration of Π which contains a membrane marked by w.

5 Preliminary Results

We begin with some preliminary results that follow directly from the definitions and from the Turing-Church thesis.

Theorem 5.1

$$PsPP_*(\alpha, prot) \subseteq PsRE, \quad PsPP_*(\alpha) \subseteq PsPP_*(\alpha, prot).$$
$$PsPP_*((ncoo)\alpha, prot) \subseteq PsPP_*(\alpha, prot),$$
$$PsPP_*((ncoo)\alpha) \subseteq PsPP_*(\alpha),$$
$$\alpha \in \{pino_i, pino_e, drip\}.$$

First we consider P_{pp} systems that use only the protein movement rules. The power of such systems is very restricted, even when there is no bound on the number of membranes to be used.

Theorem 5.2 $PsPP_*(prot) = PsFIN$.

6 Membrane Systems Using Protein-Membrane Rules

As stated by Theorem 5.2 the use of only protein movement rules results in a very limited generative power. In this section we turn to the dual situation: the use of protein-membrane rules only.

In this case the membrane structure can change during the computation, but the proteins cannot move through the regions of the system.

First we investigate P_{pp} systems using the non-cooperative versions of the pino and of the drip rules. In this case the power of the system is still very limited: the family of the so generated sets of vectors is strictly included in the family of Parikh images of context-free languages. Then we will study P_{pp} systems using only pino and drip rules; in this case the power of the system increases: one can generate now at least the family of Parikh images of the languages generated by programmed grammars without appearance checking.

Theorem 6.1 $PsPP_*((ncoo)\alpha) \subset PsCF, \alpha \in \{pino_i, pino_e, drip\}$.

The computational power of this class increases when one uses cooperative $pino_i, pino_e$ or *drip* rules. In this case the systems can generate at least the family of Parikh images of languages generated by programmed grammars without appearance checking – it is known that $PsPR$ strictly contains $PsCF$ because it also contains non-semilinear vectors of natural numbers (see [7] for further details).

Formally, we have the following result.

Theorem 6.2 $PsPR \subseteq PsPP_*(\alpha),\ \alpha \in \{pino_i, pino_e, drip\}$.

7 Membrane Systems Using Protein-Membrane and Protein Movement Rules

We will investigate now membrane systems using both protein-membrane rules and protein movement rules. As we will demonstrate the ability to attach, de-attach, and move proteins across the system in a controlled fashion increases the generative power of the systems.

The first indications of the increased generative power is given by Theorem 7.1: P_{pp} systems from $PsPP_*((ncoo)\alpha_i, prot),\ \alpha \in \{pino_i, pino_o, drip\}$, can generate at least the family of Parikh images of context-free languages (compare this result with Theorem 6.1).

Theorem 7.1 $PsCF \subseteq PsPP_*((ncoo)\alpha, prot), \alpha \in \{pino_i, pino_e, drip\}$.

If P_{pp} systems are equipped with both protein-membrane and protein movement rules, then they are computationally complete, in the sense that they are able to generate the family of Parikh images of recursively enumerable languages.

So, informally, it seems that the ability to move the proteins (in a controlled way) through the regions of the system is important for reaching computational completeness. On the other hand, it is interesting to notice that the generative power of protein movement rules, when used alone, is very "weak" (Theorem 5.2).

By comparing the following proof with the proof of Theorem 6.2 we clearly notice similarities. The main difference is the second group of rules, used to simulate the appearance checking mechanism present in the programmed grammar.

Theorem 7.2 $PsPP_*(\alpha, prot) = PsRE,\ \alpha \in \{pino_i, pino_e, drip\}$.

8 Decision Problems

Since the set of proteins attached to a membrane determines the set of rules that can be applied to this membrane, we will consider now the following decision problem: Is it decidable whether or not an arbitrary multiset w is a reachable marking for an arbitrary P_{pp} system?

We will demonstrate that this problem is decidable for P_{pp} systems using (i) only pino and/or drip rules, or (ii) only protein movement rules, while it is not decidable for P_{pp} systems using both pino (or drip) rules and protein movement rules.

Theorem 8.1 *It is undecidable whether or not, for any P_{pp} system Π and any multiset w of proteins over V_Π, w is a reachable marking of Π.*

If P_{pp} systems use only protein movement rules, only pino rules, or only drip rules, then the above problem becomes decidable.

Theorem 8.2 *It is decidable whether or not, for any P_{pp} system Π from $PP_*(prot)$ and any multiset w of proteins over V_Π, w is a reachable marking of Π.*

Theorem 8.3 *It is decidable whether or not, for any P_{pp} system Π from $PP_*(\alpha)$, $\alpha \in \{pino_i, pino_e, drip\}$, and any multiset w of proteins over V_Π, w is a reachable marking of Π.*

We conclude this section by investigating two more decision problems. The first problem concerns the reachability of a configuration in P_{pp} systems. The second problem concerns the boundness of P_{pp} systems.

First, we observe that, given an arbitrary P_{pp} system Π and an arbitrary configuration C of Π, one can compute an upper bound $map_\Pi(C)$ on the number of applications of pino and drip rules that can be used in deriving C from the initial configuration of Π (in case that C is reachable in Π).

Clearly, one can generate in a systematic fashion all reachable configurations of Π containing no more than r membranes. Since each application of a pino or drip rule increases the number of membranes this generation process takes a bounded number of steps. If C appears among these configurations, then it is reachable, otherwise C is not reachable in Π.

Thus, we have the following result:

Theorem 8.4 *It is decidable whether or not, for any P_{pp} system Π and any configuration C of Π, C is a reachable configuration of Π.*

It is perhaps worthwhile to discuss Theorem 8.4 in the light of the universality result stated in Theorem 7.2. The reason that Theorem 8.4 holds is that, for a given configuration C, one can, a priori, provide an upper bound m_c such that C is reachable in Π if and only if it is reachable by computations that do not exceed m_c steps.

On the other hand, if we want to check whether or not a particular multiset w is in the output of a successful computation of Π, then, in general, there is no

	w/o *prot*	*prot*
w/o $pino_i$		PsFIN
$(ncoo)pino_i$	$\subset$ PsCF	$\supseteq$ PsCF
$pino_i$	$\supseteq$ PsPR	PsRE

Table 1
Computational power for P_{pp} systems using $pino_i$ and protein movement rules (*prot*). The same table holds also for $pino_e$ and *drip* operations.

upper bound m_w such that: $w \in Ps(\Pi)$ if and only if w is an output of a successful computation which takes no more than m_w steps.

In fact, in general, there is no relationship between the size of w and the maximal size of a halting configuration in which w is marking one of the output membranes.

A P_{pp} system Π is *bounded* if there exists an integer k, such that, any reachable configuration of Π has less than k membranes.

Theorem 8.5 *It is decidable whether or not an arbitrary P_{pp} system Π from $PP_*(\alpha)$, $\alpha \in \{pino_i, pino_e, drip\}$, is bounded.*

9 Concluding Remarks

We have investigated membrane systems using operations involving membranes marked with multisets of proteins. These systems use two different kinds of operations: the ones that involve membranes and proteins (pino and drip operations) and the ones that attach, de-attach, and move the proteins across the regions of the system (protein movement operations).

Membrane systems using both types of operations are shown to be computationally complete. When the protein-membrane rules are restricted to be non-cooperative, then one generates at least the family of Parikh images of context-free languages.

We have also analyzed membrane systems whose evolution is based on only one of the two types of operations.

In particular we have shown that (in terms of Parikh sets) membrane systems using only pino (or only drip) rules are at least as powerful as programmed grammars without appearance checking.

Our current knowledge about the computational power of membrane systems considered in this paper is summarized in Table 1.

A number of problems have to be settled in order to get a more complete understanding of membrane systems with marked membranes. Some of them are suggested by the results obtained in this paper.

1. Is the inclusion of $PsCF \subseteq PsPP_*((ncoo)\alpha, prot)$ $\alpha \in \{pino_i, pino_e, drip\}$, strict?
2. Is the inclusion $PsPP_*((ncoo)\alpha, prot) \subseteq PsRE$, $\alpha \in \{pino_i, pino_e, drip\}$,

strict?

3. Is the inclusion $PsPR \subseteq PsPP_*(\alpha)$, $\alpha \in \{pino_i, pino_e, drip\}$, strict?
4. Is the inclusion $PsPP_*(\alpha) \subseteq PsRE$, $\alpha \in \{pino_i, pino_e, drip\}$, strict?

Also the following "natural" decision problem should be settled for membrane systems with marked membranes: is it possible to decide whether or not an arbitrary multiset of proteins is a reachable marking for an arbitrary P_{pp} system from $PP_*((ncoo)\alpha, prot)$, with $\alpha \in \{pino_i, pino_e, drip\}$?

The problem is challenging since it is proved to be decidable for P_{pp} systems from $PP_*(prot)$, i.e., using only protein movement rules (see Theorem 8.2), and for P_{pp} systems from $PP_*(\alpha)$, $\alpha \in \{pino_i, pino_e, drip\}$, i.e., using only protein-membrane rules (see Theorem 8.3), while it is undecidable for arbitrary P_{pp} systems (Theorem 8.1).

A more general line of research involves the study of membrane systems having floating molecules and proteins attached to the internal or/and to the external side of a membrane (following, for instance, the idea introduced in [6] where projective brane calculus, with directed actions, has been introduced).

Interesting is also the idea to associate a time of execution to the considered protein rules (following, for instance, the idea of timed P systems introduced in [5]).

We expect several interesting results along these lines of research, bridging membrane systems and brane calculi.

Acknowledgments

The authors are indebted to the European Research Network SegraVis for supporting this research. R. Brijder and G. Rozenberg are supported by the Netherlands Organization for Scientific Research (NWO) project 635.100.006 "VIEWS".

References

[1] R. Brijder, M. Cavaliere, A. Riscos-Núñez, G. Rozenberg, D. Sburlan, Membrane Systems with Proteins Embedded in Membranes. Submitted.

[2] N. Busi, R. Gorrieri, On the Computational Power of Brane Calculi. *Proceedings Third Workshop on Computational Methods in Systems Biology*. Edinburgh, 2005.

[3] L. Cardelli, Brane Calculi. Interactions of Biological Membranes. *Proceedings Computational Methods in System Biology 2004* (V. Danos, V. Schächter, eds.), Lecture Notes in Computer Science, 3082, Springer-Verlag, Berlin, 2005, pp. 257–278.

[4] L. Cardelli, Gh. Păun, An Universality Result for a (Mem)Brane Calculus Based on Mate/Drip Operations. *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects)*, (M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez, eds.), Fénix Ed., Seville, Spain, pp. 75–94. The proceedings can be found at the address `http://www.gcn.us.es/`.

[5] M. Cavaliere, D. Sburlan, Time-Independent P Systems. *Membrane Computing, 5th International Workshop, WMC2004* (G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds.), Lecture Notes in Computer Science, 3365, Springer-Verlag, Berlin, 2005, pp. 239–258.

[6] V. Danos, S. Pradalier, Projective Brane Calculus. *Proceedings Computational Methods in System Biology 2004* (V. Danos, V. Schächter, eds.), Lecture Notes in Computer Science, 3082, Springer-Verlag, Berlin, 2005, pp. 134–148.

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

[8] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[9] H. Lodish, A. Berk, S.L. Zipursky, P. Matsudaira, D. Baltimore, J. Darnell, *Molecular Cell Biology*. Freeman, Fifth Edition.

[10] Gh. Păun, Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), pp. 108–143. First circulated as TUCS Research Report No 28, 1998.

[11] Gh. Păun, *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.

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

[13] A. Salomaa, *Formal Languages*. Academic Press, New York, 1973.

[14] http://psystems.disco.unimib.it