Membrane computing and brane calculi. Old, new, and future bridges

Gheorghe Păun *

Institute of Mathematics of the Romanian Academy, PO Box 1-764, 014700 București, Romania
Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

A R T I C L E   I N F O

Keywords:
Membrane computing
P systems
Brane calculi
Turing computability

A B S T R A C T

After a short discussion about similarities and dissimilarities of membrane computing and brane calculi, insisting mainly on some recent ideas of bridging the two areas of research, one recalls some details concerning certain classes of P systems based on brane calculi operations. Several open problems are formulated in this context.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

"Membrane computing is a young branch of natural computing, initiated in [32], aiming to abstract computing ideas, models, paradigms from the structure and functioning of the cell and from the cells organization in tissues, organs, and other higher order structures. The obtained devices, currently called P systems, are parallel distributed computing models, processing multisets of objects in compartments defined by membranes. There were investigated many classes of P systems and most of them are computationally complete; when an exponential workspace can be created in a polynomial time, e.g., by membrane division, then polynomial (often, linear) solutions to computationally hard problems (typically, NP-complete problems) can be devised."

This paragraph can be found almost in this form in the introduction of many papers in the membrane computing area; very frequent was and still is the illustration of the notion of a membrane structure (and of the associated terminology) from Fig. 1, a sort of logo of the domain.

To a great extent, these phrases and this figure capture the essence of membrane computing. It should be added another slogan of the field, stating that "the rules are used in the non-deterministic maximally parallel way", and that recently membrane computing proved to be a very promising framework for devising models for biology (and other areas, such as economics and linguistics) – with surprising applications in unexpected areas, such as approximate optimization, in the sense of evolutionary computing.

We already have here a list of features of membrane computing which are similar or different from corresponding features of brane calculi.

The next section is devoted to explicitly listing some (no such list can be complete) of these common or different features of the two areas. In principle, each of these points is an invitation to extend ingredients from one domain to another one, and in many cases such bridging investigations have already started.

* Corresponding address: Institute of Mathematics of the Romanian Academy, PO Box 1-764, 014700 București, Romania.
E-mail addresses: george.pau@imar.ro, gpaun@us.es.

1 Not anymore: eight years is a long time for a bio-inspired research area, and this is also proved by the data from [39] (bibliography, events, results, applications).

2 This is indeed a young direction of research, [8]. Actually, I am considering here only the two brane calculi from [8], based on pino, exo, phago, respectively, mate, drip, bud operations, although L. Cardelli and his collaborators have considered also other process algebra calculi before or after [8]; see, e.g., [37,21].

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.04.001
2. Things in common, things different

Of course, no ordering is assumed below among the concepts invoked, as well as no pretention of completeness. Furthermore, no technical details are given, as the reader is assumed to be familiar with basic elements of both membrane computing and brane calculi.

- **Objects in regions, objects on membranes**: This is a much mentioned difference, which, actually, is not totally valid. Even in the standard setup, of so-called transition P systems with symbol-objects, it is possible to assume that we have objects bound on the membrane, by simply assuming that the membrane is a bilayer with an interspace between layers. Actually, this was done already in [29], and was also briefly investigated in Section 9.3 of [33]. Sure, this trick with making a compartment from the membrane itself, raises a series of restrictions in handling the respective objects (and cannot be used for formalizing operations with membranes).

- **Anyway, at first sight, the role of objects** placed on membranes is different in the two areas: in membrane computing the focus is on the evolution of objects themselves, while in brane calculi the objects ("proteins") mainly control the evolution of membranes. These protein-objects also evolve at the same time, and this is a very promising research topic, up to now probably the most fruitful and most investigated import of ideas from brane calculi to membrane computing: considering P systems with objects on membranes used (i) to control brane calculi operations with membranes, but with the multisets of protein-objects being the main data structure on which the computation works, or (ii) to control usual multiset processing rules for objects placed in compartments, maybe also evolving the objects bound on membranes. The first idea is followed in [10,3,26,34] (in this last paper, the operations are not exactly those from brane calculi, in the sense that their semantics – the way of moving the contents of membranes – is different, that is why the operations were called *create* and *dissolve*), while the second one is investigated in [15,13,14,30,31,27]. In [15,13,14] the "proteins" can be attached to and detached from membranes, while in the other papers they remain always bound on membranes.

- **The focus on operations with membranes** is again only in a superficial view a difference between the two areas. While the basic data structure of membrane computing is indeed the multiset of symbol-objects (but also sets of strings, and sporadically also sets of symbols or numerical variables), rules for handling membranes were always used, starting with [32], where the operation of membrane dissolving was introduced, for a while considered an "innocent" operation which however proved to be crucial in investigations related to complexity matters (see, e.g., [25]). Membrane creation and membrane division are basic ways to create the exponential workspace for solving NP-complete (recently, also PSPACE-complete) problems in polynomial time using P systems. Merging, separating, gemmating, and other operations with membranes were also used in a series of papers. There are also papers (we refer to the bibliography from [39] for details) where the membrane structure itself is the goal of the computations, e.g., as a description of a tree, or a way to describe strings in a context-free language. A systematic investigation in this respect was started in [19], where a sort of AFL (Abstract Family of Languages) theory of membrane structures and their operations was proposed – still waiting for further research efforts.

- **Synchronous versus asynchronous computing**: This was many times invoked as the main difference between membrane computing and brane calculi, and again the assertion is only partially valid. It is true that most papers in membrane computing deal with synchronized systems, having a general clock which marks the time for the whole system and in each time unit each compartment has to evolve in a maximal way. This assumption starts from the observation that biology "has a high level of parallelism", but, of course, it is mathematically grounded/useful: the synchronization (especially, the maximal parallelism, alone or together with the halting condition in defining successful computations) allows "appearance checking" (in the terminology of regulated rewriting) and "check for zero" (in the terminology of register/counter machines), hence this is a shortcut towards Turing universality. \(^3\) There are however a large number of papers where the computations are sequential – see [22] and its references.

There also are variants/degrees of parallelism. For instance, P systems with bounded parallelism were investigated in a series of papers by O.H. Ibarra and his group (see, e.g., [20]), as well as in [2]. In [11,12] one considers time-free, clock-free, time-independent systems, while G. Ciobanu and his group has considered both sequential and time-independent systems (see, e.g., [1]). The reader can find further references at [39]. Recently, a promising version of parallelism was introduced in [16], called minimal parallelism: in each region of a P system, if at least one rule can be used, then at least one must be used, without any other restriction.

- **Halting computations**: Starting with Turing machines, halting is a standard way to define successful computations and it is introduced in membrane computing mainly to have a simple and powerful condition for defining the moment when the result of a computation "is ready", but for certainly it is neither essential nor obligatory – although so frequently used. The end of a computation can also be signaled by means of events of various kinds, and this was done in several papers: P. Frisco frequently uses an acknowledging membrane (initially empty; the result of a computation is defined at the moment when any object enter this membrane; see, e.g., [24]); in general, in the case of P systems with string-objects one does not work with halting computations; events were considered recently also in [28], while in neural-like

\(^3\) There fit with this intuition those results that show that brane calculi devices working synchronously are Turing complete/undecidable, but this is not the case for asynchronous devices – see [4,5].
P systems and P automata in the sense of E. Csetah-Varjú and G. Vaszil one uses final states for concluding a computation (see a survey of P automata area in [18]). Then, of course, no idea of halting appears in applications of P systems, especially in biology or economics, where the goal is not the result of a computation but the computation itself, the evolution in time of populations of objects.

- We arrive now at an important point, also many times invoked: brane calculi pay more attention to the faithfulness to the biological reality, membrane computing is mainly interested in computational issues. Right in both parts of the assertion, whilst mentioning that in the case of membrane computing this is true only for those investigations ...interested in computational issues. When trying to build computing models, powerful or efficient, we need mathematical models, as elegant as possible, hence as restricted as possible. When trying to build models of biological processes, we stay as close as possible to biology. This is the case with the many recent applications, from oscillations in eco-systems to EGFR robustness, from circadian rhythms to quorum sensing in bacteria, from photosynthesis to Fas induced apoptosis. Details can be found in [17,38,39].

- It is perhaps the time to also mention a series of obvious common features of the two domains: both of them are directly inspired from the cell structure and functioning, are based on discrete mathematics, can handle small populations of agents and slow reactions, are implicitly or explicitly algorithmic, lead to scalable models, easily understood by biologists. All these are important arguments in promoting them as modeling tools for biology.

We can conclude that the only real difference between the two research areas is the fact that, basically, brane calculi uses process algebra as its technical framework, while membrane computing uses techniques from languages, automata, complexity, dynamical systems.

3. P systems with brane calculi operations

As an illustration of a fruitful recent bridge between membrane computing and brane calculi, in this section we recall the results from [10,34,39], where P systems were investigated using operations with membranes introduced in [8].

Six operations were considered in [8], a calculus based on pino, exo, phago operations and one based on mate, drip, bud operations. They are formalized as follows (we refer to [8,10] for explanations):

\[
\begin{align*}
\text{pino} & : \ [ P \ u \ | \ (pino(s).t) \rightarrow \ [ P \ s \ | \ u \ | \ t, \\
\text{exo} & : \ [ P \ u \ | \ (exo.t)Q \ w \ | \ (co-exo.v) \rightarrow \ P \ Q \ | \ u \ | \ w \ | \ v, \\
\text{phago} & : \ [ P \ u \ | \ (phago.t)Q \ w \ | \ (co-phago.s.v) \rightarrow \ [ P \ u \ | \ t \ | \ sQ \ w \ | \ v, \\
\text{drip} & : \ [ P \ u \ | \ (drip(s).t) \rightarrow \ [ P \ u \ | \ t \ | \ s, \\
\text{mate} & : \ [ P \ u \ | \ (mate.t)Q \ w \ | \ (co-mate.s) \rightarrow \ [ P \ Q \ u \ | \ t \ | \ w, \\
\text{bud} & : \ [ P \ u \ | \ (bud.t)Q \ w \ | \ (co-bud.s.v) \rightarrow \ [ P \ u \ | \ | t \ | \ sQ \ | \ w, \\
\end{align*}
\]

These operations create new membranes or merge two membranes into a single one, under the control of proteins placed on them.

In terms of membrane computing, we denote a membrane having a multiset \( u \) of proteins bound on it in the form \([ \_u \].\) Then, the previous operations can be rewritten as follows – actually, only pino, exo, mate, drip were formalized in this new framework, while considering P systems with phago and bud operations is still an open problem:

\[
\begin{align*}
pino_i : & \ [ I_{uv} \rightarrow \ [ [ I_u ]_v, \\
exo_i : & \ [ [ I_u ]_v \rightarrow \ [ I_{uv}, \\
pino_e : & \ [ I_{uv} \rightarrow \ [ I_v ]_{ux}, \\
exo_e : & \ [ [ I_u ]_v \rightarrow \ [ I_{uv}, \\
\end{align*}
\]
mate : [ ]_{ux} \{ v \} \rightarrow [ ]_{ux} \{ P Q \}, \quad (5)

drip : [ ]_{uv} \rightarrow [ ]_{ux} \{ v \}. \quad (6)

In all cases, we consider an alphabet \( A \) of proteins, and \( a \in A, u, v, x \in A^* \). The length of the string \( uv, v \) (hence the total multiplicity of the multiset represented by this string) from each rule is called the weight of the rule.

As one can see, for pino and exo there are two possibilities. 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 \( pino_{e}, exo_{e} \).

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.

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 side 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. A more formal definition can be found in [10].

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. Denoting these contents (empty or consisting of other membranes) by \( P, Q \), we can write the six operations as follows:

\[
pino_{i} : [ ]_{uv} \rightarrow [ ]_{ux} P, \quad (7)
\]
\[
exo_{i} : [ ]_{ux} Q, \rightarrow [ ]_{ux} P \rightarrow P, \quad (8)
\]
\[
pino_{e} : [ ]_{uv} \rightarrow [ ]_{ux} P \rightarrow P, \quad (9)
\]
\[
exo_{e} : [ ]_{ux} Q, \rightarrow [ ]_{ux} P \rightarrow P, \quad (10)
\]
\[
mate : [ ]_{uv} [ ]_{vx} \rightarrow [ ]_{uv} P Q, \quad (11)
\]
\[
drip : [ ]_{uv} \rightarrow [ ]_{ux} P, \quad (12)
\]

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. \( \mu \) 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 \( \mu \) at the beginning of the computation (one assumes that the membranes in \( \mu \) 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 labeled with 0 and \( u_0 = \lambda \);
4. \( R \) is a finite set of 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.

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 and the membrane marking the membranes; thus, the initial configuration is that defined by \( \mu \) and \( u_0, u_1, \ldots, u_m \). 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.

The set of all numbers (zero is ignored) computed in this way by \( \Pi \) is denoted by \( N(\Pi) \).

In [10,3], only systems with mate and drip rules were considered. The family of all sets \( N(\Pi) \) computed by \( P \) systems \( \Pi \) 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 \( NOP_{m}(mate_{p}, drip_{q}) \).
In [10] it is proved that \( NRE = NOP_m(\text{mate}_p, \text{drip}_q) \) for all \( m \geq 11, p \geq 5, \) and \( q \geq 5, \) but the result was improved in [3] for each of the three subscripts.

**Theorem 3.1.** \( NRE = NOP_m(\text{mate}_p, \text{drip}_q) \) for all \( m \geq 5, p \geq 4, \) and \( q \geq 4. \)

The use of \( \text{pino, exo} \) operations was left open. The \( \text{exo} \) operation was however used in [3] in the case of considering projective operations, in the sense of [21]. The idea is that the proteins are not simply placed on a membrane, but on a given side of the membrane, and this requests a different formalizations of the brane calculi operations (actually, only the \( \text{mate}, \text{drip}, \text{exo} \) operations were considered in [3], hence again appears an open problem: to consider the projective version of all six operations).

First, a 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 defined as follows in the projective case:

\[
\begin{align*}
\text{mate} & : [1]_{uv} [1]_v \rightarrow [1]_{uav},
\text{drip} & : [1]_{uv} \rightarrow [1]_{uv}, [1]_v,
\text{exo} & : [1]_v \rightarrow [1]_{uav},
\end{align*}
\]

where \( a \in A, u, x, v \in A^*, \) for an alphabet \( A \) of proteins. There is no difference between the projective and the standard \( \text{mate}, \text{drip}, \text{exo} \) 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 above \( (x_1, x_2, x_3, x_4 \text{ are generic multisets, and Q indicates the contents of the respective membranes, i.e., the possible membranes present inside it):} \)

\[
\begin{align*}
\text{mate} & : [1]_{x_1} [1]_{u_2} [1]_{x_3} [1]_{x_4} \rightarrow [1]_{x_1 x_3} [1]_{u_2 x_4},
\text{drip} & : [Q]_{x_1} [1]_{u_2} [1]_{x_2} [1]_{x_4} \rightarrow [1]_{x_1} [1]_{u_2} [Q]_{x_2} [1]_{x_4},
\text{exo} & : [1]_{x_1} [1]_{u_2} [1]_{x_3} [1]_{x_4} \rightarrow [1]_{u_2 x_4} [1]_{x_1} [1]_{x_3} Q.
\end{align*}
\]

As usual, 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 \( \text{exo} \) changing the inside–outside position.

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 \( H = (A, \mu, (u_1, v_1), \ldots, (u_m, v_m), R), \) with the meaning that \( u_i \) is the internal marking of membrane \( i \) and \( v_i \) is the external marking of this membrane. The functioning of such a system is defined as usual, with a difference arising when defining the result of a computation: because reading the result of a computation on an inner membrane might look unacceptable in the projective case, 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.

The family of all sets \( N(H) \) computed by \( P \) systems \( H \) using at any moment during a halting computation at most \( m \) membranes, and projective \( \text{mate}, \text{drip}, \text{exo} \) rules of weight at most \( p, q, r, \) respectively, is denoted by \( NOP_m(\text{pmate}_p, \text{pdrip}_q, \text{pexo}_r). \) When number 1 is ignored (as always happens with zero), we denote by \( 1NOP_m(\text{pmate}_p, \text{pdrip}_q, \text{pexo}_r) \) the corresponding families.

The following counterpart of Theorem 3.1 was proved in [3] (it is open whether it can be improved by including also number 1 in the computed sets):

**Theorem 3.2.** \( 1NRE = 1NOP_m(\text{pmate}_p, \text{pdrip}_q, \text{pexo}_r) \) for all \( m \geq 6, p \geq 4, q \geq 4, \) and \( r \geq 3. \)

A partial answer to the previously formulated open problem concerning the study of \( P \) systems based on \( \text{pino, exo} \) operations was given in [34], based however on a different interpretation of these operations – a different way of handling the contents of the membranes. Because the new semantics do not correspond to the operations \( \text{pino, exo} \) from brane calculi, we change the name of operations to \( \text{cre}, \text{dis}, \) for “membrane creation” and “membrane dissolution”). This semantics is suggested below, for the four cases corresponding to (7)–(10):

\[
\begin{align*}
\text{cre} & : [P]_{uv} \rightarrow [[P]]_{uv},
\text{dis} & : [[P]_{uv} Q]_v \rightarrow [P]_v Q_{uv},
\text{cre} & : [P]_{uv} \rightarrow [[P]]_{uv},
\text{dis} & : [[P]_u Q]_{av} \rightarrow [P]_{uav}.
\end{align*}
\]

That is, when a membrane is created inside an existing membrane, the new membrane contains all previously existing membranes, and while dissolving a membrane, its contents remains inside the membrane where it was placed before the operation. The interpretation of the latter operation is rather similar to the usual dissolution operation in membrane computing, while the membrane creation is understood as doubling the existing membrane, with a distribution of the multiset marking the initial membrane to the two new membranes.
Using rules as defined above, we can define a P system as usual. The family of all sets of numbers \( N(\Omega) \) computed by P systems \( \Omega \) using at any moment during a computation at most \( m \) membranes, and \( \text{cre} \), \( \text{dis} \) rules of weight at most \( p \), \( q \), respectively, is denoted by \( \text{NOP}_m(\text{cre}_p, \text{dis}_q) \).

The proof of the following result can be found in [34]:

**Theorem 3.3.** \( 1 \text{NRE} = 1 \text{NOP}_m(\text{cre}_p, \text{dis}_q) \) for all \( m \geq 7 \), \( p \geq 4 \), and \( q \geq 4 \).

It remains as an open problem to check whether or not the parameters appearing in Theorems 3.1–3.3 are optimal (presumably, not), to improve them, as well as to investigate the size of families with small values of these parameters (in the hope to find non-universal families, perhaps non-trivial, also having decidable properties).

Of course, another topic to investigate is the case of non-synchronized functioning of the previous systems; non-universality results are expected in those cases.

### 4. Closing remarks

After an informal discussion of similarities and differences between membrane computing and brane calculi, we have recalled the results from [10,3,34], concerning the computational completeness of certain classes of P systems using brane calculi operations, mainly as an opportunity to call attention to the many open problems which remain to be investigated in this framework.

### For further reading

[6], [7], [9], [23], [35], [36]

### Acknowledgements

Work partially supported by Project BioMAT 2-CEx06-11-97/19.09.06.

### References


