Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 171 (2007) 37–53
www.elsevier.com/locate/entcs

Membrane Systems with Peripheral Proteins: Transport and Evolution

Matteo Cavaliere, Sean Sedwards[1,2]

The Microsoft Research-University of Trento
Centre for Computational and Systems Biology
Trento, Italy

Abstract

Transport of substances and communication between compartments are fundamental biological processes, often mediated by the presence of complementary proteins attached to the surfaces of membranes. Within compartments, substances are acted upon by local biochemical rules. Inspired by this behaviour we present a model based on membrane systems, with objects attached to the sides of the membranes and floating objects that can move between the regions of the system. Moreover, in each region there are evolution rules that *rewrite* the transported objects, mimicking chemical reactions. We first analyse the system, showing that interesting *qualitative* properties can be decided (like reachability of configurations) and then present a simulator based on a stochastic version of the introduced model and show how it can be used to simulate relevant *quantitative* biological processes.

Keywords: membrane systems, simulator, stochastic, systems biology.

1 Introduction and Motivations

Membrane systems are models of computation inspired by the structure and the function of biological cells. The model was introduced in 1998 by Gh. Păun and since then many results have been obtained, mostly concerning the computational power of the model (for an updated bibliography the reader can consult the webpage [23]). More recently, membrane systems have been applied to systems biology and several models have been proposed for simulating biological processes (e.g., see the monograph dedicated to membrane systems applications, [8]).

In the original definition, membrane systems are composed of an hierarchical nesting of membranes that enclose regions in which floating objects exist. Each region can have associated rules for evolving these objects (called evolution rules, modelling the biochemical reactions present in cell regions), and/or rules for moving

[1] Email: matteoDOTcavaliereATmsr-unitn.unitn.it
[2] Email: seanDOTsedwardsATmsr-unitn.unitn.it

doi:10.1016/j.entcs.2007.05.006

objects across membranes (called symport/antiport rules, modelling some kinds of transport rules present in cells). Recently, inspired by *brane calculus*, [4], a model of a membrane system, having objects attached to the membranes, has been introduced in [5]. Other models bridging brane calculus and membrane systems have been proposed in [14,17]. A more general approach, considering both free floating objects and objects attached to the membranes has been proposed and investigated in [3]. The idea of these models is that membrane operations are moderated by the objects (proteins) attached to the membranes. However, in these models objects were associated to an atomic membrane which has no concept of inner or outer surface. In reality, many biological processes are driven and controlled by the presence, on the appropriate side of a membrane, of specific proteins. For instance, receptor-mediated endocytosis, exocytosis and budding in eukaryotic cells are processes where the presence of proteins on the internal and external surfaces of a membrane is crucial (see e.g., [1]).

These processes are, for instance, used by eukaryotic cells to take up macromolecules and deliver them to digestive enzymes stored in lysosomes inside the cells. In general, all the compartments of a cell are in constant communication, with molecules being passed from a donor compartment to a target compartment by means of numerous membrane-enclosed transport packages, or *transport vesicles*. Once transported to the correct compartment the substances are then *processed* by means of local biochemical reactions (see e.g., [1]).

Motivated by this, we introduce a model combining some basic features found in biological cells: (*i*) *evolution* of objects (molecules) by means of multiset rewriting rules associated with specific regions of the systems (the rules model biochemical reactions); (*ii*) *transport* of objects across the regions of the system by means of rules associated with the membranes of the system and involving proteins attached to the membranes (on one or possibly both the two sides) and (*iii*) rules that take care of the *attachment/de-attachment* of objects to/from the sides of the membranes. Moreover, since we want to distinguish the functioning of different regions, we also associate to each membrane a unique identifier (a label).

In this paper we present a preliminary qualitative investigation of the model when the evolution is based on a sort of *free parallelism*: we prove that in this case several interesting problems, like configuration reachability, can be decided. We also introduce a stochastic variant of the model (i.e., where each rule has an associated stochastic *rate*) that underlies an implemented simulator which we have used to model interesting biological cellular processes.

We wish to comment that the model presented follows the philosophy of the evolution-communication model introduced in [6], where the system evolves by evolution of the objects and transport of objects by means of symport/antiport rules, that are essentially synchronized exchanges of objects. However, in our case the transport of objects may depend on the presence of particular *proteins* attached to the internal and external surfaces of the membranes. Therefore this paper can be seen as a bridge between membrane systems and *projective brane calculus*, [9], where, in the framework of process algebra, directed actions associated to membranes have

been considered.

2 Formal Language Preliminaries

We will briefly recall the main notions and results of the formal language theory used in this paper. For more details the reader can consult standard books, such as [12], [22], [10], and the respective chapters of the handbook [21].

Given a set A, we denote by $|A|$ its cardinality. 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 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 $|w|_a$. The notation $Perm(x)$ indicates the set of all strings that can be obtained as a permutation of the string x.

For $x, y \in V^*$ we define their *shuffle* by $x\xi y = \{x_1y_1 \cdots x_ny_n \mid x = x_1 \cdots x_n, y = y_1 \cdots y_n, x_i, y_i \in V^*, 1 \leq i \leq n, n \geq 1\}$. The operation can be extended in a natural way to languages. Then, given L_1 and L_2, we have $L_1\xi L_2 = \bigcup_{x_1 \in L_1, x_2 \in L_2} x_1\xi x_2$.

Denoting by REG the family of regular languages, the following result holds (see e.g., [21]) (proved in a constructive way).

Theorem 2.1 *$L_1, L_2 \in REG$, then $L_1\xi L_2 \in REG$*

A *multiset* over a set V is a map $M : V \to \mathbb{N}$, where $M(a)$ denotes the multiplicity of the symbol $a \in V$ in the multiset M. This fact can also be indicated in the forms $(a, M(a))$ or $a^{M(a)}$, for all $a \in V$. If the set V is finite, e.g. $V = \{a_1, \ldots, a_n\}$, then the multiset M can be explicitly described as $\{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_n, M(a_n))\}$. The *support* of a multiset M is the set $supp(M) = \{a \in V \mid M(a) > 0\}$. A multiset is empty (so finite) when its support is empty (also finite).

A compact notation can be used for finite multisets: if $M = \{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_n, M(a_n))\}$ is a multiset of finite support, then the string $w = a_1^{M(a_1)}a_2^{M(a_2)} \ldots a_n^{M(a_n)}$ (and all its possible permutations) precisely identify the symbols in M and their multiplicities. Hence, given a string $w \in V^*$, we can assume that it identifies a finite multiset over V defined by $M(w) = \{(a, |w|_a) \mid a \in V\}$.

In this paper we make use of the notion of a *matrix grammar*.

A *matrix grammar with appearance checking (ac)* is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets of non-terminal and terminal symbols, $S \in N$ is the axiom, M is a finite set of matrices, which are sequences of context-free rules of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M.

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either

(i) $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$

or

(*ii*) $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F.

The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied (one says that these rules are applied in *appearance checking* mode).

The family of languages generated by matrix grammars with appearance checking is denoted by MAT_{ac}.

G is called a *matrix grammar without appearance checking* if and only if $F = \emptyset$. In this case, the generated family of languages is denoted by MAT.

If we denote by CF, and RE the family of context-free and recursively enumerable languages, respectively, then the following results hold:

Theorem 2.2

- $CF \subset MAT \subset RE$
- $MAT \subset MAT_{ac} = RE$

A matrix grammar is called *pure* if there is no distinction between terminals and non-terminals. The language generated by a pure matrix grammar is composed of all the sentential forms. The family of languages generated by pure matrix grammars without appearance checking is denoted by $pMAT$. A proof of this can be found, for example, in [10].

Theorem 2.3 $pMAT \subset MAT$

In what follows we assume the reader to be familiar with the basic notions of membrane systems, for instance, as presented in the introductory guide [20].

3 Membrane Operations with Peripheral Proteins

As is usual in the membrane systems field, a membrane is represented by a pair of square brackets, [ ]. To each topological *side* of a membrane we associate *multisets* u and v (over a particular alphabet V) and this is denoted by $[\,_u]_v$. We say that the membrane is *marked* by u and v; v is called the *external marking* and u the *internal marking*; in general, we refer to them as *markings* of the membrane. The objects of the alphabet V are called *proteins* or, simply, *objects*. An objects is called *free* if it is not attached to the sides of a membrane, so is not part of a marking.

Each membrane encloses a *region* and the *contents* of a region can consist of *free objects* and/or other membranes (we also say that the region *contains* free objects and/or other membranes).

Moreover, each membrane has an associated label that is written as a superscript of the membrane. If a membrane is named by the label i we call it membrane i. Each membrane encloses a unique region, so we also say region i to identify the region enclosed by membrane i. The *set of all labels* is denoted by *Lab*.

For instance, in the system $[\ abb\ [\ aaaa\ _{ab}]^1_b\ _{bba}]^2_{ab}$, the external membrane, labelled by 2, is marked by *bba* (internal marking) and by *ab* (external marking). The contents of the region enclosed by the external membrane is composed of the

free objects a, b, b and the membrane $[\ aaaa\ _{ab}]^1_b$.

We consider rules that model the attachment of objects to the sides of the membranes. These rules extend the definition given in [3].

$$attach:\ [\ a\ _u]^i_v \to [\ _{ua}]^i_v, \quad a[\ _u]^i_v \to [\ _u]^i_{va}$$
$$de-attach:\ [\ _{ua}]^i_v \to [a\ _u]^i_v, \quad [\ _u]^i_{va} \to [\ _u]^i_v a$$

with $a \in V$, $u, v \in V^*$ and $i \in Lab$.

The semantics of the attachment rules (attach) is as follows. For the first case, the rule is *applicable* to the membrane i if the membrane is marked by multisets *containing* the multisets u and v, on the appropriate sides, and region i contains the object a. In the second case, the rule is applicable to membrane i if it is marked by multisets containing the multisets u and v, as before, and is contained in a region that contains the object a.

When either rule is executed, the object a is added to the appropriate marking in the way specified. The objects not involved in the application of a rule are left unchanged in their original positions.

The semantics of the de-attachment rules is similar, with the difference that the attached object a is detached from the specified marking and added to the contents of either the internal or external region.

We now consider rules associated to the membranes that control the passage of objects across the membranes.

$$move_{in}:\ a[\ _u]^i_v \to [\ a\ _u]^i_v$$
$$move_{out}:\ [\ a\ _u]^i_v \to a[\ _u]^i_v$$

with $a \in V$, $u, v \in V^*$ and $i \in Lab$.

The semantics of the rules is as follows. In the first case, the rule is applicable to membrane i if it is marked by multisets containing the multisets u and v, on the appropriate sides, and the membrane is contained in a region containing the object a. When the rule is executed the object a is removed from the contents of the region surrounding membrane i and added to the contents of region i.

In the second case the semantics is similar, but here the object a is moved from region i to its surrounding region.

The rules of attach, de-attach, move$_{in}$, move$_{out}$ are generally called *membrane rules* over the alphabet V and the set of labels Lab.

We also introduce *evolution rules* that involve objects but not membranes. These can be considered to model the biochemical reactions that take place inside the compartments of the cell. They are *evolution rules* over the alphabet V and set of labels Lab and they follow the definition that can be found in evolution-communication P systems [6].

$$evol:\ [u \to v]^i$$

with $u \in V^+, v \in V^*$ and $i \in Lab$. The evolution rule is called *cooperative* (*coo*) if $|u| > 1$, otherwise the rule is called *non-cooperative* (*ncoo*).

The semantics of the rule is as follows. The rule is applied to region i if the region *contains* a multiset of free objects that *includes* the multiset u. When the rule is executed the objects specified by u are subtracted from the contents of region i and the objects specified by v are added to the contents of the region i.

4 Membrane Systems with Peripheral Proteins

In this section we define membrane systems having membranes marked with multisets of proteins on both sides of the membrane, free objects and using the operations introduced in Section 3.

Formally, a *membrane system with peripheral proteins* (in short, a P_{pp} system) and n membranes, is a construct

$$\Pi = (V, \mu, (u_1, v_1), \ldots, (u_n, v_n), w_1, \ldots, w_n, R, R^m)$$

- V is a finite, non-empty alphabet of objects (proteins).
- μ is a membrane structure with $n \geq 1$ membranes, injectively labelled by $1, 2, \ldots$ $\ldots, n$.
- $(u_1, v_1), \cdots, (u_n, v_n) \in V^* \times V^*$ are the markings associated, at the beginning of any evolution, to the membranes $1, 2, \cdots, n$, respectively. They are called *initial markings* of Π; the first element of each pair specifies the internal marking, while the second one specifies the external marking.
- $w_1, \cdots, w_n$ specify the multisets of free objects contained in regions $1, 2, \cdots, n$, respectively, at the beginning of any evolution and they are called *initial contents* of the regions.
- R is a finite set of evolution rules over V and the set of labels $Lab = \{1, \ldots, n\}$.
- R^m is finite set of membrane rules over the alphabet V and set of labels $Lab = \{1, \ldots, n\}$.

A *configuration* of Π consists of a membrane structure, the markings (internal and external) of the membranes and the multisets of free objects present inside the regions. In what follows, configurations are denoted by writing the markings as subscripts (internal and external) of the parentheses which identify the membranes, the labels of the membranes are written as superscripts and the contents of the regions as string, e.g.,

$$[\,[\;{}_{aa}]^4_{ab}\;[aaa\;{}_{aa}]^2_b\;[\;b\;]^3_{bb}\;a\;]^1_a$$

We suppose a standard labelling: 0 is the label of the *environment* that surrounds the entire system Π; 1 is the label of the *skin* membrane that separates Π from the *environment*.

The *initial configuration* consists of the membrane structure μ, the initial markings of the membranes and the initial contents of the regions; the environment is

empty at the beginning of the evolution.

We assume the existence of a 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 and to each region *an arbitrary number of membrane and evolution rules*. This implies that, in one step, no rule, one rule, or as many as rules as desired may be applied in a non-deterministic way (i.e., so-called *free parallelism*) and all rules have equal precedence. A single occurrence of an object (free or not) can be involved in only a single application of a rule.

A sequence of transitions, starting from the initial configuration, is called an *evolution*. An evolution is said to be *halting* if it halts, that is, if it reaches a *halting configuration*, i.e., a configuration where no rule can be applied anywhere in the system.

A configuration of a P_{pp} system Π that can be reached by a sequence of transitions, starting from the initial configuration, is called *reachable*. A pair of multisets (u, v) is a *reachable marking* for Π if there exists a reachable configuration of Π which contains at least one membrane marked internally by u and externally by v. We denote by $\mathbb{C}(\Pi)$ the set of all possible configurations of Π, by $\mathbb{C}_R(\Pi)$ the set of all reachable configurations of Π, and by $\mathbb{M}_R(\Pi)$ the set of all reachable markings of Π.

Moreover we denote by $\mathbb{P}_{pp,m}(mem_{rul}, \alpha)$, $\alpha \in \{coo, ncoo\}$ the class of membrane systems with peripheral proteins, membrane rules, evolution rules of type α and m membranes (m is changed to $*$ if it is unbounded). We omit mem_{rul} or α from the notation if the corresponding type of rules is not allowed. We also denote by V_Π the alphabet V of the system Π. The notion of free parallelism we use here is similar to the one introduced in ([19], Chapter 3.4).

5 Reachability of Configurations and Markings

A natural question with possible biological implications concerns whether or not a system can evolve to a particular specified configuration. Hence it would be useful to construct models having such qualitative properties, to be decidable.

In our case, we can prove that it is possible to decide, for an arbitrary membrane system with peripheral proteins and an arbitrary configuration, whether or not such a configuration is reachable. A proof can be demonstrated by showing that all the reachable configurations of a system Π can be produced by a pure matrix grammar without appearance checking. Moreover, we also prove that the reachability of an arbitrary marking can be decided.

Lemma 5.1 *It is decidable whether or not, for any P system Π from $\mathbb{P}_{pp,1}(coo)$ and any configuration C of Π, $C \in \mathbb{C}_R(\Pi)$*

Proof. Let $\Pi = (V, \mu = [\,]^1, (u_1, v_1), w_1, R)$. We first notice that each configuration C of Π is essentially the contents of the unique region and therefore, being a multiset, it can be represented by a string w_C, as described in Section 2 (every permutation

of the string w_C represents the same contents, so the same configuration C). We construct a pure matrix grammar G without appearance checking such that $L(G)$ contains all and only the strings representing the configurations in $\mathbb{C}_R(\Pi)$.

The grammar $G = (N, S, M)$ is defined in the following way. $N = V \cup V^{\#}$, with $V^{\#} = \{v^{\#} \mid v \in V\}$. We add to M the following matrices. $(S \to w_1)$ and, for each rule $[x \to y]^1 \in R$, the matrix

$$(x_1 \to x_1^{\#}, x_2 \to x_2^{\#}, \cdots, x_k \to x_k^{\#}, x_1^{\#} \to \lambda, x_2^{\#} \to \lambda, \cdots, x_k^{\#} \to y_1 y_2 \cdots y_q)$$

where $x = x_1 x_2 \cdots x_k$ and $y = y_1 y_2 \cdots y_q$. Each application of a matrix simulates the application of an evolution rule inside the unique region of the system. The markings are not involved in the evolution of the system since membrane rules are not allowed. It is immediate that, for each string w in $L(G)$ (i.e., all the sentential forms generated by G) there is an evolution of Π, starting from the initial configuration, that reaches the configuration represented by w. Moreover, it is easy to see that the reverse is also true since the evolution of Π is based on free parallelism: for each reachable configuration C' of Π there exists a derivation of G that generates a string representing C'. In fact it is immediate to see that $L(G)$ contains all the strings representing configurations of Π reached by applying at each step a single evolution rule. In case a configuration C' is reached by applying more than an unique evolution rule in a single step, then such a single step can be simulated in G by applying an appropriate sequence of matrices (because the evolution of Π is based on free parallelism).

Therefore to check whether or not an arbitrary configuration C of Π can be reached, we only need to check if any of the strings representing C is in $L(G)$. This can be done since there is only a finite number of strings representing C and the membership problem for pure matrix grammars without appearance checking is decidable (see, e.g., [10]); therefore the Lemma follows. □

Theorem 5.2 *It is decidable whether or not, for any P system* Π *from* $\mathbb{P}_{pp,*}(mem_{rul}, coo)$ *and any configuration* C *of* Π, $C \in \mathbb{C}_R(\Pi)$

Proof. The main idea of the proof is that the problem can be reduced to check whether or not a configuration of a system from $\mathbb{P}_{pp,1}(coo)$ is reachable, and this is decidable (Lemma 5.1).

Suppose $\Pi = (V, \mu, (u_1, v_1), \ldots, (u_n, v_n), w_1, \ldots, w_n, R, R^m)$. By $cont(i)$ we denote the label of the region surrounding membrane i (we recall that 0 is the label of the environment and 1 is the label of the skin membrane).

We construct $\overline{\Pi} = (\overline{V}, [\,]^1, (\lambda, \lambda), \overline{w_1}, \overline{R})$ from $\mathbb{P}_{pp,1}(coo)$ in the following way.

We define $\overline{V} = \bigcup_{i \in \{1,\cdots,n\}} (V_i' \cup V_i'') \cup \bigcup_{i \in \{0,1,\cdots,n\}} V_i$ with $V_i = \{a_i \mid a \in V\}$, $V_i' = \{a_i' \mid a \in V\}$, $V_i'' = \{a_i'' \mid a \in V\}$.

We use the morphisms h_i, h_i', h_i'', defined as follows.

- $h_i : V \to V_i$ defined by $h_i(a) = a_i, a \in V$, for $i \in \{0, 1, \cdots, n\}$
- $h_i' : V \to V_i'$ defined by $h_i'(a) = a_i', a \in V$, for $i \in \{1, \cdots, n\}$
- $h_i'' : V \to V_i''$ defined by $h_i''(a) = a_i'', a \in V$, for $i \in \{1, \cdots, n\}$

We define $\overline{w_1}$ as the string $h_1(w_1)\cdots h_n(w_n)h'_1(u_1)\cdots h'_n(u_n)h''_1(v_1)\cdots h''_n(v_n)$.

For each rule $move_{in}$, $a[\ \ _u]^i_v \to [\ a\ _u]^i_v \in R^m$, $i \in \{1,\cdots,n\}$ we add to $\overline{R}$ the following rules: $[\ a_k h'_i(u)h''_i(v) \to a_i h'_i(u)h''_i(v)]^1$, with $k = cont(i)$.

In the same way all the other rules present in $R \cup R^m$ can be *translated* in the evolution rules for $\overline{R}$.

Hence, given a configuration C of Π, one can construct the configuration $\overline{C}$ of $\overline{\Pi}$ having a unique region in the following way.

For each occurrence of free object a contained in region i (the environment if $i = 0$) in C, $i \in \{0,1,\cdots,n\}$ we add the object $h_i(a)$ in region 1 of $\overline{C}$. For each occurrence of object a present in the internal marking of membrane i in C, $i \in \{1,\cdots,n\}$ we add the object $h'_i(a)$ to region 1 of $\overline{C}$ and finally for each occurrence of object a present in the external marking of membrane i, $i \in \{1,\cdots,n\}$ we add the object $h''_i(a)$ to region 1 of $\overline{C}$.

Now we can decide (Lemma 5.1) whether or not $\overline{C} \in \mathbb{C}_R(\overline{\Pi})$.

From the way $\overline{\Pi}$ has been constructed it follows that:

- if $\overline{C} \in \mathbb{C}_R(\overline{\Pi})$ then $C \in \mathbb{C}_R(\Pi)$
- if $\overline{C} \notin \mathbb{C}_R(\overline{\Pi})$ then $C \notin \mathbb{C}_R(\Pi)$

and from this the Theorem follows.

□

Corollary 5.3 *It is decidable whether or not, for any P system* Π *from* $\mathbb{P}_{pp,n}(mem_{rul}, coo), n \geq 1$ *and any pair of multisets* (u,v) *over* V_Π, $(u,v) \in \mathbb{M}_R(\Pi)$.

Proof. Given Π from $\mathbb{P}_{pp,n}(mem_{rul}, coo)$ and with alphabet of objects V, one can construct $\overline{\Pi} = (\overline{V}, \mu = [\ ]^1, (\lambda,\lambda), \overline{w_1}, \overline{R})$ from $\mathbb{P}_{pp,1}(coo)$ in the way described by Theorem 5.2.

Therefore, using $\overline{\Pi}$ one can construct the grammar G as described by Lemma 5.1 such that $L(G)$ contains all and only the strings representing the configurations in $\mathbb{C}_R(\overline{\Pi})$.

Now to check whether or not an arbitrary $(u,v) \in \mathbb{M}_R(\Pi)$ one needs to check whether or not there exists an $i \in \{1,\cdots,n\}$ such that $(Perm(h'_i(u))\xi(\overline{V})^*) \cap L(G) \neq \emptyset$ *and* $(Perm(h''_i(v))\xi(\overline{V})^*) \cap L(G) \neq \emptyset$, where h'_i and h''_i are morphisms from V to V'_i and to V''_i, respectively, defined as in Theorem 5.2, and ξ denotes the shuffle operation.

The permutation and shuffle operation are used to construct all possible strings representing a configuration of $\overline{\Pi}$ containing the membrane i marked by multiset u internally and multiset v externally.

The languages $(Perm(h'_i(u))\xi(\overline{V})^*) \cap L(G)$ and $(Perm(h''_i(v))\xi(\overline{V})^*) \cap L(G)$ can be generated by matrix grammars without appearance checking (see, Theorem 2.1 and e.g., [10]) and the emptiness problem for this class of grammars is decidable (see, e.g., [10]). Therefore the Corollary follows. □

6 Stochastic Simulation of Yeast G-protein Cycle

Having defined a qualitative model, we wish to use it to examine quantitative properties of biological systems using a simulator.

Deterministic simulations are useful to describe reactions between large numbers of chemical objects, however they may not accurately represent the dynamical behaviour of small quantities of reactants. In this latter case a discrete stochastic simulation is more appropriate and, moreover, approximates the deterministic approach when the quantities are increased [11]. Hence we have created a simulator [24] based on the presented model, which assumes discrete molecular interactions and uses the Gillespie algorithm [11] to stochastically choose at each step which single rule to apply (in one of the regions) and to calculate its stochastic time delay. Thus the more general free parallel theoretical model is here reduced to a specific sequential one.

To demonstrate the simulator we model the G-protein mating response in yeast *saccharomyces cerevisiae*, based on experimental rates provided by [13]. The G-protein transduction pathway involves membrane proteins and the transport of substances between regions and is a mechanism by which organisms detect and respond to environmental signals. It is extensively studied and many pharmaceutical agents are aimed at components of the G-protein cycle in humans. Figure 1 shows the relationships between the various reactants and regions modelled in the simulation, Figure 2 is the simulation script and Figure 3 shows the results of the simulation.

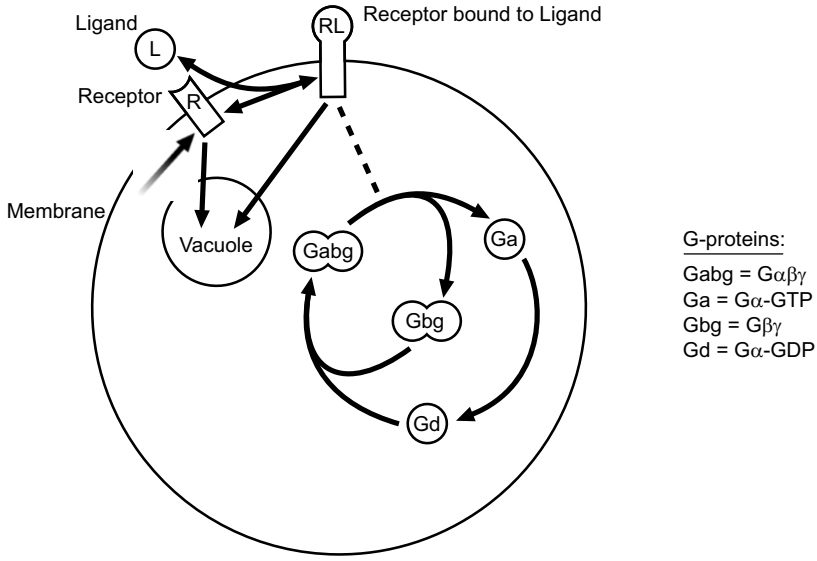

Fig. 1. Model of saccharomyces cerevisiae mating response.

A brief description of the process is that the yeast cell receives a pheromone signal (L) which binds to receptor R, integral to the cell membrane. The receptor-ligand dimer then catalyses the reaction that converts the inactive G-protein Gabg

to the active Ga. A competing sequence of reactions converts Ga to Gabg via Gd in combination with Gbg. The bound and unbound receptor (RL and R, respectively) are degraded by transport into a vacuole via the cytoplasm.

```
// Saccharomyces cerevisiae G-protein mating response
molecule L,R,RL,Gd,Gbg,Gabg,Ga

rule g_cycle {
    || 4-> |R|
    |R| + L 3.32e-18-> |RL|
    |RL| 0.011-> |R| + L
    |RL| 4.1e-3-> RL + ||
    |R| 4.1e-4-> R + ||
    Gabg + |RL| 1.0e-5-> Ga, Gbg + |RL|
    Gd + Gbg 1-> Gabg
    Ga 0.11-> Gd
}
rule vac_rule {
    || + R 4.1e-4-> R + ||
    || + RL 4.1e-3-> RL + ||
}
compartment vacuole [vac_rule]
compartment cell [vacuole, 3000 Gd, 3000 Gbg, 7000 Gabg, g_cycle :  |10000 R|]
system cell, 6.022e17 L
evolve 0-600000

plot cell[Gd,Gbg,Gabg,Ga:|R,RL|]
```

Fig. 2. Simulation script of G-protein cycle using data from [13].

7 Perspectives

We have introduced a model of membrane systems with objects attached to both sides of the membranes. In addition, the model is equipped with operations that can rewrite floating objects and move objects between regions depending on the attached objects. We have proved that when the system works in a free parallelism mode (i.e., allowing an arbitrary number of rules to be applied at each step) many useful properties can be decided (for instance, reachability of a configuration or of a certain protein marking).

In the second part of the paper we have presented a simulator that implements a stochastic variant of the introduced model. The simulator has an intuitive syntax and can be used to model biological processes where the transport of objects across membranes is coupled with the processing/decay of substances within the regions. As an example we have presented the simulation of saccharomyces cerevisiae heterotrimeric G-protein cycle.

Several different research directions may now be pursued. The model may be further developed, for example, to include evolution based on *maximal parallel* semantics, as commonly used in P systems. In that case it is most likely that many properties would not be decidable; an interesting problem is then to find (sub)classes (using restricted evolution and/or transport rules, say) where interesting properties are still decidable. Additionally, other bio-inspired operations may be introduced, such as *fission and fusion* of regions, all still dependent on the objects attached to the membranes, along the lines of the research found in [17].

Another direction of research is the application of the existing model. The implemented stochastic software can be used to simulate interesting biological pro-

cesses where the rôle of surface proteins and transport of substances is crucial (as in drug-resistance, see e.g.,[16]).

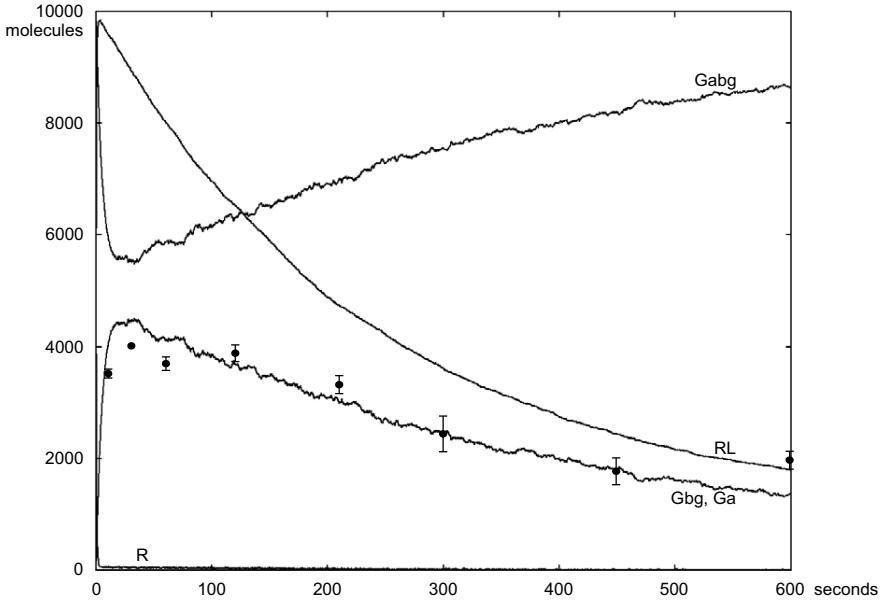

Fig. 3. Simulation results (continuous curves) and experimental data (points with error bars, [13]) corresponding to simulated Ga. Note that Gd decays rapidly and is not visible at this scale.

References

[1] B. Alberts, *Essential Cell Biology. An Introduction to the Molecular Biology of the Cell*. Garland Publ. Inc., New York, London, 1998.

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

[3] R. Brijder, M. Cavaliere, A. Riscos-Núñez, G. Rozenberg, D. Sburlan, Membrane Systems with Marked Membranes. Submitted.

[4] 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.

[5] 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. Also at `http://www.gcn.us.es/`.

[6] M. Cavaliere, Evolution-Communication P Systems. *Proceedings International Workshop Membrane Computing*, (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron eds.), Lecture Notes in Computer Science, 2597, Springer-Verlag, Berlin, 2003, pp. 134–145.

[7] 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.

[8] G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds., *Applications of Membrane Computing*. Springer-Verlag, Berlin, 2006.

[9] 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.

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

[11] D. T. Gillespie, Exact Stochastic Simulation of Coupled Chemical Reactions, *Journal of Physical Chemistry*, 81, 25, 1977.

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

[13] T.-M. Yi, H. Kitano, M. I. Simon, A quantitative characterization of the yeast heterotrimeric G protein cycle, *Proceedings of the National Academy of Science*, 100, 19, 2003.

[14] S.N. Krishna, Universality Results for P Systems based on Brane Calculi Operations. *Theoretical Computer Science*, to appear.

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

[16] K. H. Lundstrom, *G Protein Coupled Receptors in Drug Discovery*, Taylor & Francis, 2005.

[17] A. Păun, B. Popa, P Systems with Proteins on Membranes and Membrane Division. *Proceedings Tenth International Conference in Developments in Language Theory, DLT06*, Lecture Notes in Computer Science, Springer-Verlag, to appear.

[18] 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.

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

[20] Gh. Păun, G. Rozenberg, A Guide to Membrane Computing. *Theoretical Computer Science*, 287-1 (2002), pp. 73–100.

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

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

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

[24] http://www.msr-unitn.unitn.it/downloads.php

A The Simulator Syntax

The simulator syntax aims to be an intuitive interpretation of the P_{pi} system model. A simulator script conforms to the following grammar:

$$\begin{aligned} SimulatorScript = \quad & \{ObjectDeclaration \;\; NewLine\}^{+} \\ & \{RuleDefinition \;\; NewLine\}^{+} \\ & \{CompartmentDefinition \;\; NewLine\} \\ & SystemStatement \;\; NewLine \\ & EvolveStatement \;\; NewLine \\ & PlotStatement \;\; [NewLine] \end{aligned}$$

where $NewLine$ is an appropriate sequence of characters to generate a new line.

An example of a simple simulator script is shown below, together with its P_{pi} system counterpart.

Simulator script	P_{pi} system *lotka*
`// Lotka reactions`	
`object X,Y1,Y2,Z`	$V_{lotka} = \{X, Y1, Y2, Z\}$
	$rate_{lotka} = \{$
`rule r1 X + Y1 0.0002-> 2Y1 + X`	$[\, XY1 \to Y1Y1X \,]^0_{\vert\,\vert} \mapsto 0.0002$
`rule r2 Y1 + Y2 0.01-> 2Y2`	$[\, Y1Y2 \to Y2Y2 \,]^0_{\vert\,\vert} \mapsto 0.01$
`rule r3 Y2 10-> Z`	$[\, Y2 \to Z \,]^0_{\vert\,\vert} \mapsto 10\ \}$
`system 100000 X,1000 Y1,1000 Y2,r1,r2,r3`	$w_{0,lotka} = X^{100000}Y1^{1000}Y2^{1000}$
	$\mu_{lotka} = [\,]^0$
`evolve 0-1000000`	$t_{in,lotka} = 0$
`plot Y1,Y2`	

The syntax of the sections of a simulator script are described below.

A.1 Comments

Comments begin with a double forward slash (`//`) and include all subsequent text on a single line. They may appear anywhere in the script.

A.2 Object Declaration

The reacting objects are defined in one or more statements beginning with the keyword `object` followed by a comma separated list of unique reactant names. E.g.:

```
object X,Y1,Y2,Z
```

The names are case-sensitive and must start with a letter but may include digits and the underscore character (_). This corresponds to defining the alphabet V of the P_{pi} system.

A.3 Rule Definition

The reaction rules are defined using rule definitions comprising the keyword `rule` followed by a unique name and the rewriting rule itself. E.g.:

```
rule r1 X + Y1 0.0002-> 2Y1 + X
```

These correspond to the attach / de-attach and evolution rules of the P_{pi} system model. Note, however, that simulator rules are user-defined types which may be instantiated in more than one region. The value preceding the implication symbol (`->`) is the average reaction rate and corresponds to an element of the range of the

mapping *rate* given in Definition 1. In the simulator it is also possible to define a reaction rate as the product of a constant and the rate of a previously defined rule, using the name of the previous rule in the following way:

```
rule r2 Y1 + Y2 50 r1-> 2Y2
```

This has the meaning that rule `r2` has a rate 50 times that of `r1`. In addition, in the simulator it is possible to define a group of rules using a single identifier and braces. E.g.,

```
rule lotka {
     X + Y1 0.0002-> 2Y1 + X
     Y1 + Y2 0.01-> 2Y2
     Y2 10-> Z  }
```

To include membrane operations the simulator rule syntax is extended with the `||` symbol. Objects listed on the left hand side of the `||` represent the internal markings, objects listed on the right hand side represent the external markings and objects listed between the vertical bars are the integral markings of the membrane. E.g.:

```
rule r4 X + |Y2| 0.1-> |X,Y2|
```

means that if one `X` exists within the compartment and one `Y2` exists integral to the membrane, then the `X` will be added to the integral marking of the membrane. The P_{pi} system equivalent is the following $attach_{in}$ rule:

$$[\ X\ ]_{|Y2|} \rightarrow [\ ]_{|XY2|}$$

To represent an $attach_{out}$ rule in the simulator the following syntax is used:

```
rule r4 |Y2| + X 0.1-> |X,Y2|
```

Here the `X` appears to the right of the `||` symbol following a `+`, meaning that it must exist in the region surrounding the membrane for the rule to be applied. Hence the `+` used in simulator membrane rules is non-commutative.

A.4 Compartment Definition

Compartments may be defined using the keyword `compartment` followed by a unique name and a list of contents and rules, all enclosed by square brackets. For example,

```
compartment c1 [100 X, 100 Y1, r1, r2]
```

instantiates a compartment having the label `c1` containing 100 `X`, 100 `Y1` and rules `r1` and `r2`. In a P_{pi} system such a compartment would have a P_{pi} system (partial) initial instantaneous description

$$[\ X^{100}Y1^{100}\ ]^1$$

Note that a P_{pi} system requires a numerical membrane label and that any rules associated to the region or membrane must be defined separately.

Compartments may contain other pre-defined compartments, so the following simulator statement

```
compartment c2 [100 Y2, c1]
```

corresponds to the P_{pi} system (partial) initial instantaneous description

$$[\, Y2^{100}[\, X^{100}Y1^{100}\,]^1\,]^2$$

Membrane markings in the simulator are added to compartment definitions using the symbol `||`, to the right of and separated from the floating contents by a colon. E.g.,

```
compartment c3 [100 X, c2 :  10 Y2||10 Y1]
```

has the meaning that the compartment `c3` contains compartment `c2`, 100 `X`, and the membrane surrounding `c3` has 10 `Y2` attached to its inner surface and 10 `Y1` attached to its outer surface. This corresponds to the P_{pi} system (partial) initial instantaneous description

$$[\, X^{100}[\, Y2^{100}[\, X^{100}Y1^{100}\,]^1\,]^2\,]^3_{Y2^{10}|\,|Y1^{10}}$$

A.5 System Statement

The system is instantiated using the keyword `system` followed by a comma-separated list of constituents. E.g.:

```
system 100000 X,1000 Y1,1000 Y2,r1,r2,r3
```

This statement corresponds to the definition of $u_0 \ldots u_n$, $v_0 \ldots v_n$, $w_0 \ldots w_n$, $x_0 \ldots x_n$ and μ of the P_{pi} system.

The system statement may be extended to multiple lines by enclosing the list of constituents between braces. E.g.:

```
system {
   100000 X,
   1000 Y1,
   1000 Y2,
   r1,r2,r3 }
```

It is also possible to add or subtract reactants from the simulation in runtime using the following syntax in the `system` statement:

```
-10 X @50000, 10 Y1 @50000
```

These *instructions* request a subtraction of ten X from the system and an addition of ten Y1 to the system at time step 50000. Negative quantities are not allowed in the simulator, so if a subtraction requests a greater amount than exists, only the existing amount will be deleted.

A.6 Evolve Statement

The simulator requires a directive to specify the total number of evolution steps to perform and also the number of the evolution step at which to start recording data. This is achieved using the keyword `evolve` followed by the minimum and maximum evolution steps to record. E.g.,

```
evolve 0-1000000
```

Note that the minimum evolution step does not correspond to t_{in} of the P_{pi} system, since the simulation always starts from the 0^{th} step. By convention, the simulator sets the initial time of the simulation to 0, hence $t_{in} = 0$ for all simulations. Note that although t_{fin} of a P_{pi} system evolution *corresponds* to the maximum evolution step, the units are different and there is no explicit conversion.

A.7 Plot Statement

To specify which objects are to be observed during the evolution the `plot` keyword is used followed by a list of reactants. To plot the contents of a specific compartment the `plot` statement uses syntax similar to that used in the compartment definition. E.g.,

```
plot X, c3[X,Y1 :  Y1|Y2|]
```

plots the number of free-floating `X` in the environment and the specified contents of compartment `c3` and its membrane.