Two Universality Results for (Mem)Brane Systems

Daniela Besozzi[1], Nadia Busi[2], Giuditta Franco[3], Rudolf Freund[4], Gheorghe Păun[5]

[1] Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
Via Comelico 39, 20135 Milano, Italy
`besozzi@dico.unimi.it`
[2] Dipartimento di Scienze dell'Informazione, Università di Bologna
Mura A. Zamboni 7, 40127 Bologna, Italy
`busi@cs.unibo.it`
[3] Department of Computer Science, University of Verona
strada Le Grazie, 15, 37134 Verona, Italy
`franco@sci.univr.it`
[4] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9-11, 1040 Vienna, Austria
`rudi@emcc.at`
[5] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania, and
Department of Computer Science and AI, University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
`george.paun@imar.ro, gpaun@us.es`

Summary. We prove that P systems with *mate* and *drip* operations and using at most five membranes during any step of a computation are universal. This improves a recent similar result from [5], where eleven membranes are used. The proof of this result has the "drawback" that the output of a computation is obtained on an inner membrane of the system. A universality proof is then given for the case when the result of a computation is found on the skin membrane (on its external side, hence "visible" from the environment), but in this case we use one more membrane, as well as another basic brane operation *exo*; moreover, the operations are now of the projective type, as introduced in [1].

1 Introduction

Membrane computing [9] and brane calculi [4] are two research areas starting from the same reality, the biology of the cell. They were developed initially in different directions, with different goals, and different tools, but in the last time started to increasingly interact, importing from each other basic ingredients, such as using objects placed in/on membranes, operations with membranes, ways of using the rules – synchronously or not, etc. We refer the reader to the recent papers [3], [5], [6], [8], [10] for details.

This paper is a further contribution to this effort, of bridging brane calculi and membrane computing, making use of the natural idea of considering P systems based on brane operations. It should be noted that membrane computing has considered already many operations with membranes, but in an idealized framework and under the control of objects placed *inside* the compartments defined by membranes. This last aspect is an essential difference with respect to brane calculi, where one adopts the other extreme position, taking into account only the objects ("proteins") placed *on* membranes. In particular, different classes of proteins associated to the cellular membranes can be considered: the *peripheral* proteins, which are linked only to the external or to the internal side of the membrane, and *integral* (or transmembrane) proteins, which span the cellular bilayer and thus have part of their molecule on either sides of the membrane.

This peculiar differences of membrane proteins inspired to investigate not the case of usual brane operations (already considered in [5]) but their *projective* version, proposed in [1], with the proteins placed *on one of the sides of membranes*, hence visible/active only inside or only outside, respectively, representing either peripheral proteins, either the external or internal active site of integral proteins.

Specifically, we start from the four basic operations introduced in [4], *mate, drip, pino, exo*, and we consider them as operations in a P system – hence used in a maximally parallel manner, with the system itself used for computing a set of numbers. However, a simple examination of the proof of the universality result from [5] shows that this proof works also in the projective case, just assuming that all proteins are placed on the external side of the membranes. This is due to the fact that one works with membrane structures with two levels, and only the inner membranes play a role in the computation, the external one never interacts with the other membranes; in particular, the inner membranes have no other membranes inside.

Still, the construction from [5] has a "practical drawback": the result of a computation is placed in the halting configuration on the inner membrane, which means that we have to "break the computer" in order to read the result. We try here to do better, and we succeeded to have the result of a computation on the skin membrane, even on its external side, the one visible from the environment. This was achieved at the price of using three operations *mate, drip, exo* (of course, of the projective type, in order to move proteins from the internal membranes to the external side of the skin membrane, as a biologically well motivated process which happens in living cells due to the bilayer structure of membranes).

Actually, this result follows from a new proof of the result from [5]. This proof is based on the simulation of register machines, and it improves the proof from [5], where eleven membranes were necessary: here we use only five. In the projective case, when reading the result on the external side of the skin membrane, we need one additional membrane, hence the universality is obtained for six membranes (with an additional price paid in the fact that number 1 is ignored).

Of course, the problem remains open whether or not the number of membranes can be further decreased, and, perhaps more interesting, to investigate other combinations of rules (maybe non-universal, so that decidable properties can be found, of possible interest for computational biology).

2 Prerequisites

We assume the reader to be familiar with basic language and automata theory (e.g., from [11]), as well as with basics of membrane computing (e.g., from [9] – we also refer to [12] for current information in this area), so that we introduce here only some notations and the notion of register machines, used later in proofs.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V; the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. The length of a string $x \in V^*$ is denoted by $|x|$.

The family of Turing computable sets of natural numbers is denoted by NRE (this is the family of length sets of recursively enumerable languages, hence the notation).

A *register machine* is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction HALT), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\mathtt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : \mathtt{HALT}$ (the halt instruction).

A register machine M computes a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known (see, e.g., [7]) that register machines (even with a small number of registers, but this detail is not relevant here) compute all sets of numbers which are Turing computable, hence they characterize NRE.

Without loss of generality, we may assume that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation, we only add to its contents. In the proofs from the next sections we will always assume that the register machines which we simulate have these properties.

In the following sections, when comparing the power of two number computing devices the number zero is ignored. Thus, when saying that a set Q is in NRE, we do not care whether or not $0 \in Q$. (This is similar to the usual convention in formal language theory to ignore the empty string when comparing the power of two generating/accepting devices.)

Moreover, we consider the family $1NRE$, of sets $Q \in NRE$ which do not contain number 1 (and, of course, neither number 0).

3 P Systems Using the Mate, Drip Operations

As we have mentioned in the Introduction, in brane calculi there were introduced several operations with membranes, but in the universality result given in the next section we work only with *mate* and *drip*, that is why we only define here these operations. One more operation will be defined in Section 5; we also refer to [5] for further details.

As usual in membrane computing, we represent a membrane by a pair of square brackets, $[\ ]$, maybe with labels for identifying them. With membranes we associate *multisets of proteins* (we also use to speak about "objects" instead of "proteins"), which are supposed here to be placed *on* membranes (like in brane calculi), and visible/accessible from both sides of them. We briefly remark that a different idea would consist in defining the set of proteins u associated to a membrane as a *string* instead of a multiset, thus a formal structure where each protein have a specific mutual position with respect to all other proteins in the string. From a mathematical point of view, this would introduce different methods to control and apply operations; from a biological point of view, this representation is inspired from the role played by specific membrane proteins in different places of the membranes (for instance, the non-casual distribution – clustering – of receptors matters in receptor-mediated endocytosis at the plasma membrane). Anyway, this topic will be investigated in a forthcoming paper and, in the following, we will only consider the case of multisets of proteins.

A membrane having associated a multiset u of proteins (we represent multisets by strings, with the obvious observation that all permutations of a string represent the same multiset) is written in the form $[\ ]_u$; we also use to say that the membrane is *marked* with the multiset u.

With this notation, the two operations are written as follows:

$$\begin{aligned} mate &: [\ \ ]_{ua}[\ \ ]_{v} \rightarrow [\ \ ]_{uxv}, \\ drip &: [\ \ ]_{uav} \rightarrow [\ \ ]_{ux}[\ \ ]_{v}, \end{aligned}$$

in all cases with $a \in V$, $u, x \in V^*$, $v \in V^+$, $ux \in V^+$ for *drip* rules, for a specified alphabet V of proteins. (We imposed here the restriction that the strings ux, v

are non-empty only for uniformity with [5], thus making our result from the next section comparable with that from [5], but this restriction can be removed, e.g., allowing $v = \lambda$. Another possible change in this definition is to consider one more "active" protein, b, placed on the membrane where v is placed, and which is transformed together with a into multiset x when applying the operation. For instance, the *mate* operation would look like $[\ \ ]_{ua}[\ \ ]_{bv} \rightarrow [\ \ ]_{uxv}$, with inspiration from the biological process of recognition that take place between "complementary" proteins lying on the membranes which mate together (for instance, the process of specific targeting which allows vesicular transport among donor and target compartments inside the cell. This would add more possibilities to control our operations, but we do not introduce this extension here.)

The length of the string uav (hence the total multiplicity of the multiset represented by this string) from each rule is called the *weight* of the rule (thus, we work here only with rules of weight greater than or equal to two).

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). It is important to note 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 drip rule is precisely split into ux and v, with these two multisets assigned to the two new membranes.

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 *drip*, they are randomly distributed to the two resulting membranes.

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 possibly only other membranes. In the case of the *mate* operation, the contents of the starting membranes are put together in the resulting membrane; in the case of *drip* operation, one membrane is empty, while the contents of the initial membrane is placed in the membrane which inherits the multiset v, as suggested below:

$$\mathit{drip}\ :\ [\ \mathbf{Q}\ ]_{uav} \rightarrow [\ \ ]_{ux}[\ \mathbf{Q}\ ]_{v}.$$

Rules as above can be used in a P system of the form

$$\Pi = (A, \mu, u_0, u_1, \ldots, u_m, R),$$

where:

1. A is an alphabet (finite, non-empty) of proteins;
2. μ is a membrane structure with at least two membranes (hence $m \geq 1$);
3. $u_1, \ldots, u_m$ are multisets of proteins (represented by strings over A) bound to the m inner 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 $u_1, \dots, u_m$ precisely defined; the labels play no other role than specifying this initial marking of membranes); the skin membrane is labeled with 0 and $u_0 = \lambda$;

4. R is a finite set of *mate, drip* rules, of the forms specified above, using proteins from the set A.

Note that the skin membrane has no protein associated, because it cannot enter any rule, it is only meant to delimit the system from its environment.

When using any rule of any type, the membranes from its left hand side 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. All other proteins which mark the membranes which are consumed remain unchanged, and they are transferred to the newly created membranes. In the case of *mate* all proteins are placed on the new membrane; in the case of *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 proceeds through *transitions* among *configurations*, based on the *non-deterministic maximally parallel* use of rules. In each step, each membrane and each protein can be involved in only one rule. 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 $u_0, u_1, \dots, u_m$. In each step (a global clock is assumed to exist), we choose non-deterministically and apply in a parallel manner a maximal set of rules which can be applied to the current configuration. A membrane remains unchanged if no rule is applied to it. The skin membrane never evolves.

A sequence of transitions constitutes a *computation.* A computation which starts from the initial configuration is *successful* only 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 the empty multiset) and an inner one. The *result* of a successful computation is the number of proteins which mark the inner membrane in the halting configuration. (We can also take as the result of a computation the vector which describes the multiplicity of objects placed on the inner membrane in the halting configuration – this was the case considered in [5]; the proof from the next section remains valid also for this generalization.)

The set of all numbers computed in this way by Π is denoted by $N(\Pi)$. The family of all sets $N(\Pi)$ 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 p, q, respectively, is denoted by $NOP_m(mate_p, drip_q)$. When one of the parameters m, p, q is not bounded, we replace it with $*$.

4 Universality for the Mate/Drip Case

In [5] it is proved that $NRE = NOP_m(mate_p, drip_q)$ for all $m \geq 11, p \geq 5$, and $q \geq 5$. We improve here this result for each of the three subscripts.

Theorem 1. *$NRE = NOP_m(mate_p, drip_q)$ for all $m \geq 5$, $p \geq 4$, and $q \geq 4$.*

Proof. We only have to prove the inclusion $NRE \subseteq NOP_5(mate_4, drip_4)$, and to this aim we use the characterization of NRE by means of register machines. Let $M = (m, H, l_0, l_h, I)$ be a register machine as in Section 2. We construct the P system

$$\Pi = (A, \mu, u_0, u_1, u_2, R),$$

with

$$\begin{aligned} A &= \{a_r \mid 1 \leq r \leq m\} \cup \{l, l', l'' \mid l \in H\} \\ &\quad \cup \{d_i \mid 1 \leq i \leq 8\} \cup \{b, c, h, h', s, \#\}, \\ \mu &= [[\ ]_1[\ ]_2]_0, \\ u_0 &= \lambda,\ u_1 = hl_0,\ u_2 = bc, \end{aligned}$$

and the set R of rules constructed as follows.

The computations of Π start with one inner membrane marked with the label l_0 corresponding to the initial instruction of M, together with the helping object h, and the second inner membrane marked with the auxiliary (control) objects b and c. The contents of a register r of M will be represented by the number of copies of object a_r present in the system (on membranes produced from $[\ ]_{hl_0}$).

Both ADD and SUB instructions of M are simulated in eight steps of a computation in Π, with the membranes produced from $[\ ]_{bc}$ evolving in the same way in the two cases. It is also important to note that the contents of registers, represented by copies of objects a_r, $1 \leq r \leq m$, and placed on the membranes which are produced from $[\ ]_{hl_0}$, are separated on at most two membranes, which immediately produce a unique membrane by means of a *mate* operation. Hence these objects are not "lost" in the system, they stay together on one or at most two membranes.

Because the simulation of a SUB instruction is more complex, and it fixes the number of steps mentioned above, eight, we start by presenting the way of handling such an instruction, $l_i : (\mathtt{SUB}(r), l_j, l_k)$. One uses the rules from Table 1 (in the configurations presented in the right hand column we do not specify the other objects present on the respective membranes, but only those involved in the rules from the left hand column).

Note that, if a copy of the symbol a_r exists (that is, the register r is not empty), then it is removed by rule 4 and the computation continues with the symbol l'_i (corresponding to the l_k instruction); if a_r does not exist, then the computation continues with the symbol l''_i (corresponding to the l_j instruction); in both cases the configuration $[\ [\ ]_{hl^?_i s}[\ ]_{d_7}[\ ]_{d_5 d_6}[\ ]_c\ ]_\lambda$ is reached after step 5.

Step	Rules	Types	Configuration
initial			$[\,[\ ]_{hl_i}[\ ]_{bc}\,]_\lambda$
1	$[\ ]_{bc} \to [\ ]_{d_1d_2d_3}[\ ]_c$	drip	$[\,[\ ]_{hl_i}[\ ]_{d_1d_2d_3}[\ ]_c\,]_\lambda$
2	$[\ ]_{d_1d_2d_3} \to [\ ]_s[\ ]_{d_2d_3}$	drip	$[\,[\ ]_{hl_i}[\ ]_s[\ ]_{d_2d_3}[\ ]_c\,]_\lambda$
3	$[\ ]_{hl_i}[\ ]_s \to [\ ]_{hl'_is}$	mate	$[\,[\ ]_{hl'_is}[\ ]_{d_4}[\ ]_{d_3}[\ ]_c\,]_\lambda$
	$[\ ]_{d_2d_3} \to [\ ]_{d_4}[\ ]_{d_3}$	drip	
4	$[\ ]_{l'_ia_rhs} \to [\ ]_{l'_i}[\ ]_{hs}$	drip	$[\,[\ ]_{l'_i}[\ ]_{hs}[\ ]_{d_5d_6d_3}[\ ]_c\,]_\lambda$
	$[\ ]_{d_4}[\ ]_{d_3} \to [\ ]_{d_5d_6d_3}$	mate	or $[\,[\ ]_{l'_ish}[\ ]_{d_5d_6d_3}[\ ]_c\,]_\lambda$
5	$[\ ]_{l'_i}[\ ]_{sh} \to [\ ]_{l''_ish}$	mate	$[\,[\ ]_{hl^?_is}[\ ]_{d_7}[\ ]_{d_5d_6}[\ ]_c\,]_\lambda$
	$[\ ]_{d_3d_5d_6} \to [\ ]_{d_7}[\ ]_{d_5d_6}$	drip	
6	$[\ ]_{hl^?_is}[\ ]_{d_7} \to [\ ]_{hl_\alpha sd_7}$	mate	$[\,[\ ]_{hl_\alpha sd_7}[\ ]_{d_8}[\ ]_{d_5}[\ ]_c\,]_\lambda$
	$[\ ]_{d_6d_5} \to [\ ]_{d_8}[\ ]_{d_5}$	drip	
7	$[\ ]_{hsl_\alpha d_7} \to [\ ]_h[\ ]_{l_\alpha d_7}$	drip	$[\,[\ ]_h[\ ]_{l_\alpha d_7}[\ ]_{bc}[\ ]_{d_5}\,]_\lambda$
	$[\ ]_{d_8}[\ ]_c \to [\ ]_{bc}$	mate	
8	$[\ ]_{l_\alpha d_7}[\ ]_h \to [\ ]_{l_\alpha h}$	mate	$[\,[\ ]_{l_\alpha h}[\ ]_{bc}\,]_\lambda$
	$[\ ]_{bc}[\ ]_{d_5} \to [\ ]_{bc}$	mate	

Table 1. Rules simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$; $l^?_i \in \{l'_i, l''_i\}$, if $l^?_i = l''_i$, then $l_\alpha = l_j$, and if $l^?_i = l'_i$, then $l_\alpha = l_k$.

Due to the subscripts of objects $d_i, 1 \leq i \leq 8$, the simulation proceeds almost deterministically: in each step there is only one choice of rules to apply, with the only three exceptions being the possibilities of using (i) the rule $[\ ]_{d_2d_3} \to [\ ]_{d_4}[\ ]_{d_3}$ already in step 2, and not in step 3 as indicated in Table 1, (ii) the rule $[\ ]_{d_6d_5} \to [\ ]_{d_8}[\ ]_{d_5}$ already in step 5, and not in step 6 as indicated in Table 1, and (iii) the rule $[\ ]_{bc} \to [\ ]_{d_1d_2d_3}[\ ]_c$ in step 8 instead of the rule $[\ ]_{bc}[\ ]_{d_5} \to [\ ]_{bc}$.

In order to prevent the halting of the computation in these cases, we introduce in R the additional rules:

1. $[\ ]_h[\ ]_{d_5} \to [\ ]_{\#\#d_5}$,
2. $[\ ]_{\#\#} \to [\ ]_\#[\ ]_\#$,
3. $[\ ]_\#[\ ]_\# \to [\ ]_{\#\#}$.

In fact, there are the following three different computations to consider as alternative to that outlined in Table 1.

In all the three cases (i), (ii), (iii), when we do not follow the rules from Table 1, at some moment a membrane marked (among others) with h and a membrane marked (among others) with d_5 have to mate by means of the first additional rule above. In this way, the trap object # is introduced, and the computation continues forever by means of rules 2, 3. For instance, assume that in step 2 we use the rule $[\ ]_{d_2d_3} \to [\ ]_{d_4}[\ ]_{d_3}$. This means that we get the configuration $[[\ ]_{hl_i}[\ ]_{d_4}[\ ]_{d_3}[\ ]_c]_\lambda$ (with d_1 placed on one of the membranes marked with d_3 and d_4). We continue to $[[\ ]_{hl_i}[\ ]_{d_5d_6d_3d_1}[\ ]_c]_\lambda$. Now we either use the rule from step 5 or the one from step 6 for handling the membrane $[\ ]_{d_5d_6d_3d_1}$; in the first case we separate d_5 from d_6 in one further step, also introducing d_8 instead of d_6, in the next step d_5 and h are not used by other rules than rule 1 mentioned above, and the computation never halts. Similar results are obtained also in other cases.

Therefore, if we do not use exactly the rules indicated in Table 1 for the eight steps of the simulation, then the computation never halts. If we use the rules as indicated, then the simulation of the SUB instruction is correctly completed (one copy of a_r was removed in step 4, if this was possible, and, depending on this fact, we introduce the correct label-object l_j or l_k), and we return to a configuration as the one we have started with, hence with two inner membranes, one marked with hl_α, for $\alpha \in \{l_j, l_k\}$, and one marked with bc.

As said before, the simulation of an ADD instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is very much similar to the simulation of a SUB instruction, with many rules being used in common. For the sake of readability, we give in Table 2 the rules used in the eight steps of the simulation.

Step	Rules	Types	Configuration
initial			$[\ [\ ]_{hl_i}[\ ]_{bc}\ ]_\lambda$
1	$[\ ]_{bc} \to [\ ]_{d_1d_2d_3}[\ ]_c$	drip	$[\ [\ ]_{hl_i}[\ ]_{d_1d_2d_3}[\ ]_c\ ]_\lambda$
2	$[\ ]_{d_1d_2d_3} \to [\ ]_s[\ ]_{d_2d_3}$	drip	$[\ [\ ]_{hl_i}[\ ]_s[\ ]_{d_2d_3}[\ ]_c\ ]_\lambda$
3	$[\ ]_{hl_i}[\ ]_s \to [\ ]_{hl'_ia_rs}$ $[\ ]_{d_2d_3} \to [\ ]_{d_4}[\ ]_{d_3}$	mate drip	$[\ [\ ]_{hl'_ia_rs}[\ ]_{d_4}[\ ]_{d_3}[\ ]_c\ ]_\lambda$
4	$[\ ]_{l'_ihs} \to [\ ]_{l'_i}[\ ]_{hs}$ $[\ ]_{d_4}[\ ]_{d_3} \to [\ ]_{d_5d_6d_3}$	drip mate	$[\ [\ ]_{l'_i}[\ ]_{hs}[\ ]_{d_5d_6d_3}[\ ]_c\ ]_\lambda$
5	$[\ ]_{l'_i}[\ ]_{sh} \to [\ ]_{l''_ish}$ $[\ ]_{d_3d_5d_6} \to [\ ]_{d_7}[\ ]_{d_5d_6}$	mate drip	$[\ [\ ]_{hl''_is}[\ ]_{d_7}[\ ]_{d_5d_6}[\ ]_c\ ]_\lambda$
6	$[\ ]_{hl''_is}[\ ]_{d_7} \to [\ ]_{hl_\alpha sd_7}$ $[\ ]_{d_6d_5} \to [\ ]_{d_8}[\ ]_{d_5}$	mate drip	$[\ [\ ]_{hl_\alpha sd_7}[\ ]_{d_8}[\ ]_{d_5}[\ ]_c\ ]_\lambda$
7	$[\ ]_{hsl_\alpha d_7} \to [\ ]_h[\ ]_{l_\alpha d_7}$ $[\ ]_{d_8}[\ ]_c \to [\ ]_{bc}$	drip mate	$[\ [\ ]_h[\ ]_{l_\alpha d_7}[\ ]_{bc}[\ ]_{d_5}\ ]_\lambda$
8	$[\ ]_{l_\alpha d_7}[\ ]_h \to [\ ]_{l_\alpha h}$ $[\ ]_{bc}[\ ]_{d_5} \to [\ ]_{bc}$	mate mate	$[\ [\ ]_{l_\alpha h}[\ ]_{bc}\ ]_\lambda$

Table 2. Rules simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$; $l_\alpha \in \{l_j, l_k\}$.

In step 3 we already introduce a copy of a_r, as required by the ADD instruction l_i. In step 4 we use a rule for the first inner membrane, and from step 5 we continue with l''_i present in the configuration; note that this passage is redundant (we could continue the computation with l'_i), but we proceed in this way for uniformity with the simulation of the SUB instruction (the rest of the computation is just the same for both the instructions). In step 6 one of the label-objects l_j and l_k is introduced non-deterministically. After 8 steps we return at a configuration similar to the starting one. The non-determinism in choosing the rules for steps 2, 5, and 8 is treated in the same way as for SUB instructions.

The simulation of ADD and SUB instructions is repeated. It is important to note that because the label l_i precisely identifies the instruction, one cannot mix other rules than those common in Tables 1 and 2 when simulating different instructions.

The computation in Π halts if and only if the computation in M halts. However, in order to have a successful computation in Π we need to keep only one membrane inside the system and to remove all other objects than a_1, the object whose multiplicity represents the contents of register 1 of M. To this aim, we use the rules from Table 3. The steps of this phase proceed deterministically (we are assuming that the numerical result of a halting computation is not 0, that is, that there exists at least one copy of a_1 in the system), hence the computation halts correctly.

Step	Rules	Types	Configuration
initial			$[\ [\ ]_{hl_h}[\ ]_{bc}\ ]_\lambda$
1	$[\ ]_{hl_h}[\ ]_{bc} \rightarrow [\ ]_{hl_h b}$	mate	$[\ [\ ]_{hl_h b}\ ]_\lambda$
2	$[\ ]_{l_h hb} \rightarrow [\ ]_{l_h h'}[\ ]_b$	drip	$[\ [\ ]_{l_h h'}[\ ]_b\ ]_\lambda$
3	$[\ ]_{h'l_h}[\ ]_b \rightarrow [\ ]_{h'b}$	mate	$[\ [\ ]_{h'b}\ ]_\lambda$
4	$[\ ]_{h'ba_1} \rightarrow [\ ]_{h'}[\ ]_{a_1}$	drip	$[\ [\ ]_{h'}[\ ]_{a_1}\ ]_\lambda$
5	$[\ ]_{h'}[\ ]_{a_1} \rightarrow [\ ]_{a_1}$	mate	$[\ [\ ]_{a_1}\ ]_\lambda$

Table 3. Rules ending the computation.

The observation that the maximum number of membranes used is 5 (after step 2 of simulating ADD and SUB instructions), and that we never use more than 4 objects in controlling the rules (in steps 4, 6, 7 from Table 1, in steps 6, 7 from Table 2, and in step 1 from Table 3), concludes the proof. □

We do not know whether the previous result is optimal in what concerns the number of membranes and the weight of the used rules.

5 P Systems with Projective Operations

Let us now pass to the projective version of the brane operations, as introduced in [1]. Actually, we formalize only three of them, *mate, drip, exo*, because only these operations are used below.

First, the notation: the fact that a multiset u is placed on the internal side of a membrane and a multiset v on its external side is denoted as $[\ \ {}_u]_v$, hence with the right hand bracket having both left and right subscripts.

Then, the three operations are as follows:

$$\begin{aligned}
\mathit{mate} &: [\ \ ]_{ua}[\ \ ]_{v} \rightarrow [\ \ ]_{uxv},\\
\mathit{drip} &: [\ \ ]_{uav} \rightarrow [\ \ ]_{ux}[\ \ ]_{v},\\
\mathit{exo} &: [[\ ]_{v}\ \ {}_{au}] \rightarrow [\ \ {}_{uxv}],
\end{aligned}$$

where $a \in V, u, x \in V^*$, $ux \neq \lambda$, $v \in V^+$, for an alphabet V of proteins. One sees that there is no difference between the projective and the standard *mate, drip* operations, they use only proteins placed on the external side of membranes. In what concerns the proteins present on the membranes entering these operations, they are distributed on one of the sides of the resulting membranes as suggested below (x_1, x_2, x_3, x_4 are generic multisets, and $\mathbf{Q}$ indicates the contents of the respective membranes, i.e., the possible membranes present inside it):

$$\begin{aligned}
\mathit{mate} &: [\ \ {}_{x_1}]_{uax_2}[\ \ {}_{x_3}]_{vx_4} \rightarrow [\ \ {}_{x_1x_3}]_{uxvx_2x_4},\\
\mathit{drip} &: [\ \mathbf{Q}\ \ {}_{x_1x_2}]_{uavx_3x_4} \rightarrow [\ \ {}_{x_1}]_{uxx_3}[\ \mathbf{Q}\ \ {}_{x_2}]_{vx_4},\\
\mathit{exo} &: [[\ \mathbf{Q}\ \ {}_{x_1}]_{vx_2}\ \ {}_{aux_3}]_{x_4} \rightarrow [\ \ {}_{uxvx_2x_3}]_{x_1x_4}\mathbf{Q}.
\end{aligned}$$

Thus, the operations are controlled by the multiset uav, with u and v being the "context" where a is transformed; all other proteins are move unchanged on the resulting membranes, with precise destinations, in the case of *exo* changing the inside-outside position; when splitting a multiset in two multisets, this is done randomly, one of them can be any sub-multiset of the starting multiset and the other the complement.

An additional extension would consist in allowing two "dual" operations of *drip* type, by letting it be driven by proteins occurring either on the external side, or on the internal side of the membranes (but the two cases should not be mixed together, if one wants to maintain some biological motivation). Namely, the *drip* operation could be substituted by

$$\begin{aligned}
\mathit{drip}^{int} &: [\mathbf{Q}\ \ {}_{x_1x_2uav}]_{x_3x_4} \rightarrow [\ {}_{x_1ux}]_{x_3}\ [\mathbf{Q}\ \ {}_{x_2v}]_{x_4},\\
\mathit{drip}^{ext} &: [\mathbf{Q}\ \ {}_{x_1x_2}]_{uavx_3x_4} \rightarrow [\ {}_{x_1}]_{uxx_3}\ [\mathbf{Q}\ \ {}_{x_2}]_{vx_4}.
\end{aligned}$$

The same is not biologically motivated for *mate* or *exo* operations, hence this note introduces some difference between (mem)brane operations which can exist – with respect to the appearance side of active proteins – only in one form, that is, *mate* and *exo*, and those which can exist in two reciprocal forms, that is, *drip*. In what follows, we will not consider this extension anyway.

Now, a P system using operations as above is defined in the standard way, with the only difference that for each membrane we have to specify two multisets, the one marking it from inside and the one marking it from outside. Formally, we write $\Pi = (A, \mu, (u_1, v_1), \ldots, (u_m, v_m), R)$, with the meaning that u_i is the internal marking of membrane i and v_i is the external marking of this membrane. In each step, each membrane and each protein can be involved in only one rule, but the rules are applied in the maximally parallel way, non-deterministically choosing the rules, the objects, and the membranes. In the same style as in Section 3, we consider as successful only halting computations (we do not need to restrict the number of membranes – although this is possible, as we will see in the next section, even to a single membrane).

In this moment, we have a problem: how to define the result of a computation? If, like above, we impose to have only one inner membrane and we count the objects placed on this membrane in the halting configuration, then the proof of Theorem 1 also holds for the projective case: there are only two levels of membranes, and the skin membrane is inert, it never participates to any operation; if we assume that the inner membranes are marked on their external side, the rules used in the proof of Theorem 1 can be considered as projective *mate* and *drip*.

Still, as mentioned also in the Introduction, reading the result of a computation on an inner membrane might look unacceptable, that is why we define now the result of a computation as the number of objects which mark the external side of the skin membrane of the system in the halting configuration.

Again, the length of the string uav involved in a rule gives the weight of that rule. Then, the family of all sets $N(\Pi)$ computed by P systems Π using at any moment during a halting computation at most m membranes, and projective *mate, drip, exo* rules of weight at most p, q, r, respectively, is denoted by $NOP_m(pmate_p, pdrip_q, pexo_r)$. When one of the parameters m, p, q, r is not bounded we replace it with $*$. When number 1 is ignored, we denote by $1NOP_m(pmate_p, pdrip_q, pexo_r)$ the corresponding families.

6 Universality for the Projective Operations

For the above way of defining the result of a computation, on the external side of the skin membrane, it is not possible to use only *mate* and *drip* operations, we also need operations which handle multisets placed on the inner side of membranes – in particular, on the inner side of the skin membrane.

Theorem 2. *$1NRE = 1NOP_m(pmate_p, pdrip_q, pexo_r)$ for all $m \geq 6$, $p \geq 4$, $q \geq 4$, and $r \geq 3$.*

Proof. We proceed as in the proof of Theorem 1, proving only the inclusion $1NRE \subseteq 1NOP_6(pmate_4, pdrip_4, pexo_3)$. For a set $Q \in 1NRE$, we consider the set $Q' = \{n - 1 \mid n \in Q\}$. This is clearly a set in NRE, hence there is a register machine M such that $N(M) = Q'$. We construct the P system Π as in the proof of Theorem 1, such that $Q' = N(M) = N(\Pi)$, and then we modify this system as follows.

First, we start from the initial configuration

$$[\ [\ [\quad {}_{f_1}]_{l_0 h}\ [\quad {}_{\lambda}]_{bc}\quad {}_{f_2}]_{\lambda}\quad {}_{f_3}]_{\lambda}$$

(some membranes are marked on both sides, and one further membrane is added around the system); the new objects f_1, f_2, f_3 are added to the alphabet of objects.

We repeat the construction of system Π with the markings of membranes used in rules considered of the projective type. Because the *mate* and *drip* operations do not move proteins from one side of a membrane to another side, the simulation of ADD and SUB instructions using the projective version of the rules from Tables 1 and 2 are correct, and they do not use the external membranes, those with internal markings f_2 and f_3.

In what concerns the final sequence of steps, we replace the last two rows from Table 3 with the two steps indicated in Table 4 – we recall also the configuration obtained after step 3, including the multiset a_1^n marking the inner membrane, $n \geq 1$.

Step	Rules	Types	Configuration
7			$[[[\ \ {}_{f_1}]_{bh'a_1a_1^{n-1}}\ {}_{f_2}]_\lambda\ {}_{f_3}]_\lambda$
8	$[[\ ]_{h'a_1}\ {}_{f_2}] \rightarrow [\ \ {}_{h'a_1}]$	exo	$[[\ \ {}_{h'a_1^n}]_{f_1}\ \ {}_{f_3}]_\lambda$
9	$[[\ ]_{f_1}\ \ {}_{f_3}] \rightarrow [\ \ {}_{f_1}]$	exo	$[\ \ {}_{f_1}]_{h'a_1^n}$

Table 4. Rules for the last two steps of the computation in the projective case.

The two *exo* steps move the result of the computation on the external side of the unique membrane present in the system in the halting configuration. Besides the n copies of a_1 corresponding to a value $n \in N(M) = Q'$, we also have here the object h', hence we compute the number $n + 1 \in Q$. Consequently, for Π' being the system obtained by modifying Π as suggested above, we have $N(\Pi') = Q$. Because the *exo* rules have the weight at most three, the proof is complete. $\square$

It remains as an open problem to improve the previous result by including also number 1 in the computed sets.

7 Final Remarks

We have improved here the universality result from [5], decreasing the number of membranes from 11 to 5. In the case of projective operations, we have shown that the universality can be obtained (using one additional membrane and "losing" the number 1) also when imposing that the result of a computation is "accessible" from outside the system, namely, encoded in the number of objects placed on the external side of the skin membrane. The optimality of these results remains to be checked.

A related general research topic is to investigate other combinations of rules, checking whether they lead or not to universality.

References

1. V. Danos, S. Pradalier: Projective brane calculus. In *Computational Methods in Systems Biology: International Conference CMSB 2004, Paris, France, May 26-28, 2004, Revised Selected Papers*, (V. Danos, V. Schachter, eds.), LNCS 3082, Springer-Verlag, Berlin, 2005, 134–148.
2. N. Busi, R. Gorrieri: On the computational power of brane calculi. *Third Workshop on Computational Methods in Systems Biology*, Edinburgh, 2005.
3. N. Busi: On the computational power of the mate/bud/drip brane calculus: interleaving vs, maximal parallelism. In *Membrane Computing, International Workshop, WMC6, Vienna, Austria, 2005, Selected and Invited Papers* (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 3850, Springer-Verlag, Berlin, 2006, 144–158.
4. L. Cardelli: Brane calculi. Interactions of biological membranes. In *Computational Methods in Systems Biology. International Conference CMSB 2004, Paris, France, May 2004, Revised Selected Papers* (V. Danos, V. Schachter, eds.), LNCS 3082, Springer-Verlag, Berlin, 2005, 257–280.
5. L. Cardelli, Gh. Păun: An universality result for a (mem)brane calculus based on mate/drip operations. *Intern. J. Found. Computer Sci.*, 17, 1 (2006), 49–68.
6. M. Cavaliere, A. Riscos-Nunez, R. Brijder, G. Rozenberg: Membrane systems with marked membranes. Submitted, 2005.
7. M. Minsky: *Computation – Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
8. A. Păun, B. Popa: P systems with proteins on membranes. Submitted, 2005.
9. Gh. Păun: *Membrane Computing – An Introduction.* Springer-Verlag, Berlin, 2002.
10. Gh. Păun: One more universality result for P systems with objects on membranes. *Proc. Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, RGNC Report 01/2005, 263–274.
11. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.
12. The P Systems Web Page: `http://psystems.disco.unimib.it`.