# Membrane systems with proteins embedded in membranes

Robert Brijder [a], Matteo Cavaliere [b,c], Agustín Riscos-Núñez [b], Grzegorz Rozenberg [a], Dragoş Sburlan [b,d]

[a] *Leiden Institute of Advanced Computer Science (LIACS), Universiteit Leiden, 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, Constantza, Romania*

A B S T R A C T

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.

## 1. Introduction

Membrane computing is a biologically inspired computational paradigm introduced by Gh. Păun in 1998, [9]. 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 – 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.

## 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 [8,12,7], and the handbook [11].

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 $\emptyset$.

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 $\lambda$. 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 *generalized sequential machine* (in short *gsm*) is a system $\Gamma = (K, V_1, V_2, s_0, F, \delta)$, where $K$ is a finite set of states, $s_0 \in K$ is the initial state, $F \subseteq K$ the set of final states, and $V_1, V_2$ are the input and output alphabet, respectively. The transition function $\delta$ is defined by $\delta : K \times V_1 \longrightarrow \mathbb{P}(V_2^* \times K)$. For $s, s' \in K$, $a \in V_1$, $y \in V_1^*$, $x, z \in V_2^*$ we write $(x, s, ay) \longmapsto (xz, s', y)$ if $(z, s') \in \delta(s, a)$. Then, for $w \in V_1^*$, we define $\Gamma(w) = \{z \in V_2^* \mid (\lambda, s_0, w) \longmapsto^* (z, s, \lambda), s \in F\}$. The mapping $\Gamma$ is extended in a natural way to languages over $V_1$.

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*.

We denote the set of labels as $Lab(G) = \{b \mid \text{there exists } (b : A \to x, E_b, F_b) \in P\}$. Also, for $X \in N$, we denote $\{b \mid \text{there exists } (b : X \to x, E_b, F_b) \in P\}$ by $l_G(X)$, or $l(X)$ for short.

The language generated by a grammar $G$ is denoted by $L(G)$. 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 1.** *$CF \subset PR \subset PR_{ac} = RE$.*

In *pure* programmed grammars there is no distinction between terminals and nonterminals. Consequently, the language generated by a pure programmed grammar is defined as the set of all strings that can be generated from the axiom, hence the set of all sentential forms. The family of languages generated by pure programmed grammars without appearance checking is denoted by *pPR*. It is easy to prove (in a constructive way) that

**Lemma 2.** *$pPR \subset PR$.*

The following normal form for programmed grammars, referred to as the *$l_h$-normal form*, will be useful in this paper.

**Lemma 3.** *For any programmed grammar $G$ (with appearance checking) there exists an equivalent programmed grammar $G'$ (with appearance checking, respectively) such that there is a unique initial production (with label $l_0$) and a unique final production $Z \to \lambda$ (with label $l_h$) in $G'$.*

**Proof.** Consider an arbitrary programmed grammar $G = (N, T, P, S)$. We recall that $l(S)$ is the set of labels corresponding to productions having $S$ at the left-hand side. Given a set of labels $L$, we use $L'$ to denote the set of the primed version of the labels in $L$.

Let $G' = (N', T, P', S')$, where $N' = N \cup \{S', Z\}$, $S', Z \notin N$, $Lab(G') = Lab'(G) \cup \{l_0, l_h\}$, and $P'$ consists of the following productions:

$$\{(l_0 : S' \to ZS, l'(S), \emptyset), (l_h : Z \to \lambda, \emptyset, \emptyset)\} \cup$$
$$\{(l_i' : A \to \alpha, E_{l_i}' \cup \{l_h\}, F_{l_i}' \cup \{l_h\}) \mid (l_i : A \to \alpha, E_{l_i}, F_{l_i}) \in P\}.$$

It is easily seen that $L(G) = L(G')$, and that the final production in any successful derivation in $G'$ is the one labeled by $l_h$ (deletion of the nonterminal $Z$). The same construction works for both programmed grammars with or without appearance checking (in this last case the failure fields are removed from the productions in $P'$). It is worth to notice that the unsuccessful derivations in $G'$ are of the following types: $S' \Rightarrow^* Z(N \cup T)^*$ or $S' \Rightarrow^* Z(N \cup T)^*N(N \cup T)^* \Rightarrow (N \cup T)^*N(N \cup T)^*$, if $G'$ is without appearance checking while only of the second type if $G'$ is with appearance checking. □

We assume the reader to be familiar with the basic notions of membrane computing, see, e.g., [10].

## 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 nonempty, 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 $(ncoo)pino_i : [\ ]_a \to [\ [\ ]_x]$ is a non-cooperative $pino_i$ rule.

The described rules are applicable to any membrane whose marking *includes* the multiset indicated on 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 on 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*):

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

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;
- $\mu$ is a membrane structure with $m \geq 1$ membranes;
- $u_1, \ldots, u_m \in V^*$ are the markings of the $m$ membranes of $\mu$ at the beginning of the computation (the *initial markings* of $\Pi$);
- $R$ 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 $V_\Pi$, $\mu_\Pi$, $R_\Pi$, and $F_\Pi$ to denote $V$, $\mu$, $R$, and $F$ respectively.

A *configuration* of $\Pi$ 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 $\mu$ 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 $\Pi$ 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. Owing to the nondeterminism 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 $\Pi$, and denoted by $Ps(\Pi)$.

Since a halting configuration in a $P_{pp}$ system can have several membranes marked with $F$, we can have more than one output membrane. Therefore, the output of a successful computation is a finite family of vectors (each vector corresponding to an output membrane). This differs from assigning the output in "standard" membrane systems, where we have only one output vector. 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" in the sense that we can compare the output $Ps(\Pi)$ with the output of "standard" membrane systems.

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 $\alpha$, protein movement rules, and at most $m$ membranes ($\alpha$ 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 $PP_m(\alpha, prot)$ ($\alpha$ 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 $\Pi$ 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 $\Pi$ if there exists a reachable configuration of $\Pi$ 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 4.**

$PsPP_*(\alpha, prot) \subseteq PsRE$, $PsPP_*(\alpha) \subseteq PsPP_*(\alpha, prot)$.

$PsPP_*((ncoo)\alpha, prot) \subseteq PsPP_*(\alpha, prot)$,

$PsPP_*((ncoo)\alpha) \subseteq PsPP_*(\alpha)$,

$\alpha \in \{pino_i, pino_e, drip\}$.

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

**Proof.** The inclusion $PsPP_*(prot) \subseteq PsFIN$ comes from the fact that, using only protein movement rules, it is not possible to increase the total number of objects (and membranes) present in a $P_{pp}$ system during the computation.

On the other hand, the Parikh image of every finite language can be generated by a $P_{pp}$ system from $PP_*(prot)$: in fact, the Parikh image of a finite language can be represented in the initial markings, where each protein is a flag, and actually there is no need to use protein movement rules. Therefore, also the inclusion $PsFIN \subseteq PsPP_*(prot)$ holds and then the theorem follows. □

## 6. Membrane systems using protein-membrane rules

As stated by Theorem 5 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.

First we give an example.

**Example 6.** Consider the regular language $L = \{a^{2^n} \mid n \geq 1\}$. It is easy to show that the Parikh image of $L$, $\Psi_{\{a\}}(L) = \{2n \mid n \geq 1\}$, cannot be generated by a $P_{pp}$ system $\Pi$ from $PP_*((ncoo)pino_i)$. Indeed, suppose that there is such a $\Pi$. Since $L$ is infinite, there is a $x \in L$ with $|x|$ larger than the length of the right-hand side of any $pino_i$ rule of $\Pi$. Thus $x$ has some proteins that were not involved in the application of the last $pino_i$ rule. Since during the application of a $pino_i$ rule one such protein could also have moved to the other membrane, we also have $x - 1 \in \Psi_{\{a\}}(L)$, a contradiction. □

The previous example illustrates that the "random splitting" of the proteins that are not specified in the applied rule is a feature that can reduce the computational power of the system.

**Lemma 7.** *Let $\alpha \in \{pino_i, pino_e, drip\}$. Then, $PsPP_1((ncoo)\alpha) \subseteq PsCF$ iff $PsPP_*((ncoo)\alpha) \subseteq PsCF$.*

**Proof.** The "if" part of the statement obviously holds. We now prove the "only if" part. Let $\Pi = (V, \mu, u_1, \ldots, u_m, R, M) \in PsPP_m((ncoo)\alpha)$ for some $m > 1$. Now, let $\Pi_i = (V, \mu', u_i, R, M) \in PsPP_1((ncoo)\alpha)$ for $i \in \{1, \ldots, m\}$, with $\mu'$ a single membrane initially marked with $u_i$. We have $Ps(\Pi) = Ps(\Pi_1) \cup Ps(\Pi_2) \cup \cdots \cup Ps(\Pi_m)$ if each $\Pi_i$ has a halting configuration, and otherwise $Ps(\Pi) = \emptyset$. Since $CF$ is closed under finite union, we have the desired result. □

**Theorem 8.** $PsPP_*((ncoo)\alpha) \subset PsCF$, $\alpha \in \{pino_i, pino_e, drip\}$.

**Proof.** By Example 8 and Lemma 7 it suffices to show that $PsPP_1((ncoo)\alpha) \subseteq PsCF$. Given a $P_{pp}$ system $\Pi = (V, \mu, u, R, M)$ from $PP_1((ncoo)pino_i)$, we show that one can construct a context-free grammar $G$ such that the Parikh image of $L(G)$ is exactly $Ps(\Pi)$. Note that $\mu$ must be a single membrane. Let $Lab(R) = \{r_1, r_2, \ldots, r_n\}$ be a labeling of the elements of $R$ with $|R| = n$.

We now construct a context-free grammar $G = (N, T, P, S)$, dependent on $\Pi$, with

$$N = \{a \in V \cup \{S\} \mid a \to \alpha \in P \text{ for some } \alpha\}, \text{ where } V \cap \{S\} = \emptyset,$$
$$T = Lab(R) \cup (V - N),$$
$$P = \{a \to r_j x \mid r_j : [\ ]_a \to [\ [\ ]_x\ ] \in R, 1 \le j \le n\} \cup \{S \to u\}.$$

In this way $G$ faithfully simulates $\Pi$ (ignoring the elements of $Lab(R)$), assuming that during each $pino_i$ rule, all proteins move to the inner membrane. We now define a nondeterministic gsm dependent on $G$ which includes the possibility of "random splitting" of proteins (as commented already after Example 8).

It is possible to construct a nondeterministic gsm $\Gamma_G$, dependent on $G$, with input alphabet $T$ and output alphabet $T$, such that the set of output strings on input $y \in T^*$, denoted by $\Gamma_G(y)$, is

$$\{w'x, w' \mid y = w_1 r_j x w_2,\ a \to r_j x \in P,\ w' \in Scub(w_1 w_2)\} \cup U,$$

where $U = \{u\}$ if $u \in T^+$, and $U = \emptyset$ otherwise. For $y \in L(G)$, each decomposition of $y$ into $w_1 r_j x w_2$ with $r = a \to r_j x \in P$ corresponds to a derivation in which $r$ is the last production applied. The other symbols, $w_1 w_2$ were nondeterministically distributed to the outer membrane and the inner membrane. Therefore, $w'x$ ($w'$, resp.) represents the set of markings of the inner (outer, resp.) membrane. In the special case when there is no such decomposition of $y$, we have $y = u \in T^+$ and $u$ is a marking of a "halting" membrane.

Let $h$ be the morphism which deletes the elements of $Lab(R)$, formally defined by $h(a) = \lambda$ for $a \in Lab(R)$, and $h(a) = a$ for $a \in T - Lab(R)$. Now, $\Psi_T(L)$ with $L = h(\Gamma_G(L(G)))$ precisely represents the set of multisets marking the reachable halting membranes of $\Pi$. Since context-free languages are closed under gsm mapping and applications of morphisms, we have $L \in CF$. We now only need to select those multisets of $L$ which contain proteins of $M$. Therefore, $\Psi_T(L') = Ps(\Pi)$ for context-free $L' = L \cap V^* M V^*$ (context-free languages are closed under intersection with regular languages). □

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

**Proof.** Consider a programmed grammar $G = (N, T, P, S)$ without appearance checking in the $l_h$-normal form (see Lemma 3).

We construct a $P_{pp}$ system $\Pi$ from $PP_*(pino_i)$ that generates exactly the Parikh image of $L(G)$ and it is defined as follows:

$$\Pi = (V, \mu, u_1, u_2, R, F),$$

where

- $V = N \cup T \cup \{E\} \cup Lab(G) \cup \{\#\}$,
- $\mu = [\ [\ ]_{ESl_0}\ ]_\lambda$,
- $F = T$.

The pino rules in $R$ are grouped as follows (the grouping is done according to their intended use):

1. *(simulation of the programmed grammar productions)*,
$[\ ]_{EAi} \to [\ [\ ]_{Exj}\ ]_i$, for $(i : A \to x, E_i) \in P, j \in E_i, i \ne l_h$,
2. *(used if a production of G cannot be applied)*,
$[\ ]_A \to [\ [\ ]_{E\#\#}]$, $A \in N$,
3. *(used for non-halting)*,
$[\ ]_{E\#\#} \to [\ [\ ]_{E\#\#}\ ]_\#$,
4. *(used to keep the symbols from a sentential form on the same membrane)*,
$[\ ]_{Xi} \to [\ [\ ]_{E\#\#}\ ]_i$, for $X \in (N \cup T)$, $i \in Lab(G)$,
5. *(used to halt the computation)*,
$[\ ]_{EZl_h} \to [\ [\ ]\ ]_{El_h}$.

The so-constructed system $\Pi$ simulates $G$ in the following way. The structure of the system contains at any time during the computation a unique innermost membrane. The marking of this membrane contains the object $E$ (except in the last

step of the computation), the label of the next production of $G$ to simulate (at the beginning this label is $l_0$), and the objects corresponding to the current sentential form (at the beginning of a computation of $\Pi$ only the object $S$).

The objects from $N$ are called nonterminal objects, while the objects from $T$ are called terminal objects.

The simulation of the application of a production in $G$ is done by using one of the rules in group 1.

The object $A$ is rewritten to $x$ if $(i : A \to x, E_i) \in P$.

Also the label $j$ of the next production is produced while the old one, $i$, is stored on the created external membrane. Note that we should "trash" the computation of $\Pi$ if no productions of $G$ can be simulated and there are nonterminal objects attached to the innermost membrane of $\Pi$. This is accomplished by the rules in groups 2 and 3.

Indeed, if no rule from group 1 can be applied, then a rule of group 2 must be applied, because of the maximal parallelism. If this rule is applied, then the membrane $[\ ]_{E\#\#}$ is created and the rule in group 3 is applied forever; thus the computation does not halt – the computation is "trashed".

We have to guarantee that, after a rule of group 1 is applied (simulating the application of a production from $G$), the objects not modified in the sentential form do not go to the created external membrane. To this aim the rules of group 4 are used. In fact, if any object of the sentential form is attached to a membrane not containing the object $E$ (that is always attached only to the innermost membrane), then the computation of $\Pi$ never halts.

To make the computation halting, the symbols $E$ and $l_h$ must be removed from the innermost membrane, and the nonterminal object $Z$ must be erased. To this aim we use the rules of group 5. Notice that this should be done only when the sentential form is composed by only terminal objects. In fact, if this is not true, then in the next step, a rule of group 2 is applied, and then the computation will never halt.

Thus, from the above explanation, it follows that, any successful derivation of $G$ producing $w$ can be simulated by a successful computation in $\Pi$ halting in a configuration containing a unique innermost membrane, which is also the unique output membrane which is marked by the multiset $\Psi_V(w)$.

On the other hand, unsuccessful derivations of $G$ can be of the type $S \Rightarrow^* Z(N \cup T)^*$ or of the type $S \Rightarrow^* Z(N \cup T)^* N (N \cup T)^* \Rightarrow (N \cup T)^* N (N \cup T)^*$. The simulation in $\Pi$ of these two types of derivations results in non-halting computations.

Therefore $Ps(\Pi)$ is exactly the Parikh image of the language generated by the grammar $G$.

The proof given can be easily adapted using only $pino_e$ or using only *drip* rules. Therefore the theorem follows. □

## 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 10: $P_{pp}$ systems from $PsPP_*((ncoo)\alpha, prot)$, $\alpha \in \{pino_i, pino_o, drip\}$, can generate at least the family of Parikh images of context-free languages (compare this result with Theorem 8).

**Theorem 10.** $PsCF \subseteq PsPP_*((ncoo)\alpha, prot)$, $\alpha \in \{pino_i, pino_e, drip\}$.

**Proof.** Given a context-free grammar $G = (N, T, P, S)$ one can construct a $P_{pp}$ system $\Pi$ from $PP_*((ncoo)pino_i, prot)$ such that $Ps(\Pi)$ is exactly the Parikh image of $L(G)$. Without loss of generality we suppose that each nonterminal is at the left-hand side of at least one production of the grammar.

We construct $\Pi = (V, \mu, u_1, u_2, R, F)$ with $V = N \cup T \cup \{t, E\}$, $F = T$, and $\mu = [\ [\ ]_{St}\ ]_E$.

The rules of $R$ are grouped according to their intended use:

1. *(Pino rules)*,

$[\ ]_a \to [\ [\ ]_{tx}\ ]$ for $a \to x \in P$,

2. *(protein movement rules – movement of terminal objects)*,

$[\ ]_{ta} \to [\ ]_t a$, for $a \in T$,

$[a] \to [\ ]a$, for $a \in T$,

$[\ ]a \to [a]$, for $a \in T$,

$[a]_E \to [\ ]_{aE}$, for $a \in T$,

$[\ ]_E \to [\ ]E$,

3. *(protein movement rules – movement for non-halting)*,

$[\ ]_{tt} \to [\ ]_t t$,

$[t] \to [\ ]t$,

$[\ ]t \to [t]$.

Intuitively, $\Pi$ simulates the context-free productions of $G$ using the pino rules, and when terminal objects are created, they are collected on the skin membrane. We will show that during the computation of $\Pi$ the membrane structure is such that the marking of the skin membrane contains exactly one copy of $E$ and no copies of $t$, while the markings of the other membranes contain exactly one copy of $t$ and no copies of $E$.

The system $\Pi$ works in the following way. The rules of group 1 simulate the rewriting in $G$. Each time a pino rule is applied, then the "special" object $t$ is also attached to the created internal membrane. If a membrane with two (or more)

objects $t$ attached is produced, then the computation will not halt because at least a membrane with a marking containing two objects $t$ is produced, and then the rules from group 3 can be used forever.

This guarantees that each membrane present in the system, except the skin membrane, is marked with objects from $N \cup T$ and exactly one $t$. This object $t$ is used to de-attach the terminal objects from the membranes and to make them migrate toward the skin membrane where they remain attached. This is done by using the rules from group 2 (this process of migration can start at any moment during the computation; it does not interfere with the result of the computation).

Finally, the object $E$ attached to the skin is removed; it can be removed only when all objects from $T$ present in $\Pi$ have been moved and attached to the skin membrane, otherwise these objects move through the regions of the system forever and the computation will not halt.

In this way for any string $w$ in $L(G)$ one can obtain, at the end of a halting computation, a marking of the skin membrane corresponding to the Parikh vector of $w$. In fact, this can be done by applying the pino rules and then moving all the objects from $T$ to the skin membrane in the above described way.

On the other hand, each multiset $w$ produced by $\Pi$ is a marking of the skin membrane in a halting configuration, and it can be only obtained in the way described above – hence, there exists a derivation in $G$ that produces a string with its Parikh vector corresponding to $w$.

The proof can be adapted for systems using $(ncoo)pino_e$ or $(ncoo)drip$ rules by adapting the protein movement rules. Hence, the theorem holds. □

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).

By comparing the following proof with the proof of Theorem 9 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 11.** $PsPP_*(\alpha, prot) = PsRE, \alpha \in \{pino_i, pino_e, drip\}$.

**Proof.** We first prove the theorem for systems from $PP_*(pino_i, prot)$.

The inclusion $PsPP_*(pino_i, prot) \subseteq PsRE$ follows from the Church–Turing thesis. The reverse inclusion, $PsRE \subseteq PsPP_*(pino_i, prot)$ can be proved by simulating a programmed grammar with appearance checking $G$ by a $P_{pp}$ system $\Pi$ from $PP_*(pino_i, prot)$.

To this aim, consider $G = (N, T, P, S)$ in the $l_h$-normal form. Let $Lab'(G) = \{i' \mid i \in Lab(G)\}$.

The $P_{pp}$ system $\Pi$ is defined as follows.

$\Pi = (V, \mu, u_1, u_2, R, F)$, where

- $V = N \cup T \cup \{E\} \cup Lab(G) \cup Lab'(G) \cup \{\#, d, d', h\} \cup \{h'_i, h''_i \mid i \in Lab(G)\}$;
- $\mu = [\ [\ ]_{ESl_0h}\ ]_\lambda$;
- $F = T$;
- The pino rules and the protein movement rules in $R$ are given in groups, according to their intended use during the simulation of $G$ by $\Pi$.

1. *(simulation of the productions of the programmed grammar)*,
$[\ ]_{hEAi} \to [\ [\ ]_{hExj}]_i$, for $(i : A \to x, E_i, F_i) \in P, j \in E_i, i \neq l_h$,
2. *(simulation of the skipping of a production – appearance checking)*,
$[\ ]_{Ehi} \to [\ [\ ]_{Eh'_ih''_i}]_i$, $i \in Lab(G), i \neq l_h$,
$[\ ]_{Eh'_ih''_i} \to [\ ]_{Eh'_i}h''_i$, $[\ ]_{Eh''_i} \to [\ [\ ]_{E\#\#}]_{h''_i}$, $i \in Lab(G), i \neq l_h$,
$[\ ]_{Eh'_iA} \to [\ [\ ]_{E\#\#}]_{h'_iA}$, for $(i : A \to x, E_i, F_i) \in P$,
$[h''_i]_i \to [\ ]_{h''_ii}$, $i \in Lab(G)$,
$[\ ]_{h''_ii} \to [\ [\ ]_{j'd}]_i$, $i \in Lab(G), j \in F_i$,
$[\ ]_{j'd} \to [\ ]_dj'$, $j \in Lab(G)$,
$[\ ]_Ej' \to [\ ]_{Ej'}$, $j \in Lab(G), j \neq l_0, j \neq l_h$,
$[\ ]_{Ej'h'_i} \to [\ [\ ]_{Ejh}]_{j'}$, $i, j \in Lab(G)$,
3. *(used to produce non-halting)*,
$[\ ]_{E\#\#} \to [\ [\ ]_{E\#\#}]_\#$,
4. *(used to keep the symbols from a sentential form on the same membrane)*,
$[\ ]_{Xi} \to [\ [\ ]_{E\#\#}]_i$, $X \in (N \cup T), i \in Lab(G) \cup Lab'(G)$,
5. *(used to halt a computation)*,
$[\ ]_{El_h} \to [\ ]_{l_h}E$,
$[\ ]_{l_hZ} \to [\ ]_{l_h}Z$,
$[\ ]_{l_hh} \to [\ ]_{l_h}h$,
$[\ ]_{l_h} \to [\ ]l_h$,

$[\ ]_E l_h \to [\ ]_{El_h}$,
$[\ ]_Z l_h \to [\ ]_{Zl_h}$,
$[\ ]_h l_h \to [\ ]_{hl_h}$,
$[\ ]_X l_h \to [\ ]_{Xl_h}$, $X \in N$.

The so-constructed system $\Pi$ works as follows.

Each computation starts from the initial configuration $[\ [\ ]_{ESl_0h}\ ]_\lambda$. At each step, there is a unique membrane of $\Pi$ marked by a multiset composed by the object $E$, the objects corresponding to the current sentential form of $G$ (only the object $S$ at the beginning of a computation), the label of the next production of $G$ to simulate ($l_0$ at the beginning of a computation), and the "support" object $h$.

The pino rules of group 1 simulate the application of a production $(i : A \to x, E_i, F_i)$ of $G$, not used in the appearance checking mode (i.e., the nonterminal $A$ is present in the multiset marking the membrane to which $E$ is attached). In this case the pino rule corresponding to the production $A \to x$, with label $i$ is applied, and together with the objects $x$ also the label of the next production to simulate ($j \in E_i$) is produced.

If a production cannot be applied (because $A$ is not present in the multiset marking the membrane to which $E$ is attached), then the production has to be used in the appearance checking mode (i.e., has to be skipped) and, for this goal, the rules of group 2 are used. The system "guesses", by applying a rule $[\ ]_{Ehi} \to [\ [\ ]_{Eh'_ih''_i}]_i$ from group 2, that the production of $G$ with label $i$ that should be currently simulated, cannot be executed.

For instance, consider the configuration $[\ \cdots [\ ]_{hExi}\ \cdots]$, with $x$ a string representing a multiset over $N \cup T$ (it represents the current sentential form of $G$) and $i$ the label of the next production of $G$ to simulate. By applying $[\ ]_{Ehi} \to [\ [\ ]_{Eh'_ih''_i}]_i$ we obtain the configuration $[\cdots [\ [\ ]_{Eh'_ih''_ix}]_i \cdots]$. Then, the rule $[\ ]_{Eh'_ih''_i} \to [\ ]_{Eh'_ih''_i}$ is applied (the other rules that could be applied lead to a non-halting computation). So the configuration $[\cdots [\ [\ ]_{Eh'_ix}h''_i]_i \cdots]$ is obtained. In the next step, in parallel, the rule $[\ ]_{Eh'_iA} \to [\ [\ ]_{E\#\#}]_{h'_iA}$ with $(i : A \to x, E_i, F_i) \in P$ is applied (if possible) and the rule $[h''_i]_i \to [\ ]_{h''_ii}$ is applied (this is certainly possible). If the first rule is applied, then this means that the production with label $i$ could be simulated and the (guess) decision to skip it was wrong. In this case the proteins $E\#\#$ are attached to the created membrane and the computation never halts. If the first rule is not applied, then the next configuration reached is $[\cdots [\ [\ ]_{Eh'_ix}\ ]_{h''_ii} \cdots]$. Now only the pino rule $[\ ]_{h''_ii} \to [\ [\ ]_{j'd}]_i$, $j \in F_i$, can be applied. Therefore, the next configuration obtained is the following one (notice the movement of the contents in the pino operation): $[\cdots [\ [\ ]_{j'd}[\ ]_{Eh'_ix}]_i \cdots]$. Now the protein $j'$ is de-attached using the rule $[\ ]_{j'd} \to [\ ]_dj'$. So the next configuration obtained is $[\cdots [\ [\ ]_dj'[\ ]_{Eh'_ix}]_i \cdots]$. The protein $j'$ is added to the marking of the membrane where the protein $E$ is already attached, by using $[\ ]_Ej' \to [\ ]_{Ej'}$. In this way the configuration $[\cdots [\ [\ ]_d[\ ]_{j'Eh'_ix}]_i \cdots]$ is obtained. Finally, the pino rule $[\ ]_{Ej'h'_i} \to [\ [\ ]_{Ejh}]_{j'}$ can be applied. So the next configuration is $[\cdots [\ [\ ]_d[\ [\ ]_{Ejhx}]_{j'}]_i \cdots]$. Therefore, the process can be iterated by applying (or skipping) the production of $G$ with label $j$, in the way described above. Also, the rules of group 4 ensure that the sentential form is always entirely attached to the membrane where $E$ is attached, and it is never divided randomly between the membranes created by the pino operation.

The correct halting of the computation is assured by applying the rules of group 5. The computation can be halted when the production with label $l_h$ should be simulated. In this case, it is necessary to remove the nonterminal $Z$ and to check that the objects attached to the membrane containing the current sentential are terminals.

For instance, consider the configuration $[\ \cdots [\ ]_{hExl_h}\ \cdots]$, where $x$ is a string representing a multiset over $V$ (the current sentential form of $G$). First, the proteins $E, Z$ (present in $x$) and $h$ are de-attached, obtaining the configuration $[\ \cdots [\ ]_{x'l_h}hEZ\ \cdots]$. Then, finally, also $l_h$ is released yielding to the configuration $[\ \cdots [\ ]_{x'}l_hhEZ\ \cdots]$. The release of $l_h$ cannot be done earlier, since otherwise $l_h$ would be re-attached because of the presence of $E$, $Z$ or $h$. Once $l_h$ has been released, it is attached again to the same membrane if and only if a nonterminal object is attached to this membrane (rule $[\ ]_Xl_h \to [\ ]_{Xl_h}$); this label is released again by the rule $[\ ]_{l_h} \to [\ ]l_h$ and this attachment/de-attachment runs forever. This guarantees that, when the computation halts, the unique output membrane (the one where is attached at least a flag) contains only objects from $T$.

From the above explanation it is easily seen that any successful derivation of $G$ producing $w$ can be simulated in $\Pi$ by a successful computation halting in a configuration with a unique output membrane marked with the multiset $\Psi_T(w)$. The simulation in $\Pi$ of this type of derivation leads to a non-halting computation.

Therefore it follows that $Ps(\Pi)$ is exactly the Parikh image of $L(G)$.

Moreover the proof given can be easily adapted by using $pino_e$ or by using *drip* rules (by adjusting the protein movement rules). Thus the theorem follows. □

## 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 12.** *It is undecidable whether or not, for any $P_{pp}$ system $\Pi$ and any multiset $w$ of proteins over $V_\Pi$, $w$ is a reachable marking of $\Pi$.*

**Proof.** *Sketch.* The result follows from the universality results proved in Theorem 11.

For any programmed grammar $G$ with appearance checking it is possible to construct a $P_{pp}$ system $\Pi$ that can simulate the derivations in $G$. Consider now the construction given in Theorem 11. If there exists an algorithm to check if an arbitrary multiset $w$ is a reachable marking of $\Pi$, then the same algorithm together with the construction from Theorem 11, could be used to decide whether or not for an arbitrary Parikh vector $v$ a sentential form $z$ with Parikh vector equal to $v$ can be generated in $G$. This, however, contradicts the universality of programmed grammars with appearance checking (which has been proved in a constructive way, see [7]). □

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

**Theorem 13.** *It is decidable whether or not, for any $P_{pp}$ system $\Pi$ from $PP_*(prot)$ and any multiset $w$ of proteins over $V_\Pi$, $w$ is a reachable marking of $\Pi$.*

**Proof.** Given a $P_{pp}$ system $\Pi$ from $PP_*(prot)$ the number of possible distinct reachable configurations for $\Pi$ is finite and therefore the problem is decidable (e.g., by using an exhaustive search). □

**Theorem 14.** *It is decidable whether or not, for any $P_{pp}$ system $\Pi$ from $PP_*(\alpha)$, $\alpha \in \{pino_i, pino_e, drip\}$, and any multiset $w$ of proteins over $V_\Pi$, $w$ is a reachable marking of $\Pi$.*

**Proof.** We first show that the set of strings representing all the reachable markings for $\Pi \in PP_*(pino_i)$ can be generated by a programmed grammar $G$ without appearance checking.

Let $\Pi = (V, \mu, p_1, p_2, \ldots, p_m, R, F)$ where $R = \{r_1, \cdots, r_k\}$.

In what follows, in order to avoid writing an entire pino rule $r_i : [\ ]_{uav} \to [\ [\ ]_{ux}\ ]_v$ we will often refer directly to the strings $u$, $v$, and $x$ and to the symbol $a$. Thus, e.g., we may write "consider a string $u$ of $r_i$".

We also use the morphism $h : V \longrightarrow V'$ defined by $h(x) = x'$, $x \in V$, and $V' = \{a' \mid a \in V\}$.

Let $G = (N, T, P, S)$ be a pure programmed grammar, thus $N = T$, where $N$ is the set $V \cup V'$.

In what follows, in order to simplify the notation, we assume that several productions of $G$ can have the same label (in this case the production to be applied is chosen nondeterministically among the ones with the same label). Note that this assumption is only a notational convenience: it is easy to see that for each such $G$ there is an equivalent pure programmed grammar having an injective labeling of the productions.

For the sake of readability we will use $l_{beg}$ to denote the set of labels of $G$ which correspond to productions that are used to initiate the simulation of pino rules. Thus, $l_{beg}$ (the set of beginning labels) is defined as

$$\begin{aligned} l_{beg} &= \{l'_{r_i,1} \mid r_i \text{ has } u \neq \lambda,\ 1 \leq i \leq k\} \\ &\cup\ \{l'_{r_i} \mid r_i \text{ has } u = \lambda,\ 1 \leq i \leq k\} \cup \{l''_{r_i,1} \mid r_i \text{ has } v \neq \lambda,\ 1 \leq i \leq k\} \\ &\cup\ \{l''_{r_i,r+1} \mid r_i \text{ has } v = \lambda,\ 1 \leq i \leq k\}. \end{aligned}$$

The label used to start the simulation of a pino rule $r_i$ can be different according to the presence of strings $u$ and $v$ in the rule $r_i$. In fact, if in the chosen rule $r_i$, $u$ or/and $v$ are missing, then some of the productions of $G$ need to be skipped.

The set of productions $P$ is divided into several groups, according to their intended use during the simulation of $\Pi$.

1. *(nondeterministic choosing of one membrane and of one pino rule),*

$(l_0 : S \to p_i, l_{beg})$, for $1 \leq i \leq n$,

If the pino rule $r_i : [\ ]_{uav} \to [\ [\ ]_{ux}\ ]_v$, $1 \leq i \leq k$, is present in $R$, with $u = u_1u_2 \cdots u_j$, $v = v_1v_2 \cdots v_r$ and $x = x_1x_2 \cdots x_p$, then we add to $P$ the following productions.

2(a). *(prime the symbols of the string u),*

$(l'_{r_i,1} : u_1 \to h(u_1), \{l'_{r_i,2}\})$,

$(l'_{r_i,2} : u_2 \to h(u_2), \{l'_{r_i,3}\})$,

$\cdots$

$(l'_{r_i,j} : u_j \to h(u_j), \{l'_{r_i}\})$,

3(a). *(prime the symbol a),*

$(l'_{r_i} : a \to h(a), \{l'_{r_i,j+1}\})$, if $r_i$ has $v \neq \lambda$,

$(l'_{r_i} : a \to h(a), \{l'_{r_i,j+r+1}, l_{1,i}\})$, if $r_i$ has $v = \lambda$,

4(a). *(delete the symbols of v),*

$(l'_{r_i,j+1} : v_1 \to \lambda, \{l'_{r_i,j+2}\})$,

$(l'_{r_i,j+2} : v_2 \to \lambda, \{l'_{r_i,j+3}\})$,

$\cdots$

$(l'_{r_i,j+r} : v_r \to \lambda, \{l'_{r_i,j+r+1}, l_{1,i}\})$,

5(a). *(delete nondeterministically),*

$(l_{1,i} : d \to d, \{l'_{r_i,j+r+2}\})$, $d \in N$, if $r_i$ has $u \neq \lambda$,

$(l_{1,i} : d \to d, \{l'_{r_i,2j+r+2}\})$, $d \in N$, if $r_i$ has $u = \lambda$,

$(l'_{r_i,j+r+1} : d \to \lambda, \{l'_{r_i,j+r+1}, l'_{r_i,j+r+2}\})$, $d \in V$, if $r_i$ has $u \neq \lambda$,

$(l'_{r_{i,j+r+1}} : d \to \lambda, \{l'_{r_{i,j+r+1}}, l'_{r_{i,2j+r+2}}\})$, $d \in V$, if $r_i$ has $u = \lambda$,
6(a). *(de-prime the symbols of u)*,
$(l'_{r_{i,j+r+2}} : u'_1 \to u_1, \{l'_{r_{i,j+r+3}}\})$,
$(l'_{r_{i,j+r+3}} : u'_2 \to u_2, \{l'_{r_{i,j+r+4}}\})$,
$\cdots$
$(l'_{r_{i,2j+r+1}} : u'_j \to u_j, \{l'_{r_{i,2j+r+2}}\})$,
7(a). *(apply $a \to x$ and choose the next pino rule)*,
$(l'_{r_{i,2j+r+2}} : a' \to x, l_{beg})$,
2(b). *(prime the symbols of v)*,
$(l''_{r_{i,1}} : v_1 \to h(v_1), \{l''_{r_{i,2}}\})$,
$(l''_{r_{i,2}} : v_2 \to h(v_2), \{l''_{r_{i,3}}\})$,
$\cdots$
$(l''_{r_{i,r}} : v_r \to h(v_r), \{l''_{r_{i,r+1}}\})$,
3(b). *(prime the symbol a)*,
$(l''_{r_{i,r+1}} : a \to h(a), \{l''_{r_{i,r+2}}\})$, if $r_i$ has $u \neq \lambda$,
$(l''_{r_{i,r+1}} : a \to h(a), \{l''_{r_{i,r+j+2}}, l_{2,i}\})$, if $r_i$ has $u = \lambda$,
4(b). *(delete the symbols of u)*,
$(l''_{r_{i,r+2}} : u_1 \to \lambda, \{l''_{r_{i,r+3}}\})$,
$(l''_{r_{i,r+3}} : u_2 \to \lambda, \{l''_{r_{i,r+4}}\})$,
$\cdots$
$(l''_{r_{i,r+j+1}} : u_j \to \lambda, \{l''_{r_{i,r+j+2}}, l_{2,i}\})$,
5(b). *(delete nondeterministically)*,
$(l_{2,i} : d \to d, \{l''_{r_{i,r+j+3}}\})$, $d \in N$,
$(l''_{r_{i,r+j+2}} : d \to \lambda, \{l''_{r_{i,r+j+2}}, l''_{r_{i,r+j+3}}\})$, $d \in V$,
6(b). *(delete the symbol $a'$)*,
$(l''_{r_{i,r+j+3}} : a' \to \lambda, \{l''_{r_{i,r+j+4}}\})$, if $r_i$ has $v \neq \lambda$,
$(l''_{r_{i,r+j+3}} : a' \to \lambda, l_{beg})$, if $r_i$ has $v = \lambda$,
7(b). *(de-prime the symbols of v and choose the next pino rule)*,
$(l''_{r_{i,r+j+4}} : v'_1 \to v_1, \{l''_{r_{i,r+j+5}}\})$,
$(l''_{r_{i,r+j+5}} : v'_2 \to v_2, \{l''_{r_{i,r+j+6}}\})$,
$\cdots$
$(l''_{r_{i,r+j+4+r-1}} : v'_r \to v_r, l_{beg})$,

The so-constructed grammar $G$ simulates $\Pi$ in the following way.

The underlying idea is that $G$ stores in its sentential forms the strings corresponding to reachable markings of $\Pi$ (with one reachable marking stored in one sentential form).

The grammar simulates, by using its productions, the evolution of a single membrane from a reachable configuration of $\Pi$; if a membrane has several possible evolutions then $G$ "chooses" only one of them in a nondeterministic fashion. When a pino rule is simulated, then the grammar chooses, nondeterministically, to follow the evolution of either the created internal membrane or the created external membrane.

Initially, the grammar $G$ applies one of the productions present in the group 1, having label $l_0$. So the symbol $S$ is rewritten in a nondeterministic way into one of the strings $p_1, p_2, \ldots, p_m$ corresponding to the initial markings of $\Pi$. The choice can be done in a nondeterministic manner since in the systems we consider here, the evolution of a membrane present in a certain configuration is independent from the evolution of the other membranes present in the same configuration.

The next production is selected by choosing, in a nondeterministic way, a label in the set $l_{beg}$ associated with the production with label $l_0$.

We will discuss the functioning of $G$ when it simulates a pino rule in which both contexts $u$ and $v$ are nonempty (the reader can easily verify the functioning of $G$ in the case when one or both contexts are empty).

Given a pino rule $r_i : [\ ]_{uav} \to [\ [\ ]_{ux}\ ]_v$, the label $l'_{r_{i,1}}$ is used to start the sequence of productions that simulate the pino rule $r_i$, in the case $G$ follows the evolution of the created internal membrane; on the other hand $l''_{r_{i,1}}$ is used to start the sequence of productions that simulate the pino rule $r_i$ in the case $G$ follows the evolution of the created external membrane.

The productions in group (a) are used in the former case, while the productions in group (b) are used in the latter case.

(i): We now analyze the former case. Thus we suppose that, after applying the production labeled by $l_0$, we have chosen the production with label $l'_{r_{i,1}}$ and $r_i$ is the pino rule $[\ ]_{uav} \to [\ [\ ]_{ux}\ ]_v$. Therefore, the grammar simulates the rule $r_i$ and chooses to follow the evolution of the created *internal* membrane.

First, the productions of the group 2(a) are executed in sequence – they are used to mark all the symbols in the sentential form corresponding to the objects of the string $u$ of the pino rule $r_i$.

Then the production $a \to h(a)$ is applied, and the object $a$ is primed (group 3(a)).

After that, the productions of group 4(a) are applied in sequence (we suppose $v \neq \lambda$). This corresponds to the deletion from the sentential form of the objects from the string $v$ of the pino rule $r_i$. These objects are deleted because the grammar has chosen to follow the evolution of the created internal membrane.

When this phase is completed, then some (possibly none) of the symbols in the sentential form are randomly deleted. This is used to simulate the random distribution of the objects between the two newly created membranes, and in particular the deletion simulates the nondeterministic distribution of some of the objects to the created external membrane.

This deletion can be stopped by choosing to execute the production with label $l'_{r_{i,j+r+2}}$ (notice that the deletion can be even totally skipped by choosing the special "dummy" production with label $l_{1,i}$).

When the deletion is stopped, then the introduced symbols of $u$ are de-marked by applying, in sequence, the productions from group 6(a), and finally the symbols of the string $x$ are introduced by using the rules of group 7(a). Moreover, when this last production is applied, a new pino rule is nondeterministically selected by choosing a label in $l_{beg}$ and the above described process is repeated.

(ii): Now we analyze the latter case. Thus we suppose that, after executing the production with label $l_0$, we have chosen the production with label $l''_{r_{i,1}}$ where the pino rule $r_i$ is $[\ ]_{uav} \rightarrow [\ [\ ]_{ux}\ ]_v$. Therefore, the constructed grammar simulates the application of the pino rule $r_i$ with the choice to follow the evolution of the created *external* membrane.

This is done by applying, in a way analogous to the one described above, the rules of the group (b), in the order described by groups 1, 2(b), 3(b), 4(b), 5(b), 6(b), and 7(b).

Since $G$ is pure, the language $L(G)$ consists of all reachable sentential forms. Notice that during the intermediate steps of the simulation of a pino rule there are always primed symbols in the sentential forms produced by $G$.

By Lemma 2, $L(G)$ can also be generated by a programmed grammar without appearance checking.

We are interested in the set of strings corresponding to reachable markings of $\Pi$ and to obtain this set, we only need to intersect $L(G)$ with the regular set $V^*$, filtering out in this way the strings from $L(G)$ containing primed symbols (note that these are the only strings in $L(G)$ not corresponding to reachable markings of $\Pi$).

The family of languages generated by programmed grammars without appearance checking is closed under intersection with regular sets (see, e.g., [7]); therefore, the language $L_{reach} = L(G) \cap V^*$ can also be generated by such grammars (the proof of this closure property is constructive, i.e., we can construct the grammar generating $L_{reach}$ starting from $G$ and from the automaton for $V^*$).

Therefore to check if a multiset of proteins $w$ is a reachable marking of $\Pi$, we only need to decide if (any permutation of) the string $w$ is in $L_{reach}$, and this is decidable (see, e.g., [7]).

Since the membrane structure is not really important in the described simulation then it is easy to adapt the given proof for $P_{pp}$ systems using only $pino_e$ rules or using only *drip* rules. Therefore, the theorem holds. □

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 $\Pi$ and an arbitrary configuration $C$ of $\Pi$, 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 $\Pi$ (in case that $C$ is reachable in $\Pi$).

Clearly, one can generate in a systematic fashion all reachable configurations of $\Pi$ 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 $\Pi$.

Thus, we have the following result:

**Theorem 15.** *It is decidable whether or not, for any $P_{pp}$ system $\Pi$ and any configuration $C$ of $\Pi$, $C$ is a reachable configuration of $\Pi$.*

It is perhaps worthwhile to discuss Theorem 15 in the light of the universality result stated in Theorem 11. The reason that Theorem 15 holds is that, for a given configuration $C$, one can, a priori, provide an upper bound $m_c$ such that $C$ is reachable in $\Pi$ 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 $\Pi$, then, in general, there is no 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 $\Pi$ is *bounded* if there exists an integer $k$, such that, any reachable configuration of $\Pi$ has less than $k$ membranes.

**Theorem 16.** *It is decidable whether or not an arbitrary $P_{pp}$ system $\Pi$ from $PP_*(\alpha)$, $\alpha \in \{pino_i, pino_e, drip\}$, is bounded.*

**Proof.** Given a $P_{pp}$ system $\Pi$ from $PP_*(\alpha)$ $\alpha \in \{pino_i, pino_e, drip\}$, one can construct a programmed grammar $G$ without appearance checking such that $L(G)$ consists of strings corresponding to all the reachable markings of $\Pi$. Such a grammar $G$ can be constructed in the way described in the proof of Theorem 14.

Since it is decidable whether or not the language of an arbitrary programmed grammar without appearance checking is finite (see, e.g., [7]), we can decide whether or not $L(G)$ is finite. If $L(G)$ is infinite, then, obviously, $\Pi$ is not bounded. Assume

**Table 1**
Computational power for $P_{pp}$ systems using $pino_i$ and protein movement rules ($prot$)

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

The same table holds also for $pino_e$ and $drip$ operations.

now that $L(G)$ is finite. Note that still $\Pi$ can be unbounded because, e.g., many membranes can have the same marking in a certain configuration. It is easy to see that $L(G)$ can be effectively constructed from $G$: by iteration we can find a $k$ such that $L(G) \cap V^k V^* = \emptyset$, where $V$ is the alphabet of $G$, and we now need to check the membership of $w$ in $L(G)$ (which is decidable) for only a finite number of $w \in V^{k-1}$.

By analyzing $L(G)$ it is possible to decide if a pino (or a drip) rule can be applied an unbounded number of times. This is done by constructing a graph having nodes labeled by the strings of $L(G)$. We add directed edges between the nodes in the following way. If a node $x$ is labeled by $w_1$ and a node $y$ is labeled by $w_2$, then there is an edge between the two nodes, directed from $w_1$ to $w_2$, if and only if there is a pino (or a drip) rule in $\Pi$ that applied to the membrane $[\ ]_{w_1}$ can produce two membranes where at least one of them is marked by $w_2$.

Clearly, if the so-constructed graph has a loop, then $\Pi$ is unbounded; otherwise $\Pi$ 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 13), and for $P_{pp}$ systems from $PP_*(\alpha)$, $\alpha \in \{pino_i, pino_e, drip\}$, i.e., using only protein-membrane rules (see Theorem 14), while it is undecidable for arbitrary $P_{pp}$ systems (Theorem 12).

A more general research line is to consider a more realistic model of the way that proteins are embedded in membranes. A possible starting point is to accommodate within our model the concept of the parametric regular spherical membrane presented in [1].

An interesting research topic is to consider protein rules with different execution times (following, for instance, the idea of timed P systems introduced in [5]). Also, it would be interesting to consider proteins marking only one of the two sides of a membrane, similar as defined in [6]. In this way, these proteins would only have an effect on one side of the membrane.

## 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] D. Besozzi, G. Rozenberg, Formalizing spherical membrane structures and membrane proteins populations, in: H.J. Hoogeboom, G. Paun, G. Rozenberg, A. Salomaa (Eds.), Workshop on Membrane Computing, in: Lecture Notes in Computer Science, vol. 4361, Springer, 2006, pp. 18–41.

[2] N. Busi, R. Gorrieri, On the computational power of brane calculi, in: C. Priami, G. Plotkin (Eds.), Transactions on Computational Systems Biology VI, in: Lecture Notes in Computer Science, vol. 4220, Springer, 2006, pp. 16–43.

[3] L. Cardelli, Brane calculi - interactions of biological membranes, in: V. Danos, V. Schachter (Eds.), Computational Methods in System Biology, CSMB2004, Paris, France, May 2004, in: Lecture Notes in Computer Science, vol. 3082, Springer, 2005, pp. 257–280. Revised Papers.

[4] L. Cardelli, Gh. Păun, An universality result for a (mem)brane calculus based on mate/drip operations, International Journal of Foundations of Computer Science 17 (1) (2005) 49–68.

[5] M. Cavaliere, D. Sburlan, Time-independent P systems, in: G. Mauri, Gheorghe Paun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.), Workshop on Membrane Computing, in: Lecture Notes in Computer Science, vol. 3365, Springer, 2004, pp. 239–258.

[6] V. Danos, S. Pradalier, Projective brane calculus, in: V. Danos, V. Schächter (Eds.), CMSB, in: Lecture Notes in Computer Science, vol. 3082, Springer, 2004, pp. 134–148.

[7] J. Dassow, Gh. Păun, Regulated Rewriting in Formal Language Theory, Springer, Berlin, 1984.

[8] J. Hopcroft, J. Ulmann, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.

[9] Gh. Păun, Computing with membranes, Journal of Computer and System Sciences 61 (1) (2000) 108–143. Also, Turku Center for Computer Science-TUCS Report No. 208, 1998.

[10] Gh. Păun, Membrane Computing. An Introduction, Springer, Berlin, 2002.

[11] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 1–3, Springer, 1997.

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