An Universality Result for a (Mem)Brane Calculus Based on Mate/Drip Operations

Luca Cardelli[1], Gheorghe Păun[2]

[1] Microsoft Research Cambridge
7, J.J. Thomson Avenue, Cambridge, CB3 0FB, UK
E-mail: luca@microsoft.com

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
and
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: george.paun@imar.ro, gpaun@us.es

Summary. Operations with membranes are essential both in brane calculi and in membrane computing. In this paper we take four basic operations from brane calculi, *pino, exo, mate, drip*, we express them in terms of the membrane computing formalism, and then we investigate the computing power of the P systems using the *mate, drip* operations as unique evolution rules. All operations are controlled by – and make evolve – multisets of protein-objects embedded in the membranes themselves (not contained in the compartments of the cell, as standard in membrane computing; all compartments delimited by membranes are here empty). Somewhat surprisingly, for systems which use the *mate, drip* operations we obtain the Turing completeness. The power of P systems based on other operations remains to be investigated.

1 Introduction

Membrane computing [7] and brane calculi [2] start from the same reality, the living cell, but they have different objectives and develop in different directions. While membrane computing tries to abstract computing models, in the Turing sense, from the structure and the functioning of the cell, making use especially of automata and language theoretic tools, brane calculi pay more attention to the fidelity to the biological reality, have as a primary target systems biology, and use especially the framework of process algebra. Various operations with membranes appear in both areas, fully idealized in the former area, less idealized in the latter. For instance, an important distinction concerns the role the membranes play in the two fields: separators of compartments in membrane computing, with

the computation done *inside* compartments (the main data structure used is the multiset of abstract objects placed in the compartments of a cell), and "first class citizens" in brane calculi, with the emphasis put on the structure, properties, and evolution of membranes; in particular, in the latter case the membranes are the support of bio-chemistry, with the proteins embedded in them being central to the cell evolution, rather than the chemicals from the compartments.

All these similarities and differences raise the natural question of bridging the two research areas, and this paper is a first attempt in this respect.

We consider here two brane calculi, the one using the *pino, exo, phago* operations, and the one using the *mate, drip, bud* operations. In the next section we recall the basic elements concerning these operations and calculi. As we will see, the first calculus is more expressive than the second one, we can simulate the latter operations by the former ones. This observation was also confirmed in [1] in what concerns the computational power of the two calculi: Turing universality in the first case (in fact, using only *phago* and *exo* was sufficient), decidability in the second case. It should be noted that the proofs from [1] were done by simulating Minsky register machines, with the register values encoded in the number of embedded membranes, using the mentioned operations as introduced in [2], formalized in process algebraic terms.

In what follows, we write the previous operations – actually, only *pino, exo, mate, drip*, with two possible variants in the case of *pino, exo* – in the formalism of membrane computing, with slight differences in what concerns the evolution of proteins which control the operations. Like in brane calculi, the obtained P systems work only with the membranes and the proteins embedded in them (no object is present in the compartments), but using the style of membrane computing: the rules are applied in the maximally parallel manner (if a membrane can evolve, it has to do it), and considering as successful only the halting computations. Formal details will be given in Section 5. In this setup, we find that systems using the *mate, drip* operations are Turing complete, they can compute all Turing computable sets of numbers (or of vectors of numbers). It is of interest to note that we do not use the *bud* operation, and that our result essentially differs from that in [1] (the "explanation" lies both in the versions of the operations we use, and in the fact that we encode the information we work with in a different manner – in turn, the number of current membranes used in our proof is very small, at most eleven, while in [1] this number can be arbitrarily large).

After the brane calculi details from Section 2, we introduce (Section 3) the necessary notions and notations of language theory we need in the proof (especially, matrix grammars with appearance checking), then we define the operations (Section 4) and the classes of P systems we investigate here (Section 5). Section 6 gives the above mentioned universality proof for *mate, drip*. We close with a short discussion in Section 7, especially pointing out further research topics.

2 The Pino/Exo/Phago and Mate/Drip/Bud Calculi

We give an informal introduction to the basic operations of brane calculi, as a prelude to their use later in P systems. A formal treatment of brane calculi can be found in [2]. There, a concrete syntax is first defined for membrane configurations. The syntax is factored by a congruence relation, e.g., to add multiset laws and other "chemical mixing" rules. Then, a binary reduction relation (invariant under congruence) is defined inductively. The reduction relation includes the basic reductions that correspond to the basic operations. The transitive closure of the reduction relation describes the possible, non-deterministic, evolutions of a configuration.

More informally, a *membrane system* consists of a collection of nested membranes. Membranes are formed of patches s, where a patch can be a composition of sub-patches $s_1|s_2$. An elementary patch consists of an action a followed (after the action is consumed) by another patch: $a.s$. Actions come (often) in complementary pairs that cause interactions between subsystems. Here is a system consisting of two membranes (written as brackets $[\ldots]$) nested inside a third (membranes carry patches, indicated by s, t, u):

$$[\ [\ ]\ s\ [\ ]\ t\ ]u \qquad \text{a membrane system}$$

Each specific brane calculus has a fixed collection of actions with a specific operational meaning in terms of reductions. Let us start with the *pino* action, that has no complementary co-action. Here P, Q are arbitrary subsystems, u, s, t are arbitrary patches, and the symbol $\rightarrow$ means "may reduce to" (a number of different reductions may be possible out of the same configuration).

$$\text{(pino)} \quad [\ P\ ]u|(pino(s).t) \rightarrow [\ P\ [\ ]\ s\ ]u|t$$

The action $pino(s)$ creates an empty bubble within the membrane where the pino action resides; we should imagine that the original membrane buckles towards the inside and pinches off. The patch on the empty bubble so created, s, is a parameter to pino.

Next, we consider the *exo* action, which comes with a complementary co-action:

$$\text{(exo)} \quad [\ [\ P\ ]u|(exo.t)Q]w|(co\text{-}exo.v) \rightarrow P[\ Q\ ]u|w|t|v$$

This operation models the merging of two nested membranes, which starts with the membranes touching at a point. In the process (which is a smooth, continuous process), the subsystem P gets expelled to the outside, and all the residual patches of the two membranes, u, w, t, v, become contiguous.

Finally we consider the *phago* action, which also comes with a complementary co-action:

$$\text{(phago)} \quad [\ P\ ]u|(phago.t)[\ Q\ ]w|(co\text{-}phago(s).v) \rightarrow [\ [\ [\ P\ ]u|t\ ]sQ\ ]w|v$$

This operation models a membrane (the one with Q) "eating" another membrane (the one with P). But, again, the process has to be smooth and continuous, so it is biologically implementable. It proceeds by the Q membrane wrapping around the P membrane and joining itself on the other side. Hence, an additional layer of membrane is created around the eaten membrane: the patch on that membrane is specified by the parameter s on the *co-phago* action (similar to the parameter on the *pino* action).

These three operations, *pino, exo, phago*, form a complete set, in the sense, e.g., that the brane calculus with such operations is Turing complete under opportune conditions (since operations are always consumed, a form of iteration is needed as well). Moreover, these three operations are sufficiently mechanistic that it is possible to write meaningful programs and algorithms with them.

Those are, however, not the only operations one can imagine on membranes, and in fact are not the only ones that have a biological realization. For example, we can consider a different brane calculus with the following three operations.

$$\begin{aligned}
&\text{(drip)} && [\ P\ ]u|(drip(s).t) \rightarrow [\ P\ ]u|t[\ ]s,\\
&\text{(mate)} && [\ P\ ]u|(mate.t)[Q]w|(co\text{-}mate.v) \rightarrow [\ PQ\ ]u|t|wv,\\
&\text{(bud)} && [\ [\ P\ ]u|(bud.t)Q]w|(co\text{-}bud(s).v) \rightarrow [\ [\ P\ ]u|t\ ]s[\ Q\ ]w|v.
\end{aligned}$$

Drip produces an empty bubble (like *pino*), but outside a membrane. *Mate* merges two membranes (like *exo*), but the membranes are not originally nested. *Bud* expels a membrane from inside another one (the opposite of *phago*), but still wrapping an additional layer.

It turns out that the *drip, mate, bud* operations are expressible in the *pino, exo, phago* calculus by simple operations, but the *drip, mate, bud* calculus is not Turing complete (under the same appropriate conditions) [1]. In practice, all six operations have separate, direct, biological implementations, and therefore can all be considered "primitives".

3 Prerequisites (Matrix Grammars)

Excepting the matrix grammars, which will be introduced immediately, all notions of formal language theory we use are elementary, and can be found in any basic monograph, such as [9], [6]. Details about matrix grammars can be found in [3], [4], and in the introductory chapter of [8]. We specify here only the notation we use.

For an alphabet V, by V^* we denote the set of all strings over V, the empty string λ included; the set of non-empty strings over V, that is $V^* - \{\lambda\}$, is denoted by V^+. The length of a string $x \in V^*$ is denoted by $|x|$, and $|x|_a$, for $a \in V$, is the number of occurrences of the symbol a in x. If $V = \{a_1, \ldots, a_n\}$, then the Parikh mapping associated with V is $\Psi_V : V^* \longrightarrow \mathbf{N}^n$, defined by $\Psi_V(x) = (|x|_{a_1}, \ldots, |x|_{a_n})$, for all $x \in V^*$. For a language $L \subseteq V^*$, $lg(L) = \{|x| \mid x \in L\}$

is the length set, and $\Psi_V(L) = \{\Psi_V(x) \mid x \in L\}$ is the Parikh set/image of L. The left derivative of $L \subseteq V^*$ with respect to a string $x \in V^*$ is defined by $\partial_x^l(L) = \{w \in V^* \mid xw \in L\}$.

The four basic families of languages from Chomsky hierarchy – regular, context-free, context-sensitive, recursively enumerable – are denoted by REG, CF, CS, RE, respectively (RE is the family of languages which can be recognized by Turing machines). For a family FL of languages, we denote by $NFL, PsFL$ the family of length sets and of Parikh images of languages from FL (hence NRE is the family of Turing computable sets of numbers, and $PsRE$ is the family of sets of vectors of numbers which are Turing computable).

A multiset over an alphabet $V = \{a_1, \ldots, a_n\}$ is a mapping $m : V \longrightarrow \mathbf{N}$. Because the natural extension of m to strings over V gives the Parikh mapping, we can represent the multiset m by any string $w \in V^*$ such that $\Psi_V(w) = (m(a_1), \ldots, m(a_n))$ (thus, if a string w represents the multiset m, then all permutations of w represent it). The empty multiset is represented by λ. Operations with multisets are defined in this way as operations with strings.

In our universality proof we use the characterization of recursively enumerable languages by means of matrix grammars with appearance checking. Such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets (of non-terminals and terminals, respectively), $S \in N$ (axiom), M is a finite set of *matrices*, that is, sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$, and F is a set of occurrences of rules in the matrices of 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 the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either (1) $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or (2) $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (All rules which are not in F should be applied. If applicable, the rules from F should be applied, but if they cannot be applied, then we may skip them. That is why the rules from F are said to be applied in the *appearance checking* mode.) If $F = \emptyset$, then the grammar is said to be without appearance checking.

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$, where $\Longrightarrow^*$ is the reflexive and transitive closure of the relation $\Longrightarrow$.

The family of languages of this form is denoted by MAT_{ac}, while the family of languages generated by matrix grammars without appearance checking is denoted by MAT. The following results are known:

1. $CF \subset MAT \subset MAT_{ac} = RE$.
2. $NCF = NMAT \subset NRE$, $PsCF \subset PsMAT \subset PsRE$ ($PsCF$ contains only semilinear sets, while $PsMAT$ contains non-semilinear sets of vectors).

We say that a matrix grammar with appearance checking $G = (N, T, S, M, F)$ is in the *Z-binary normal form* if $N = N_1 \cup N_2 \cup \{S, Z, \#\}$, with these three sets mutually disjoint, and the matrices in M are in one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,

2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$,
3. $(X \to Y, A \to \#)$, with $X \in N_1, Y \in N_1 \cup \{Z\}, A \in N_2$,
4. $(Z \to \lambda)$.

Moreover, there is only one matrix of type 1, F consists exactly of all rules $A \to \#$ appearing in matrices of type 3 (# is a trap-symbol, if it is introduced, then it cannot be removed), and, if a sentential form generated by G contains the symbol Z, then it is of the form Zw, for some $w \in (T \cup \{\#\})^*$ (that is, the appearance of Z makes sure that, except for Z, all symbols are either terminal or the trap-symbol #). The matrix of type 4 is used only once, in the last step of a derivation.

For each language $L \in RE$ there is a matrix grammar with appearance checking G in the Z-binary normal form such that $L = L(G)$.

Convention. When comparing two language generating devices G_1, G_2, the empty string is ignored, that is, we consider G_1 equivalent with G_2 if $L(G_1) - \{\lambda\} = L(G_2) - \{\lambda\}$. Similarly, we ignore the number zero (and the zero vector) when comparing number (vector) generating devices.

4 Pino/Exo and Mate/Drip as Membrane Computing Operations

In what follows, the reader is assumed to have some familiarity with membrane computing, e.g., from [8]: current details can be found at the web address [10].

We start by writing the four operations we use below, *pino, exo, mate, drip*, using the formalism of membrane computing, as we will immediately see, with two possible variants for *pino, exo*.

In this framework, we represent a membrane by a pair of square brackets, [ ]. If necessary, the brackets can have labels, written as $[\ ]^i$, in order to refer them, but in what follows we avoid the use of labels. However, we associate here with membranes *multisets of proteins* (we also use to speak about "objects" instead of "proteins"). A membrane having associated a multiset u of proteins (remember that we represent multisets by strings) is written in the form $[\ ]_u$; we also use to say that the membrane is *marked* with the multiset u. The intuition is that the proteins are placed *on* the membrane, intercalated with its phospholipidic molecules.

Then, taking an alphabet V of proteins, the four operations – with two possibilities from *pino, exo* – are written as follows:

$$pino_i : [\ ]_{uav} \to [[\ ]_{ux}]_v, \tag{1}$$
$$exo_i : [[\ ]_{ua}]_v \to [\ ]_{uxv}, \tag{2}$$
$$pino_e : [\ ]_{uav} \to [[\ ]_v]_{ux}, \tag{3}$$
$$exo_e : [[\ ]_u]_{av} \to [\ ]_{uxv}, \tag{4}$$
$$mate : [\ ]_{ua}[\ ]_v \to [\ ]_{uxv}, \tag{5}$$
$$drip : [\ ]_{uav} \to [\ ]_{ux}[\ ]_v, \tag{6}$$

in all cases with $a \in V$, $u, x \in V^*$, $v \in V^+$, with $ux \in V^+$ for *pino, drip* rules. Therefore, each membrane, from the left hand or the right hand of any rule, has associated a non-empty multiset of proteins. Of course, this restriction can be relaxed, but in this paper we work in this setup. The length of the string uxv (hence the total multiplicity of the multiset represented by this string) from each rule is called the *weight* of the rule.

In each case, multisets of proteins are transferred from input membranes to output membranes as indicated in the rules, with protein a evolved into the multiset x (which can be empty).

Note the important fact that the multisets u, v and the protein a marking the left hand membranes of these rules correspond to the multisets u, v, x from the right hand side of the rules; specifically, the multiset uxv resulting when applying the rule is precisely split into ux and v, with these two multisets assigned to the two new membranes.

The difference between $pino_i, exo_i$ (with i from "internal") and $pino_e, exo_e$ (with e from "external") is that in the first case the "main role" is played by the internal membrane, and in the second case the external membrane contains the proteins a and x. We still use *pino, exo* as generic names for $pino_i, exo_i$ and $pino_e, exo_e$.

The rules are applied to given membranes if they are marked with multisets of proteins which include the multisets of proteins mentioned in the left hand of rules; all proteins not specified in the rules are not affected by the use of rules, but, in the case of *pino* and *drip*, they are randomly distributed to the two resulting membranes.

Because this is an important point, we also give a more formal definition.

Assume that we have a membrane $[\ ]_{zuav}$, for $a \in V, u, v, z \in V^*$. By a $pino_i$ rule as in (1), we obtain any one of the pairs of membranes $[\ [\ ]_{z_1ux}\ ]_{z_2v}$ such that $z = z_1z_2,\ z_1, z_2 \in V^*$; by a $pino_e$ rule as in (3), we obtain any one of the pairs of membranes $[\ [\ ]_{z_1v}\ ]_{z_2ux}$ such that $z = z_1z_2,\ z_1, z_2 \in V^*$; by a *drip* rule as in (6) we obtain any one of the pairs of membranes $[\ ]_{z_1ux}[\ ]_{z_2v}$, such that $z = z_1z_2,\ z_1, z_2 \in V^*$. In all cases, we have as many possible results as many decompositions of the multiset z into two multisets exist.

In the case of the other operations, the result is uniquely determined. From a pair of membranes $[\ [\ ]_{z_1ua}]_{z_2v}$, by an exo_i rule as in (2) we obtain the membrane $[\ ]_{z_1z_2uxv}$, and from $[\ [\ ]_{z_1u}]_{z_2av}$, by an exo_e rule as in (4) we obtain the membrane $[\ ]_{z_1z_2uxv}$. In turn, from a pair of membranes $[\ ]_{z_1ua}[\ ]_{z_2v}$, by means of a *mate* rule as in (5) we obtain the membrane $[\ ]_{z_1z_2uxv}$. In all cases, z_1, z_2 are arbitrary multisets over V. The former three types of rules introduce non-determinism in the evolution of the membranes, the latter three types cancel the non-determinism introduced by the former rules – providing that they are applied to the pair of membranes produced by the first rules.

The contents of membranes involved in these operations is transferred from the input membranes to the output membranes in the same way as in brane calculus, with the mentioning that here we have no objects inside a membrane, but possi-

bly only other membranes. Denoting these contents (empty or consisting of other membranes) by **P, Q** as in Section 2, we can write the six operations as follows:

$$pino_i : [\ \mathbf{P}\ ]_{uav} \to [\,[\ ]_{ux}\ \mathbf{P}\ ]_v, \tag{7}$$
$$exo_i : [\,[\ \mathbf{P}\ ]_{ua}\ \mathbf{Q}\ ]_v \to \mathbf{P}\ [\ \mathbf{Q}\ ]_{uxv}, \tag{8}$$
$$pino_e : [\ \mathbf{P}\ ]_{uav} \to [\,[\ ]_v\ \mathbf{P}\ ]_{ux}, \tag{9}$$
$$exo_e : [\,[\ \mathbf{P}\ ]_u\ \mathbf{Q}\ ]_{av} \to \mathbf{P}\ [\ \mathbf{Q}\ ]_{uxv}, \tag{10}$$
$$mate : [\ \mathbf{P}\ ]_{ua}[\ \mathbf{Q}\ ]_v \to [\ \mathbf{P}\ \mathbf{Q}\ ]_{uxv}, \tag{11}$$
$$drip : [\ \mathbf{P}\ ]_{uav} \to [\ ]_{ux}[\ \mathbf{P}\ ]_v. \tag{12}$$

Note that from the point of view of handling the contents, we can distinguish a variant of *drip*, in the form

$$[\ \mathbf{P}\ ]_{uav} \to [\ \mathbf{P}\ ]_{ux}[\ ]_v.$$

However, in the proof of the theorem from Section 6 we will use only membrane structures with two levels, with the active membranes placed in the inert skin membrane; therefore, always the contents of membranes (other than the skin) will be empty, so no care should be paid to the evolution of the contents (**P, Q** above) during the computations.

5 P Systems Based on the Previous Operations

Using rules as defined above, we can define a machine-oriented computing model in the form of a P system as follows:

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

where:

1. A is an alphabet (finite, non-empty) of proteins;
2. μ is a membrane structure with $m \geq 2$ membranes;
3. $u_1, \ldots, u_m$ are multisets of proteins (represented by strings over A) bound to the m membranes of μ at the beginning of the computation (one assumes that the membranes in μ have a precise identification, e.g., by means of labels, or of other "names", in order to have the marking by means of $u_1, \ldots, u_m$ precisely defined; the labels play no other role than specifying this initial marking of membranes); the skin membrane is labelled with 1 and $u_1 = \lambda$;
4. R is a finite set of *pino, exo, mate, drip* rules, of the forms specified above, with the proteins from the set A.

Note that the skin membrane has no protein associated. Actually, in what follows, it plays no role in the computation, *no rule can be applied to it*. The skin membrane is only meant to keep together the "computer", to delimit it from the

environment. Also, we have to stress that there is no object in the compartments of μ; a membrane can contain other membranes inside, but in-between membranes there is nothing.

When using any rule of any type, we say that the membrane(s) from its left hand side are *involved* in the rule; they all are "consumed", and the membranes from the right hand side of the rule are produced instead. Similarly, the protein a specified in the left hand side of rules is "consumed", and it is replaced by the multiset x. It is important to mention again that the proteins other than a which mark the membranes which are consumed remain unchanged, and they are transferred to the newly created membranes. In the case of rules producing only one membrane – *exo, mate* – all proteins are inherited by the new membrane; in the case of rules producing two new membranes – *pino, drip* – the proteins of the old membrane which are not involved in the rule are non-deterministically distributed to the two new membranes.

The evolution of the system is done through *transitions* among *configurations*, based on the *non-deterministic maximally parallel* use of rules. A configuration consists of the membrane structure of the system and the multisets marking the membranes; thus, the initial configuration is that defined by μ and $u_1, \ldots, u_m$. In what follows, we will specify the configurations by writing the markings as subscripts of the right hand parentheses which identify the membrane. For instance, $[\ [\ ]_{aabbcde}\ ]_\lambda$ describes the configuration consisting of the skin membrane (with no protein on it) and the inner membrane marked with *aabbcde*. In each step (a global clock is assumed), we choose non-deterministically and apply in a parallel manner a maximal set of rules which can be applied to the configuration at hand. That is, which rules to apply and to which membranes to apply is randomly decided, all possible choices are allowed (of course, with the restriction that the multisets u, v and the protein a from the left hand of the rules are included in the multisets marking the membranes to which the rules are applied). In each step, any membrane can be involved in at most one rule. However, the choice of rules should be maximal, in the sense that after choosing some rules to apply, no further rule can be applied to the membranes which are not involved in the chosen rules. A membrane remains unchanged if not evolving by a rule. The skin membrane never evolves.

Again, an example can be useful. If we have the configuration

$$[\ [\ ]_a\ [\ ]_b\ [\ ]_c\ [\ ]_d\ ]_\lambda,$$

and the *mate* rules

$$r_1 : [\ ]_a[\ ]_b \to [\ ]_{ab},$$
$$r_2 : [\ ]_c[\ ]_d \to [\ ]_{cd},$$
$$r_3 : [\ ]_b[\ ]_c \to [\ ]_{bc},$$

then using only r_1 is not maximal, because also r_2 can be applied to the remaining membranes; using r_1, r_2 in parallel, or only r_3 is maximal, because the remaining

membranes (none in the first case) cannot evolve by the available rules. Similarly, using only once r_1 for evolving

$$[\ [\ ]_a\ [\ ]_b\ [\ ]_a\ [\ ]_b\ ]_\lambda,$$

is not allowed/maximal, because both pairs of membranes $[\ ]_a[\ ]_b$ have to evolve, hence we have to apply r_1 twice.

Note the important detail that the evolution is parallel at the level of membranes, but sequential at the level of each multiset marking a membrane: at most one protein, a, evolves by applying a rule, and at most one rule is applied to each membrane.

The contents of membranes evolving by means of any rule as above is passed to the resulting membranes as indicated in (7) – (12), with the mentioning that, because the contents consist of membranes, they can evolve at the same time (due to the parallelism). For instance, starting from the configuration $[\ [\ [\ ]_{cc'}\ ]_a\ ]_b\ ]_\lambda$ and using the rules $(pino_i)\ [\ ]_{cc'} \to [\ [\ ]_c\ ]_{c'}$ and $(exo_i)\ [\ [\ ]_a\ ]_b \to [\ ]_{ab}$, we obtain the configuration $[\ [\ [\ ]_c\ ]_{c'}\ [\ ]_{ab}\ ]_\lambda$ (the innermost membrane evolves to $[\ [\ ]_c\ ]_{c'}$ by the *pino* rule, at the same time with the evolution of the membranes marked with a and b through the *exo* rule; in this way, the contents of the former membrane marked by a will be placed outside the membrane marked by ab).

Because in what follows we work only with membranes placed on two levels (the skin containing membranes which evolve through *mate, drip* operations, hence always obtaining membrane structures with two levels of nesting), and this is a simpler and more transparent case, we avoid a cumbersome formal definition of the way the membrane structures are changed by means of the rules a system can contain, and rely on the intuition provided by equations (7) – (12), which, in turn, correspond to the brane calculi operations as introduced in Section 2.

A sequence of transitions constitutes a *computation*. A computation which starts from the initial configuration is *successful* if (i) it halts, that is, it reaches a configuration where no rule can be applied, and (ii) in the halting configuration there are only two membranes, the skin (marked with λ) and an inner one. The *result* of a successful computation is given by the multiset which marks the inner membrane in the halting configuration. Here we consider as the result the vector describing the multiplicity of proteins in this multiset. Such a vector is said to be computed by the system Π. Note that the computations which do not halt, or halt with more than one inner membrane provide no output. (The latter condition can be relaxed, by considering a special membrane as the output one – which makes again necessary considering labels – or, simply, by taking as results of the computation all multisets marking membranes in the halting configuration; a filtering can be also considered in the second case, accepting only membranes whose markings contain/do not contain certain designated objects. Here we work in the restricted setup mentioned above, with the successful computations halting with only one inner membrane.)

Because of the non-determinism in using the rules, starting from the initial configuration we can get several computations, possibly several of them successful.

The set of all vectors (from $\mathbf{N}^n$, for $n = card(A)$) computed in this way by Π is denoted by $Ps(\Pi)$ (with "Ps" coming from "Parikh set").

Note that, because of the restriction we have started with, to have ux and v non-empty in each rule, we cannot compute the number/vector zero, that is why zero is ignored when comparing two computing devices.

In what follows we only investigate the power of P systems using the *mate, drip* operations.

The family of all sets of vectors $Ps(\Pi)$ computed by P systems Π using at any moment during a halting computation at most m membranes, and *mate, drip* rules of weight at most r, s, respectively, is denoted by $PsOP_m(mate_r, drip_s)$. When one of the parameters m, r, s is not bounded we replace it with $*$.

The system we use in the proof below is described in good details, so that we do not illustrate here the previous definitions with any example.

We end this section by pointing out some results which follow directly from the definitions (and from Turing-Church thesis – this can be also done through a cumbersome direct construction of a Turing machine simulating a P system as above).

Lemma 1. (i) $PsOP_m(mate_r, drip_s) \subseteq PsOP_{m'}(mate_{r'}, drip_{s'})$, *for all* $m \leq m'$, $r \leq r'$, $s \leq s'$.

(ii) $PsOP_*(mate_*, drip_*) \subseteq PsRE$.

6 Universality for the Mate/Drip Case

We consider here only the systems based on the *mate, drip* operations. Their study is simpler than that of systems using *pino, exo* operations, because the membrane structure cannot grow the depth, and all membranes from the same level can mate with each other. In the case of *pino, exo*, the membrane structure can get more complex and the vertical nesting imposes a precise neighboring relation, with only adjacent membranes allowed to evolve through *exo*.

What is interesting is that the Turing completeness (because the proof is constructive and the main data we process are the multisets of objects placed on membranes, this also implies universality, in the standard sense, that is why we use the words completeness and universality as interchangeable) is obtained for systems with a reduced number of membranes and using rules of a reduced weight.

Theorem 1. $PsRE = PsOP_m(mate_r, drip_s)$ *for all* $m \geq 11, r \geq 5$, *and* $s \geq 5$.

Proof. In view of Lemma 1, we only have to prove the inclusion $PsRE \subseteq PsOP_{11}(mate_5, drip_5)$, and to this aim, we use the equality $PsRE = PsMAT_{ac}$.

Let us consider a set of vectors $Q \in \mathbf{N}^n$, for some $n \geq 1$. We have $Q = \Psi_V(L)$, for some language $L \in RE = MAT_{ac}$, $L \subseteq V^*$, for an alphabet V with n symbols. We write this language in the form

$$L = (L \cap \{\lambda\}) \cup \bigcup_{a \in V} \{a\}\partial_a^l(L).$$

The family MAT_{ac} is closed under left derivatives, hence a matrix grammar with appearance checking $G_a = (N_a, V, S_a, M_a, F_a)$ exists such that $L(G_a) = \partial_a^l(L)$, for each $a \in V$. We consider these grammars G_a in the Z-normal form introduced in Section 3, hence with $N_a = N_{a,1} \cup N_{a,2} \cup \{S_a, Z_a, \#\}$, and matrices of the four mentioned types. We assume the alphabets N_a mutually disjoint and we "put together" the grammars $G_a, a \in V$, constructing the matrix grammar $G = (N, V, S, M, F)$ with

$$\begin{aligned}
N &= N_1 \cup N_2 \cup \{Z_a \mid a \in V\} \cup \{S, \#\},\\
N_1 &= \bigcup_{a \in V} N_{a,1},\\
N_2 &= \bigcup_{a \in V} N_{a,2},\\
M &= \{(S \to XA) \mid (S_a \to XA) \in M_a, a \in V\}\\
&\quad \cup \{(X \to Y, A \to x) \mid \text{for } (X \to Y, A \to x) \text{ a two-rule matrix in } M_a, a \in V\}\\
&\quad \cup \{(Z_a \to a) \mid a \in V\}.
\end{aligned}$$

Obviously, $L(G) = L$ (modulo the empty string, which is ignored; note that instead of rules $Z_a \to \lambda$ we have here $Z_a \to a$).

We assume that all two-rules matrices from M are injectively labelled, in the form $m_l : (X \to Y, A \to x)$, $l \in Lab$, for a set Lab of labels.

Starting from the grammar G we now construct a P system

$$\Pi = (A, [\ [\ ]\ ], \lambda, S_1S_2, R),$$

with the alphabet

$$\begin{aligned}
A &= \{X, X', X'', X''', X^{iv}X^v, X^{vi}, X^{vii} \mid X \in N_1\}\\
&\quad \cup \{X_l \mid l \in Lab\}\\
&\quad \cup \{\alpha, \alpha' \mid \alpha \in N_2 \cup V\}\\
&\quad \cup \{E_A \mid A \in N_2\}\\
&\quad \cup \{a'', Z_a \mid a \in V\}\\
&\quad \cup \{E, E', H, H', H'', S_1, \ldots, S_9, c_1, \ldots, c_8, c_1', \ldots, c_8', d_1, d_2, d_1', d_1'', d_2', f', \#\},
\end{aligned}$$

and the rules from the set R as constructed below (we give them in groups dedicated to various tasks to accomplish during the computation).

Any computation starts from the configuration $[\ [\ ]_{S_1S_2}\ ]_\lambda$, by using rules from the following group:

$$\begin{aligned}
&\text{Step 1 } [\ ]_{S_1S_2} \to [\ ]_{S_3S_4}[\ ]_{S_2},\\
&\text{Step 2 } [\ ]_{S_3S_4} \to [\ \ ]_{S_5S_6XA}[\ ]_{S_4},
\end{aligned}$$

Step 3 $[\,]_{S_5S_6XA} \to [\,]_{c_1c_1'}[\,]_{S_6XA}$, $[\,]_{S_2}[\,]_{S_4} \to [\,]_{S_7S_4}$,

Step 4 $[\,]_{S_6XA} \to [\,]_{S_9}[\,]_{XA}$, $[\,]_{c_1c_1'} \to [\,]_{c_1}[\,]_{c_1'}$, $[\,]_{S_4S_7} \to [\,]_{S_8}[\,]_{S_7}$,

Step 5 $[\,]_{S_9}[\,]_{XA} \to [\,]_{EHXA}$, $[\,]_{S_8}[\,]_{c_1} \to [\,]_{c_1d_1}$, $[\,]_{S_7}[\,]_{c_1'} \to [\,]_{c_1'd_1'}$,
for $(S_a \to XA) \in M_a, a \in V$,
$[\,]_{S_8}[\,]_{S_7} \to [\,]_{\#\#S_7}$.

These rules are meant to simulate the initial matrices $(S \to XA)$ from M (corresponding to $(S_a \to XA) \in M_a$, for $a \in V$), thus preparing the system for simulating two-rules matrices from M. After using the rule $[\,]_{S_1S_2} \to [\,]_{S_3S_4}[\,]_{S_2}$, the only applicable rule in the next step is $[\,]_{S_3S_4} \to [\quad]_{S_5S_6XA}[\,]_{S_4}$. We have three inner membranes present, $[\,]_{S_5S_6XA}$, $[\,]_{S_2}$, and $[\,]_{S_4}$. If we continue with rule $[\,]_{S_5S_6XA} \to [\,]_{c_1c_1'}[\,]_{S_6XA}$ for the first membrane, then the next two steps continue deterministically using the rules indicated above for steps 4 and 5.

Assume that we take the "wrong" path, of using instead the rule $[\,]_{S_6XA} \to [\,]_{S_9}[\,]_{XA}$ (this happens in parallel with using the rule $[\,]_{S_2}[\,]_{S_4} \to [\,]_{S_7S_4}$, which produces the membrane $[\,]_{S_7S_4}$); in this case, the objects c_1, c_1' are not introduced. In the next step we use the rule $[\,]_{S_4S_7} \to [\,]_{S_8}[\,]_{S_7}$, and now the only rule applicable to membranes obtained in this way is $[\,]_{S_8}[\,]_{S_7} \to [\,]_{\#\#S_7}$, which introduces the trap-object $\#$.

We also introduce the following rules for handling the trap-object:

$$[\,]_{\#\#} \to [\,]_{\#}[\,]_{\#}, \; [\,]_{\#}[\,]_{\#} \to [\,]_{\#\#}.$$

By using these rules, the computation can continue forever (if the two trap-objects are together, then there is a rule to apply, if they are separated, then again there is a rule to apply).

Therefore, the only computation which can terminate is the one leading to three inner membranes of the form

$$[\,]_{EHXA}, \; [\,]_{c_1d_1}, \; [\,]_{c_1'd_1'},$$

where $(S_a \to XA) \in M_a$, for some $a \in V$. We use to refer to these membranes (and to their descendants) as "the first membrane", "the second" and "the third" one, although, obviously, there is no ordering of them in the system.

In what follows, we have to have in mind that the symbols from N_1 "remember" which is the symbol $a \in V$ to be introduced in the end by a rule $Z_a \to a$, and that all alphabets $N_{a,2}$ were supposed mutually disjoint, hence no interference of the rules corresponding to various grammars G_a is possible.

Let us assume that, after these first five steps, we have produced the configuration $[\,[\,]_{EHXA}\,[\,]_{c_1d_1}\,[\,]_{c_1'd_1'}\,]_\lambda$, for XA the starting sentential form for a grammar G_a. By repeatedly applying *mate* and *drip* operations to the membrane marked with $XAEH$, we simulate a derivation in grammar G; the membranes with markers $c_1d_1, c_1'd_1'$ and their successors will help in this process, more exactly, in

simulating the matrices with rules to be used in the appearance checking mode. The simulation of any matrix will take 9 steps.

For a better understanding of the nine steps (hence of the interplay between the membranes which will emerge from the three initial inner membranes), we present the respective rules in Table 1 and Table 2; the first table refers to simulating a matrix without a rule to be used in the appearance checking mode, the second table refers to matrices with rules used in the appearance checking mode. The last two columns are the same in the two tables, and they indicate the rules used for evolving the auxiliary membranes, those having initially the markings $c_1 d_1$ and $c'_1 d'_1$. The simulation of matrices is done through the multisets marking the membranes emerging from the membrane with the marking $EHXA$ – at a current stage, this membrane is supposed to be marked with $[\ ]_{EHzXA}$, for some $z \in (N_2 \cup V)^*$; initially, this is the case, with $z = \lambda$.

In Table 1, the primed string x' is defined as follows. If $x = x_1\alpha$, for some $\alpha \in N_2 \cup V$, then $x' = x_1\alpha'$; if $x = \lambda$, then $x' = f'$. In the latter case, $\alpha' = f'$, but in row 4 the symbol α is replaced by λ.

Step 1	$[\ ]_{EHXA} \to [\ ]_{EHX_l}[\ ]_A$	$[\ ]_{c_1 d_1} \to [\ ]_{c_2}[\ ]_{d_1}$	$[\ ]_{c'_1 d'_1} \to [\ ]_{c'_2}[\ ]_{d'_1}$
Step 2	$[\ ]_A[\ ]_{EHX_l} \to [\ ]_{x'EHX_l}$	$[\ ]_{c_2}[\ ]_{d_1} \to [\ ]_{c_3 c_4 d_1}$	$[\ ]_{c'_2}[\ ]_{d'_1} \to [\ ]_{c'_3 c'_4 d'_1}$
Step 3	$[\ ]_{X_l \alpha' EH} \to [\ ]_{Y'}[\ ]_{\alpha' EH}$	$[\ ]_{d_1 c_3 c_4} \to [\ ]_{d_2}[\ ]_{c_3 c_4}$	$[\ ]_{d'_1 c'_3 c'_4} \to [\ ]_{d'_2}[\ ]_{c'_3 c'_4}$
Step 4	$[\ ]_{EH\alpha'}[\ ]_{Y'} \to [\ ]_{EH\alpha Y'}$	$[\ ]_{c_4 c_3}[\ ]_{d'_2} \to [\ ]_{c_4 c_5 d'_2}$	$[\ ]_{c'_3 c'_4} \to [\ ]_{c'_5}[\ ]_{c'_4}$
Step 5	$[\ ]_{EHY'}[\ ]_{d_2} \to [\ ]_{EHY'' d_2}$	$[\ ]_{d'_2 c_4 c_5} \to [\ ]_{c_6 c_7}[\ ]_{c_4 c_5}$	$[\ ]_{c'_4}[\ ]_{c'_5} \to [\ ]_{d'_1 c'_5}$
Step 6	$[\ ]_{Ed_2 Y'' H} \to [\ ]_{Ed_2 Y'''}[\ ]_H$	$[\ ]_{c_6 c_7} \to [\ ]_{d_1}[\ ]_{c_7}$ $[\ ]_{c_4 c_5} \to [\ ]_{c_1}[\ ]_{c_5}$	$[\ ]_{c'_5 d'_1} \to [\ ]_{c'_6}[\ ]_{d'_1}$
Step 7	$[\ ]_{Y''' Ed_2}[\ ]_H \to [\ ]_{Y''' EH}$	$[\ ]_{c_5}[\ ]_{c_7} \to [\ ]_{c_8 c_7}$	$[\ ]_{c'_6}[\ ]_{d'_1} \to [\ ]_{c'_7 d'_1}$
Step 8	$[\ ]_{Y''' EH} \to [\ ]_{Y^{iv}}[\ ]_{EH}$	$[\ ]_{c_8 c_7}[\ ]_{d_1} \to [\ ]_{c_8 d_1}$	$[\ ]_{c'_7 d'_1} \to [\ ]_{c'_8}[\ ]_{d'_1}$
Step 9	$[\ ]_{Y^{iv}}[\ ]_{EH} \to [\ ]_{YEH}$	$[\ ]_{d_1 c_8}[\ ]_{c_1} \to [\ ]_{d_1 c_1}$	$[\ ]_{c'_8}[\ ]_{d'_1} \to [\ ]_{c'_1 d'_1}$

Table 1. Rules for simulating a matrix $m_l : (X \to Y, A \to x) \in M$

Besides the rules in these tables, we also consider the following rules:

$$[\ ]_{XEH} \to [\ ]_{\#\#}[\ ]_{EH},$$
$$[\ ]_E[\ ]_{d_2} \to [\ ]_{\#\# d_2},$$
$$[\ ]_{c_3 c_4} \to [\ ]_{\#\#}[\ ]_{c_4},$$
$$[\ ]_{c_1 d'_2} \to [\ ]_{\#\#}[\ ]_{d'_2},$$
$$[\ ]_{c_5 d'_2} \to [\ ]_{\#\#}[\ ]_{d'_2}.$$

Let us examine in some details how the simulation proceeds. Assume that we have a membrane marked with a multiset $zXEH$. If no matrix can be simulated,

Step 1	$[\ ]_{XEH} \to [\ ]_X [\ ]_{EH}$	$[\ ]_{c_1 d_1} \to [\ ]_{c_2} [\ ]_{d_1}$	$[\ ]_{c'_1 d'_1} \to [\ ]_{c'_2} [\ ]_{d'_1}$
Step 2	$[\ ]_{HE} [\ ]_{X_l} \to [\ ]_{HE_B X_l}$	$[\ ]_{c_2} [\ ]_{d_1} \to [\ ]_{c_3 c_4 d_1}$	$[\ ]_{c'_2} [\ ]_{d'_1} \to [\ ]_{c'_3 c'_4 d'_1}$
Step 3	$[\ ]_{E_B H B X_l} \to [\ ]_{E_B H \# \#} [\ ]_{X_l}$	$[\ ]_{d_1 c_3 c_4} \to [\ ]_{d_2} [\ ]_{c_3 c_4}$	$[\ ]_{d'_1 c'_3 c'_4} \to [\ ]_{d'_2} [\ ]_{c'_3 c'_4}$
Step 4	$[\ ]_{H X_l E_B} [\ ]_{d_2} \to [\ ]_{H X_l E' d_2}$	$[\ ]_{c_4 c_3} [\ ]_{d'_2} \to [\ ]_{c_4 c_5 d'_2}$	$[\ ]_{c'_3 c'_4} \to [\ ]_{c'_5} [\ ]_{c'_4}$
Step 5	$[\ ]_{H E X_l E' d_2} \to [\ ]_{H Y^v} [\ ]_{E' d_2}$	$[\ ]_{d'_2 c_4 c_5} \to [\ ]_{c_6 c_7} [\ ]_{c_4 c_5}$	$[\ ]_{c'_4} [\ ]_{c'_5} \to [\ ]_{d'_1 c'_5}$
Step 6	$[\ ]_{Y^v H} \to [\ ]_{Y^{vi}} [\ ]_H$ $[\ ]_{E' d_2} \to [\ ]_E [\ ]_{d_2}$	$[\ ]_{c_6 c_7} \to [\ ]_{d_1} [\ ]_{c_7}$ $[\ ]_{c_4 c_5} \to [\ ]_{c_1} [\ ]_{c_5}$	$[\ ]_{c'_5 d'_1} \to [\ ]_{c'_6} [\ ]_{d'_1}$
Step 7	$[\ ]_{Y^{vi}} [\ ]_{d_2} \to [\ ]_{Y^{vii} d_2}$	$[\ ]_{c_5} [\ ]_{c_7} \to [\ ]_{c_8 c_7}$	$[\ ]_{c'_6} [\ ]_{d'_1} \to [\ ]_{c'_7 d'_1}$
Step 8	$[\ ]_{Y^{vii} d_2} [\ ]_H \to [\ ]_{Y^{vii} H}$	$[\ ]_{c_8 c_7} [\ ]_{d_1} \to [\ ]_{c_8 d_1}$	$[\ ]_{c'_7 d'_1} \to [\ ]_{c'_8} [\ ]_{d'_1}$
Step 9	$[\ ]_{H Y^{vii}} [\ ]_E \to [\ ]_{HYE}$	$[\ ]_{d_1 c_8} [\ ]_{c_1} \to [\ ]_{d_1 c_1}$	$[\ ]_{c'_8} [\ ]_{d'_1} \to [\ ]_{c'_1 d'_1}$

Table 2. Rules for simulating a matrix $m_l : (X \to Y, B \to \#) \in M$

then the rule $[\ ]_{XEH} \to [\ ]_{\#\#} [\ ]_{EH}$ must be used and the computation never stops.

Assume that we start with rule $[\ ]_{EHXA} \to [\ ]_{EHX_l} [\ ]_A$ corresponding to a matrix $m_l : (X \to Y, A \to x)$, hence with the second rule not to be applied in the appearance checking mode. The two membranes mate in the next step, and we obtain a membrane marked with $X_l EHx'$, where x' is the string obtained from x in $A \to x$, by priming one symbol if x is non-empty, or, if $x = \lambda$, then $x' = f'$. The evolution continues deterministically by using the rules from the second column of Table 1, correctly simulating the two rules of the matrix, and returning to a membrane again marked with EH and a symbol from N_1. The only delicate point is that in step 5 we can continue only if a membrane $[\ ]_{d_2}$ was made available by processing the membrane having initially the marking $c_1 d_1$.

The presence of this membrane is ensured by the interplay of the evolution of membranes which originate in $[\ ]_{c_1 d_1}$ and $[\ ]_{c'_1 d'_1}$. Let us examine the way of using the rules from the last two columns of Table 1. The first two steps are identical, modulo the primes of objects marking membranes evolving from $[\ ]_{c'_1 d'_1}$. In the third step we have a choice. If we use the rules $[\ ]_{d_1 c_3 c_4} \to [\ ]_{d_2} [\ ]_{c_3 c_4}$, $[\ ]_{d'_1 c'_3 c'_4} \to [\ ]_{d'_2} [\ ]_{c'_3 c'_4}$, then the continuation is correct – we get the necessary membrane marked with d_2, while in step 4 we can use the rule $[\ ]_{c_4 c_3} [\ ]_{d'_2} \to [\ ]_{c_4 c_5 d'_2}$. However, if this rule cannot be applied, then the rule $[\ ]_{c_3 c_4} \to [\ ]_{\#\#} [\ ]_{c_4}$ must be applied and the computation will never stop.

After having produced the membrane $[\ ]_{c_4 c_5 d'_2}$ we have again two possibilities. The one which does not lead to an endless computation is that using the rule $[\ ]_{d'_2 c_4 c_5} \to [\ ]_{c_6 c_7} [\ ]_{c_4 c_5}$, followed now deterministically by the rules from lines 6, 7, 8, 9 of Table 1, and ending with the membrane $[\ ]_{c_1 d_1}$. However, in step 5 we can use instead the rule $[\ ]_{c_4 c_5} \to [\ ]_{c_1} [\ ]_{c_5}$. The object d'_2 is either together with c_1 or together with c_5 on the same membrane, and this makes possible (and

obligatory, due to the maximality of the parallelism) the use of one of the rules $[\ ]_{c_1 d_2'} \to [\ ]_{\#\#}[\ ]_{d_2'}$, $[\ ]_{c_5 d_2'} \to [\ ]_{\#\#}[\ ]_{d_2'}$, making the computation never halt.

The evolution of membranes evolving from $[\ ]_{c_1' d_1'}$ from step 5 on is deterministic, and it ends by reproducing the membrane $[\ ]_{c_1' d_1'}$.

Consequently, the matrix m_l can be correctly simulated, and, if the computation in Π does not correctly simulate the matrix, then it never ends. After completing the correct simulation, the three starting membranes are reproduced, with similar markings: EHY appears on the first membrane, $c_1 d_1, c_1' d_1'$ on the other two.

The evolution of the configuration along the nine steps of (correctly) simulating the matrix $m_l : (X \to Y, A \to x)$ is indicated in Figure 1, where the maximal number of membranes present simultaneously in the system is nine (step 6).

Initial	$[\ [\ ]_{EHXA}\ [\ ]_{c_1 d_1}\ [\ ]_{c_1' d_1'}\ ]_\lambda$
Step 1	$[\ [\ ]_{EHX_l}\ [\ ]_A\ [\ ]_{c_2}\ [\ ]_{d_1}\ [\ ]_{c_2'}\ [\ ]_{d_1'}\ ]_\lambda$
Step 2	$[\ [\ ]_{x'EHX_l}\ [\ ]_{c_3 c_4 d_1}\ [\ ]_{c_3' c_4' d_1'}\ ]_\lambda$
Step 3	$[\ [\ ]_{Y'}\ [\ ]_{\alpha' EH}\ [\ ]_{d_2}\ [\ ]_{c_3 c_4}\ [\ ]_{d_2'}\ [\ ]_{c_3' c_4'}\ ]_\lambda$
Step 4	$[\ [\ ]_{EH\alpha Y'}\ [\ ]_{d_2}\ [\ ]_{c_4 c_5 d_2'}\ [\ ]_{c_5'}\ [\ ]_{c_4'}\ ]_\lambda$
Step 5	$[\ [\ ]_{EHY'' d_2}\ [\ ]_{c_6 c_7}\ [\ ]_{c_4 c_5}\ [\ ]_{c_5' d_1'}\ ]_\lambda$
Step 6	$[\ [\ ]_{Ed_2 Y''}\ [\ ]_H\ [\ ]_{d_1}\ [\ ]_{c_7}\ [\ ]_{c_1}\ [\ ]_{c_5}\ [\ ]_{c_6'}\ [\ ]_{d_1'}\ ]_\lambda$
Step 7	$[\ [\ ]_{Y''' EH}\ [\ ]_{c_8 c_7}\ [\ ]_{d_1}\ [\ ]_{c_1}\ [\ ]_{c_7' d_1'}\ ]_\lambda$
Step 8	$[\ [\ ]_{Y^{iv}}\ [\ ]_{EH}\ [\ ]_{c_8 d_1}\ [\ ]_{c_1}\ [\ ]_{c_8'}\ [\ ]_{d_1'}\ ]_\lambda$
Step 9	$[\ [\ ]_{EHY}\ [\ ]_{c_1 d_1}\ [\ ]_{c_1' d_1'}\ ]_\lambda$

Fig. 1. The evolution of membranes when simulating $m_l : (X \to Y, A \to x)$

Consider now the case when we start with the rule $[\ ]_{XEH} \to [\ ]_{X_l}[\ ]_{EH}$, hence we intend to simulate a matrix $m_l : (X \to Y, B \to \#)$. In the next step the two membranes mate, and we get $[\ ]_{E_B HX_i}$; note that all symbols from $N \cup V$ from the current sentential form of G are together on this membrane (they have been non-deterministically separated by using the first rule, but now they come back). This is important in view of simulating the rule $B \to \#$ from the matrix m_l: if any copy of B is present, then the rule $[\ ]_{E_B HBX_l} \to [\ ]_{E_B H\#\#}[\ ]_{X_l}$ must be applied, and the computation never stops. If no B is present, then this rule is not applied, the membrane remains unchanged. Simultaneously, the second membrane, the one marked with $[\ ]_{d_1 c_3 c_4}$, produces the membrane $[\ ]_{d_2}$, which makes possible the continuation of the evolution of the first membrane, by using the rule $[\ ]_{HX_l E_B}[\ ]_{d_2} \to [\ ]_{HX_l E' d_2}$.

In step 5 we can use the rule $[\ ]_{HX_l E' d_2} \to [\ ]_{HY^v}[\ ]_{E' d_2}$, and then the continuation proceeds deterministically, by using the rules from rows 6, 7, 8, 9 of Table

2. In the end, we obtain a membrane with the marking EHY, hence correct with respect to G: we have checked that B is not present, and X was changed with Y.

However, in step 5 we can also use the rule $[\ ]_{E'd_2} \to [\ ]_E[\ ]_{d_2}$, and this leas to two membranes $[\ ]_{Ez_1}\ [\ ]_{d_2z_2}$ with $z_1z_2 \in \{HX_l\}(V \cup N_2)^+$. The only rule which can be applied to these membranes is $[\ ]_E[\ ]_{d_2} \to [\ ]_{\#\#d_2}$, hence the computation never stops.

Let us also note that the rules from the second column of Table 1 cannot interfere with the rules used for simulating a matrix $m_l : (X \to Y, B \to \#)$, and, conversely, rules from the second column of Table 2 cannot be used when simulating a matrix $m_l : (X \to Y, A \to x)$; this is ensured by the different priming of symbols Y and E, and by the fact that the matrices of M are injectively labelled.

It is important to remember from the previous discussion, that the membrane marked with d_2 is produced exactly in step 3, and it is indeed produced (otherwise the second membrane introduces the trap-object); moreover, there is no d_2 available in the system from the simulation of a previous matrix, of any type (with or without rules to be used in the appearance checking manner), because, as we have seen above, the membrane $[\ ]_{d_2}$ is "consumed" during the simulation of the matrices.

We return again to three membranes with markings as in the beginning. The evolution of the membrane structure during the nine steps is indicated in Figure 2 (this time, the maximal number of membranes present simultaneously is eleven, in step 6).

Initial	$[\ [\ ]_{EHX}\ [\ ]_{c_1d_1}\ [\ ]_{c'_1d'_1}\ ]_\lambda$
Step 1	$[\ [\ ]_{X_l}\ [\ ]_{EH}[\ ]_{c_2}\ [\ ]_{d_1}\ [\ ]_{c'_2}\ [\ ]_{d'_1}\ ]_\lambda$
Step 2	$[\ [\ ]_{HE_BX_l}\ [\ ]_{c_3c_4d_1}\ [\ ]_{c'_3c'_4d'_1}\ ]_\lambda$
Step 3	$[\ [\ ]_{HE_BX_l}\ [\ ]_{d_2}\ [\ ]_{c_3c_4}\ [\ ]_{d'_2}\ [\ ]_{c'_3c'_4}\ ]_\lambda$
Step 4	$[\ [\ ]_{HX_lE'd_2}\ [\ ]_{c_4c_5d'_2}\ [\ ]_{c'_5}\ [\ ]_{c'_4}\ ]_\lambda$
Step 5	$[\ [\ ]_{HY^v}\ [\ ]_{E'd_2}\ [\ ]_{c_6c_7}\ [\ ]_{c_4c_5}\ [\ ]_{c'_5d'_1}\ ]_\lambda$
Step 6	$[\ [\ ]_{Y^{vii}}\ [\ ]_H\ [\ ]_E\ [\ ]_{d_2}\ [\ ]_{d_1}\ [\ ]_{c_7}\ [\ ]_{c_1}\ [\ ]_{c_5}\ [\ ]_{c'_6}\ [\ ]_{d'_1}\ ]_\lambda$
Step 7	$[\ [\ ]_{Y^{vii}d_2}\ [\ ]_H\ [\ ]_E\ [\ ]_{c_8c_7}\ [\ ]_{d_1}\ [\ ]_{c_1}\ [\ ]_{c'_7d'_1}\ ]_\lambda$
Step 8	$[\ [\ ]_{Y^{vii}H}\ [\ ]_E\ [\ ]_{c_8d_1}\ [\ ]_{c_1}\ [\ ]_{c'_8}\ [\ ]_{d'_1}\ ]_\lambda$
Step 9	$[\ [\ ]_{EHY}\ [\ ]_{c_1d_1}\ [\ ]_{c'_1d'_1}\ ]_\lambda$

Fig. 2. The evolution of membranes when simulating $m_l : (X \to Y, B \to \#)$

In both cases, the simulation of matrices is correct, and the process can be iterated. We continue in this way until reaching a configuration with three inner membranes marked as follows: $[\ ]_{wZ_aEH}$, $[\ ]_{c_1d_1}$, $[\ ]_{c'_1d'_1}$. In order to conclude the computation, we have to mate all membranes in a single one, to remove all auxiliary objects, and to check whether any symbol # is present, a case where the computation should not halt. These operations are done by means of the following

rules:

$$[\ ]_{Z_aHE}[\ ]_{c_1d_1} \to [\ ]_{Z_aHc_1d_1},\ [\ ]_{Z_aEH} \to [\ ]_{\#\#}[\ ]_{EH}.$$

The first membrane starts to "swallow" the other membranes, and this is the only way to halt, otherwise the second rule above is used and the computation never stops. In the same step, the rule $[\ ]_{c'_1d'_1} \to [\ ]_{c'_2}[\ ]_{d'_1}$ is used. The membrane $[\ ]_{c'_2}$ is "eaten" by the first membrane, by means of the rule

$$[\ ]_{Z_aHd_1c_1}[\ ]_{c'_2} \to [\ ]_{Z_aHd_1c'_2},$$

which is necessarily used, otherwise we use the rule

$$[\ ]_{Z_aHc_1d_1} \to [\ ]_{\#\#}[\ ]_{Hc_1d_1},$$

and the computation never stops. We continue by means of the following rules:

$$\begin{aligned}
&[\ ]_{Z_aHd_1c'_2}[\ ]_{d'_1} \to [\ ]_{Z_aHd_1d'_1},\\
&[\ ]_{Z_aHd_1d'_1} \to [\ ]_{Z_a}[\ ]_{d_1d'_1},\\
&[\ ]_{d'_1d_1}[\ ]_{Z_a} \to [\ ]_{d'_1Z_a},\\
&[\ ]_{Z_ad'_1} \to [\ ]_{a''}[\ ]_{d'_1},\\
&[\ ]_{d'_1}[\ ]_{a''} \to [\ ]_{d''_1a''},\\
&[\ ]_{a''d''_1} \to [\ ]_{a}[\ ]_{d''_1},\\
&[\ ]_{d''_1}[\ ]_{a} \to [\ ]_{a},
\end{aligned}$$

for all $a \in V$.

If the membrane $[\ ]_{Z_aHd_1d'_1}$ obtained after using the first rule above evolves (prematurely) by means of the rule $[\ ]_{Z_ad'_1} \to [\ ]_{a''}[\ ]_{d'_1}$, then we obtain two membranes $[\ ]_{w_1a''}[\ ]_{w_2d'_1}$ with $w_1w_2 = Hd_1$, but they mate by means of the rule $[\ ]_{d'_1}[\ ]_{a''} \to [\ ]_{d''_1a''}$, thus bringing together the multisets w_1, w_2. This means that Hd_1 mark the same membrane, hence we can use the rule

$$[\ ]_{Hd_1} \to [\ ]_{\#\#}[\ ]_{d_1}$$

in order to prevent this path (no other rule can be applied instead).

The evolution of the membrane structure in the end of a computation is indicated in Figure 3.

Therefore, we end with only one membrane, marked, like any string generated by G after introducing the symbol Z_a, by symbols from V and, possibly, #. In order to make the computation continue forever in this latter case, we also introduce the rule

$$[\ ]_{a\#} \to [\ ]_{\#}[\ ]_{\#},\ a \in V.$$

(Note that there is at least one symbol $a \in V$ present, the one introduced from the symbol Z_a.)

Initial	$[\ [\ ]_{EHZ_aw}\ [\ ]_{c_1d_1}\ [\ ]_{c_1'd_1'}\ ]_\lambda$
Step 1	$[\ [\ ]_{Z_aHc_1d_1}\ [\ ]_{c_2'}\ [\ ]_{d_1'}\ ]_\lambda$
Step 2	$[\ [\ ]_{Z_aHd_1c_2'}\ [\ ]_{d_1'}\ ]_\lambda$
Step 3	$[\ [\ ]_{Z_aHd_1d_1'}\ ]_\lambda$
Step 4	$[\ [\ ]_{Z_a}\ [\ ]_{d_1d_1'}\ ]_\lambda$
Step 5	$[\ [\ ]_{d_1'Z_a}\ ]_\lambda$
Step 6	$[\ [\ ]_{a''}\ [\ ]_{d_1'}\ ]_\lambda$
Step 7	$[\ [\ ]_{d_1''a''}\ ]_\lambda$
Step 8	$[\ [\ ]_{d_1''}\ [\ ]_a\ ]_\lambda$
Step 9	$[\ [\ ]_{wa}\ ]_\lambda$

Fig. 3. The evolution of membranes in the end of a computation

The previous analysis shows that $\Psi_V(L(G)) = Ps(\Pi)$. Because the number of membranes and the weight of rules is as stated in the theorem, this concludes the proof.

It is interesting to note that if we start from a matrix grammar without appearance checking, then the auxiliary membranes marked with c_1d_1 and $c_1'd_1'$ are no longer necessary; the system simplified in this way uses one rule at a time, with the rules precisely chained, that is, we can apply them in an unsynchronized manner and still we generate the Parikh image of the starting language. Therefore, by unsynchronized systems we generate at least $PsMAT$.

7 Final Remarks

The result above confirms a statement made several times in membrane computing: the cell is a very powerful "computer". This time, the Turing completeness was obtained in the rather restrictive framework of using only the *mate, drip* operations, with proteins placed on membranes and controlling the operations.

Several directions for continuing the present research are of interest. First, the use of the *pino, exo* operations should be investigated. Similarly, other operations from brane calculi could/should be considered, such as *phago* and *bud*, and the power of P systems using also such operations should be examined. Then, improvements of the previous results, in the number of membranes or the weight of rules could be sought for. Of a possible practical relevance from the point of view of systems biology is to find classes of operations which can lead to systems with decidable properties (hence sub-universal). Removing the restriction of maximal parallelism might lead to such a decrease of the power – but, as seen in the end of Section 6, with rules as above we can still generate all Parikh images of languages generated by matrix grammars without appearance checking, which, intuitively speaking, is not a small family (it equals the family of vectors accepted by partially blind register machines of [5]).

Acknowledgements

In some sense, this paper was catalyzed by Gabriel Ciobanu, during the Third Brainstorming Week on Membrane Computing, Sevilla, February 2005, by discussions with both authors about the possibility to bridge brane calculi and membrane computing.

References

1. N. Busi, R. Gorrieri: On the computational power of brane calculi. *Third Workshop on Computational Methods in Systems Biology*, Edinburgh, 2005.
2. L. Cardelli: Brane calculi. Interactions of biological membranes. In *Proc. Computational Methods in Systems Biology, 2004*, Springer, Berlin, to appear.
3. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.
4. J. Dassow, Gh. Păun, A. Salomaa: Grammars with controlled derivations. Chapter 3 in vol. 2 of *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, Berlin, 1997, 101–154.
5. S.A. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7 (1978), 311–324.
6. M. Harrison: *Introduction to Formal Language Theory*. Addison-Wesley, Reading, MA, 1978.
7. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (and Turku Center for Computer Science–TUCS Report 208, November 1998, `www.tucs.fi`).
8. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
9. A. Salomaa: *Formal Languages*: Academic Press, New York, 1973.
10. The membrane computing web page: `http://psystems.disco.unimib.it`.