Membrane Systems from the Viewpoint of the Chemical Computing Paradigm

Péter Battyányi and György Vaszil

Abstract—**Membrane systems are nature motivated abstract computational models inspired by basic features of biological cells and their membranes. They are examples of the chemical computational paradigm which describes computation in terms of chemical solutions where molecules interact according to rules defining their reaction capabilities. In this survey, we first review some of the basic features and properties of the chemical paradigm of computation, and also give a short introduction to membrane systems. Then examine the relationship of the certain chemical programming formalisms and some simple types of membrane systems.**

Index Terms—**Abstract computational models, chemical computing paradigm, membrane systems.**

I. INTRODUCTION

MEMBRANE systems are abstract computational models inspired by the architecture and the functioning of biological cells. Their structure consists of hierarchically embedded membranes, with multisets of symbolic objects associated to the regions enclosed by them. The evolution of the system is governed by rules assigned to the regions. The system performs nondeterministic transformations of these multisets, which produces a series of configuration changes which is interpreted as a computation. The area was initiated by Gh. Păun in [11] and the literature on the domain has grown very fast. Soon it became one of the most important and most popular areas of Natural Computing. For details on the developments, consult the monograph [12] or the more recent handbook [13].

In this survey, we look at the field of membrane computing as a particular example of the so called chemical computational paradigm. This paradigm aims to describe computations in terms of a symbolic chemical solution of molecules and the reactions which can take place between them. Its origins go back to the Gamma programming language of Bânatre and Le Métayer introduced in [6], [7]. Their aim was to free the expression of algorithms from the sequentiality which is not inherently present in the problem to be solved, that is, the sequentiality which is implied by the structure of the computational model on which the given algorithm is to be performed. In other words, their aim was to free the programmer from the necessity of taking into account the underlying architecture of the machine that is being programmed.

The idea was carried on into several directions, see [3] for an overview. From our point of view, one of the most interesting developments was the introduction of the so called chemical abstract machine, see [9], where the notion of membrane appears serving as a delimiter between different types of sub-solutions, forcing the reactions of the sub-solutions to occur in a locally isolated way. This model and the idea of locally delimited regions and membranes was one of the explicit motivations behind membrane systems, as they appear in [11].

In the following we give a short introduction to some of the formalisms used to describe computations in the chemical way, and also present some of the basic notions of membrane computing. Then, based on the results of [10] and [8] we present some ideas on how the chemical formalisms and membrane systems can be related to each other. This approach is interesting in at least two ways. By being able to translate chemical programs to membrane systems, we could obtain a high level programming language for the description of membrane algorithms. On the other hand, by being able to describe membrane computations with some of the chemical formalisms, we would be able to reason about the properties of membrane systems in a mathematically precise manner.

Manuscript received October 23, 2014, revised December 8, 2014.

Péter Battyányi and György Vaszil are with the Department of Computer Science, Faculty of Informatics, University of Debrecen, Kassai út 26, 4028 Debrecen, Hungary, e-mail: battyanyi.peter@inf.unideb.hu, vaszil.gyorgy@inf.unideb.hu

II. PRELIMINARY DEFINITIONS AND NOTATION

An alphabet is a finite non-empty set of symbols V, the set of strings over V is denoted by V^* .

A finite multiset over an alphabet V is a mapping $M : V \to \mathbb{N}$ where $\mathbb{N}$ denotes the set of non-negative integers, and $M(a)$ for $a \in V$ is said to be the multiplicity of a in M. The set of all finite multisets over the set V is denoted by $\mathcal{M}(V)$.

We usually enumerate the not necessarily distinct elements $a_1, \ldots, a_n$ of a multiset as $M = \langle a_1, \ldots, a_n \rangle$, but the multiset M can also be represented by any permutation of a string $w = a_1^{M(a_1)} a_2^{M(a_2)} \ldots a_n^{M(a_n)} \in V^*$, where if $M(x) \neq 0$, then there exists j, $1 \leq j \leq n$, such that $x = a_j$. The empty multiset is denoted by $\emptyset$.

For more on the basics of formal language theory and Membrane Computing the reader is referred to the monograph [15], and the handbooks [14] and [13].

III. COMPUTATION AS REACTIONS IN A CHEMICAL SOLUTION

A chemical "machine" can be thought of as a symbolic chemical solution where data can be seen as molecules and operations as chemical reactions. If some molecules satisfy a reaction condition, they are replaced by the result of the reaction. If no reaction is possible, the program terminates. Chemical solutions are represented by multisets. Molecules interact freely according to reaction rules which results in an

TABLE I
THE ELEMENTS OF THE ABSTRACT MACHINE AND THE INGREDIENTS OF A CHEMICAL SOLUTION.

Abstract machine	Chemistry
Data	Molecule
Multiset	Solution
Parallelism/nondeterminism	Brownian motion
Computation	Reaction

implicitly parallel, non-deterministic, distributed model. The chemical analogy is carried over also to the execution model: The Brownian motion of the chemical molecules correspond to the parallel and nondeterministic computation of the chemical machine. See table I for a summary of this correspondence.

To help the easier understanding of the notions, we start with the discussion of the Higher-order Chemical Language (HOCL) from [1], which can be presented in a more reader-friendly manner as the mathematically more precise γ-calculus, see [4], which we will also discuss later.

In general, a reaction rule can be written as

$$\textbf{replace } P \textbf{ by } M \textbf{ if } C$$

where *P* is a pattern, *C* is the reaction condition, and *M* is the result of the reaction. For example, the solution

$$\langle(\textbf{replace } x, y \textbf{ by } x \textbf{ if } x < y),\ 2,\ 7,\ 4,\ 3,\ 6,\ 8\rangle$$

will result in the solution

$$\langle(\textbf{replace } x, y \textbf{ by } x \textbf{ if } x < y),\ 2\rangle$$

containing the reaction and the minimum value among the molecules. Notice that the order in which the reactions are performed, that is, the order in which the numbers are compared is not specified.

Solutions can also contain sub-solutions, as seen in the following example, where the least common multiple of 4 and 6 is computed.

Example 1: Let us start with

$$\begin{aligned}&\textbf{let } \text{multiplier} = \textbf{replace } x, \omega \textbf{ by } \omega \textbf{ if}\\ &\qquad\qquad\qquad\qquad \text{not}(4 \text{ div } x \text{ and } 6 \text{ div } x),\\ &\textbf{let } \text{clean} = \textbf{replace-one } \langle\text{multiplier}, \omega\rangle \textbf{ by } \omega,\\ &\textbf{let } \text{min} = \textbf{replace } x,\ y \textbf{ by } x \textbf{ if } x < y,\end{aligned}$$

and consider the following solution

$$\begin{aligned}&\langle\text{min, clean, }\langle\text{multiplier, } 10,\ 11,\ 12,\ 13,\ 14,\ 15,\\ &\qquad 16,\ 17,\ 18,\ 19,\ 20,\ 21,\ 22,\ 23,\ 24\rangle\ \rangle\end{aligned}$$

where the "top level" is a solution containing the reactions *min, clean* and a sub-solution, which is another solution with the reaction *multiplier* and a set of numbers. When the sub-solution becomes inert, that is, when the common multiples of 4 and 6 are selected, the reactions of the top level, *min* or *clean* are activated. First only the condition of *clean* can be matched, so it is applied, and the remaining numbers from the sub-solution are "moved" one level higher while the sub-solution and the reaction *multiplier* are eliminated. Now the conditions of *min* can also be matched, resulting in the solution

$$\langle\text{min, } 12\rangle.$$

Notice the pattern ω which is special in the sense that it can match anything, and the "one-shot" reaction *clean* using the keyword **replace-one**, meaning that its application is only possible once, as the application "consumes" the reaction itself.

Based on these examples, the reader might agree that we can formulate some of the important characteristic properties of the chemical computational model as follows:

- *Parallel execution*: when two reactions involve distinct elements, they can occur simultaneously,
- *mutual exclusion*: a molecule cannot take part in more than one reaction at the same time,
- *atomic capture*: either all ingredients of the reaction are present, or no reaction occurs.

Example 2: To see why these characteristics are important, consider the problem of the dining philosophers, a common example for the demonstration of concurrent algorithm design techniques. Let us state the problem as follows: There are 5 philosophers sitting at a round table, with 5 plates of spaghetti in front of them, and 5 forks between the plates. A philosopher is either thinking or eating, but when he is eating, he needs both of the forks on each side of his plate since philosophers only eat spaghetti with two forks. Thus, it is not possible that two neighbors eat at the same time.

A description of the problem can be given in the above described chemical formalism as follows. Let

$$\begin{aligned}&\textbf{let } \text{eat} = \textbf{replace } Fork : f_1, Fork : f_2 \textbf{ by } Phil : f_1 \textbf{ if}\\ &\qquad\qquad\qquad\qquad\qquad\qquad f_2 = f_1 + 1 \bmod 5,\\ &\textbf{let } \text{think} = \textbf{replace } Phi : f \textbf{ by}\\ &\qquad\qquad\qquad Fork : f, Fork : f + 1 \bmod 5 \ \textbf{ if } true,\end{aligned}$$

and consider the following symbolic solution

$$\langle\text{eat, think, } Fork : 1,\ Fork : 2, \ldots, Fork : 5\rangle$$

which contains two reaction rules and five numbered objects of the type *Fork*, representing the situation when all the forks are on the table, that is, when no philosopher is eating. This situation can change through the reaction *eat* which replaces to adjacent forks with the corresponding numbered object of the type *Phil*. Conversely, the reaction *think* replaces an eating philosopher with the corresponding forks.

Consider now the consequences of the above mentioned three characteristic properties for the behavior of this setup. Due to *parallel execution*, if two philosophers are not neighbors at the table, they are allowed to eat simultaneously and independently of each other. The *mutual exclusion* property guarantees that one fork is used by at most one philosopher, and as the consequence of *atomic capture*, deadlocks are "automatically" avoided, since a fork can be picked up by a philosopher only in the case when the other fork is also available.

Now, before we continue, based mainly on [5], we present a more rigorous formalism which we will extend in Section V. As we have already seen, it is a tool for multiset manipulation, and the programs are collections of pairs of reaction conditions and actions. The Γ function is defined as

$$\Gamma((R_1, A_1), \ldots, (R_k, A_k))(M) =$$

$$= \begin{cases} \Gamma((R_1,A_1),\dots,(R_k,A_k))((M\backslash(x_1,\dots,x_n)) \\ \qquad\qquad\qquad\qquad \cup A(x_1,\dots,x_n)), \\ \qquad \text{if } x_1,\dots,x_n \in M \text{ and } R_i(x_1,\dots,x_n) \text{ for} \\ \qquad \text{some } 1 \le i \le k, \text{ or} \\ M, \quad \text{otherwise,} \end{cases}$$

where R_i and A_i, $1 \le i \le k$, are n-ary relations (the reaction conditions) and n-ary functions (the actions) on the elements of the multiset M, respectively. If some elements of the multiset M, say $x_1,\dots,x_n$, satisfy a reaction condition R_i for some i, $1 \le i \le k$, then the elements may be replaced by the result of the action $A_i(x_1,\dots,x_n)$, and the Γ function is applied on the resulting multiset again. This process continues until no elements satisfy any of the relations R_i, $1 \le i \le k$.

Example 3: To clarify this type of notation, consider the Gamma program for selecting the minimal element of a set of numbers.

$$\begin{aligned} minset(M) \;&=\; \Gamma(R,A)(M) \text{ where} \\ &\quad\; R(x,y) = (x < y), \\ &\quad\; A(x,y) = (x). \end{aligned}$$

IV. MEMBRANE SYSTEMS

Similarly to chemical programs, membrane systems (also called P systems) work with multisets of symbolic objects. They consist of a structured set of regions, each containing a multiset of objects and a set of evolution rules which define how the objects are produced, destroyed, or moved inside the system. A computation is performed by passing from one configuration to another one, applying the rules synchronously in each region.

We consider two variants in this paper: the rules are either multiset rewriting rules given in the form of $u \to v$, or communication rules of the form $(u, in; v, out)$ where u, v are finite multisets. In both cases, the rules are applied in the maximal parallel manner, that is, as many rules are applied in each region as possible. The end of the computation is defined by halting: the computation finishes when no more rules can be applied in any of the regions. The result is a number, the number of objects in a membrane labeled as output.

A *P system* of degree $n \ge 1$ is a construct

$$\Pi = (O, \mu, w_1, \dots, w_n, R_1, \dots, R_n, out)$$

where

- O is an alphabet of objects,
- μ is a membrane structure of the n membranes. The outmost membrane which is unique and usually labeled with 1, is called the skin membrane, and the membrane structure is denoted by a sequence of matching parentheses where the matching pairs have the same label as the membranes they represent, see Figure 1 for an example.
- $w_i \in \mathcal{M}(O)$, $1 \le i \le n$, are the initial contents of the n regions.
- R_i, $1 \le i \le n$, are the sets of evolution or communication rules associated to the regions, and
- $out \in \{1,\dots,n\}$ is the label of the output membrane.

As mentioned above, we consider two types of P systems in this paper. In the case of *rewriting P systems*, the rules

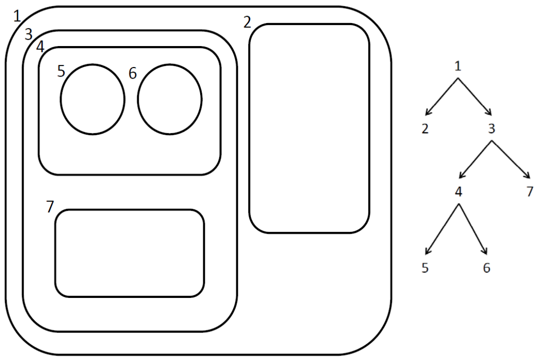

Fig. 1. A membrane structure with the skin membrane labeled as 1, and the corresponding tree representation. It can also be represented by a string of matching parentheses as $[_1\ [_2\ ]\ [_3\ [_4\ [_5\ ]\ [_6\ ]\ ]\ [_7\ ]\ ]\ ]$.

of the set R are of the form $u \to v$ where $u \in \mathcal{M}(O)$ and $v \in \mathcal{M}(O \times TAR)$ with $TAR = \{here, out\} \cup \{in_j \mid 1 \le j \le n\}$. In the case of *antiport P systems*, the rules don't allow the changing of the objects only their movement between the regions, they are of the form $(u, in; v, out)$ where $u, v \in \mathcal{M}(O)$.

The rules are applied in the non-deterministic, maximally parallel manner to the n-tuple of multisets of objects constituting the configuration of the system. For two configurations $C_1 = (u_1,\dots,u_n)$ and $C_2 = (v_1,\dots,v_n)$, we can obtain C_2 from C_1, denoted as $C_1 \Rightarrow C_2$, by applying the rules of $R_1,\dots,R_n$ in the following way.

In the case of rewriting P systems, the application of $u \to v \in R_i$ in the region i means to remove the objects of u from u_i and add the new objects specified by v to the system. The objects of v should be added to the regions as specified by the target indicators associated to them: If v contains a pair $(a, here) \in O \times TAR$, then a is placed in region i, the region where the rule is applied. If v contains $(a, out) \in O \times TAR$, then a is added to the contents of the parent region of region i; if v contains $(a, in_j) \in O \times TAR$ for some region j which is contained inside the region i (so region i is the parent region of region j), then a is added to the contents of region j.

In the case of antiport systems, the application of $(u, in; v, out) \in R_i$ in region i means to move the objects of u from the parent region into region i, and simultaneously, to move the objects of v into the parent region.

The n-tuple $(w_1,\dots,w_n)$ is the initial configuration of Π.

The objects evolve simultaneously, and the rules by which they evolve are chosen nondeterministically, but in a maximally parallel manner. This means, that in each region, objects are assigned to rules, nondeterministically choosing the rules and the objects assigned to each rule, but in such a way that no further rule can be applied to the remaining objects. A rule can be applied in the same step more than once, only the number of occurrences of objects matters. Objects which remain unassigned, appear unchanged in the next configuration.

A sequence of transitions between configurations is called a computation. A computation is successful if it halts, that is, if it reaches a configuration where no application of any of the rules are possible. In this case, the result is the multiset of objects which is present in the output region in the halting configuration.

As an example, let us consider how the dining philosopher problem can be represented in the P system framework.

Example 4: Consider the antiport P system

$$\Pi = (O, \mu, w_0, w_1, \ldots, w_5, R_0, R_1, \ldots, R_5, out)$$

where $O = \{t_i, e_i, f_i \mid 1 \leq i \leq 5\}$, $w_0 = e_1 f_1 \ldots e_5 f_5$, $w_i = t_i,\ 1 \leq i \leq 5$. The sets of rules are defined as $R_0 = \emptyset$, and for $1 \leq i \leq 5$ as

$$\begin{aligned} R_i \quad = \quad & \{(e_i f_i f_{i+1\ mod\ 5}, in; t_i, out),\ (e_i, in; e_i, out), \\ & (t_i, in; e_i f_i f_{i+1\ mod\ 5}, out),\ (t_i, in; t_i, out)\}. \end{aligned}$$

There are five regions, labeled by 1 to 5 in this system enclosed in a sixth one, the skin membrane, which is labeled by 0. The initial configuration, when the enclosed regions $i,\ 1 \leq i \leq 5$ contain the objects $t_i,\ 1 \leq i \leq 5$, respectively, corresponds to the situation when all philosophers are thinking. Applying the rule $(t_i, in; t_i, out)$, the ith philosopher may keep thinking, or applying the rule $(e_i f_i f_{i+1\ mod\ 5}, in; t_i, out)$, he may start eating.

As the properties of *parallel execution*, *mutual exclusion*, and *atomic capture* also hold in the case of membrane systems, the above given description of the dining philosophers also have the same desirable properties as the HOCL description given in the previous section.

Let us consider now an example from [12]. To this aim we introduce two features that we did not consider so far: *priorities* among the rules, and membrane *dissolution*.

Example 5: Let Π the following system of three membranes.

$$\Pi = (O, [_1\ [_2\ [_3\ ]\ ]\ ], \emptyset, \emptyset, af, R_1, R_2, R_3, 1)$$

where $O = \{a, b, d, e, f\}$, and the sets of rules are defined as

$$\begin{aligned} R_1 &= \{d \to \emptyset\}, \\ R_2 &= \{b \to d, d \to de\} \cup \{ff \to f > f \to \delta\}, \\ R_3 &= \{a \to ab, a \to b\delta, f \to ff\}. \end{aligned}$$

Priorities are introduced in the rule set R_2, denoted by the relation $ff \to f > f \to \delta$, which means that as long as the rule $ff \to f$ is applicable, $f \to \delta$ cannot be applied. Membrane dissolution is also introduced here by the symbol δ. When the rule $f \to \delta$ is used, the corresponding membrane (the membrane surrounding region 2 in this case) is dissolved/removed, and the objects it contains become elements of the parent region (region 1 in this case).

The computation of Π starts in the initial configuration $(\emptyset, \emptyset, af)$. Applying the rules $a \to ab$ and $f \to ff$ of R_3, we get $(\emptyset, \emptyset, abff)$ after one computational step. Repeating this for another $k-2$ steps, we get

$$(\emptyset, \emptyset, abff) \Rightarrow \ldots \Rightarrow (\emptyset, \emptyset, ab^{k-1}f^{2^{k-1}}).$$

Now, if we apply $a \to b\delta$ instead of $a \to ab$, the membrane delimiting region 3 is dissolved, so we arrive to the configuration $(\emptyset, b^k f^{2^k})$. Next, we can apply the rules of R_2 to the bs and fs, resulting in $(\emptyset, d^k f^{2^{k-1}})$ after the next step, by replacing all bs with ds and halving the number of fs in parallel. Note that as long as there are more than two f symbols, the rule $f \to \delta$ cannot be applied, because $ff \to f$ has higher priority. Applying $d \to de$ and $ff \to f$ as long as possible, we obtain

$$(\emptyset, d^k e^k f^{2^{k-2}}) \Rightarrow (\emptyset, d^k e^{2k} f^{2^{k-3}}) \Rightarrow \ldots \Rightarrow (\emptyset, d^k e^{(k-1)k} f)$$

and then $(d^k e^{kk})$ by applying the dissolution rule $f \to \delta$. Now if the rule of R_1 is applied erasing all d symbols, we obtain the configuration (e^{k^2}), and the system halts. The result of this computation is the number k^2. We can observe that the number k^2 can be computed by the system for any k in a similar manner, thus, the set of numbers computed by Π is the set $N(\Pi) = \{k^2 \mid k \geq 1\}$.

From a theoretical point of view membrane systems are both powerful and efficient computational devices. Powerful, as many of their variants are computationally complete, that is, equivalent in computational power to Turing machines, and efficient as, due to their parallelity and "chemical nature", they are able to provide efficient solutions to computationally hard (typically NP complete, or even PSPACE complete) problems. More details and further references can be found in [13].

V. Chemical programs and Membrane Systems

As we have seen in the previous sections, membrane systems and programs written in the Gamma language are closely related. This is not surprising because they both provide a realization of what we call the chemical paradigm of computing. They both work with symbolic chemical solutions which are represented by multisets, containing molecules which interact freely according to given reaction rules, resulting in a parallel, nondeterministic, distributed model. In this section we turn to the demonstration of links between the two formalisms.

A. Describing chemical programs by membranes systems

First we demonstrate how membrane systems could mimic the behavior of systems which are described by chemical programs. To this aim, we review the approach of [10], recalling an example from [1] which gives the chemical description of a mail system.

The mail system (see Figure 2) is described by a solution. Messages exchanged between clients are represented by basic molecules.

- Solutions named $ToSend_{d_i}$ contain the messages to be sent by the client i of domain d.
- Solutions named$Mbox_{d_i}$ contain the messages received by the client i of domain d.
- Solutions named $Pool_d$ contain the messages that the server of domain d must take care of.
- The solution named Network represents the global network interconnecting domains.
- A client i in domain d is represented by two active molecules $send_{d_i}$ and $recv_{d_i}$.
- A server of a domain d is represented by two active molecules put_d and get_d.

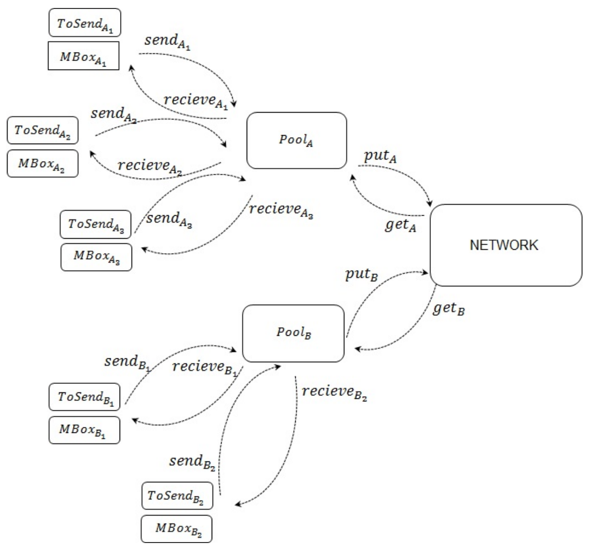

Fig. 2. The mail system described in Section V.

Clients send messages by adding them to the pool of messages of their domain. They receive messages from the pool of their domain and store them in their mailbox. Message stores are represented by sub-solutions.

The movement of messages are performed by reaction rules of the form

$$\textbf{replace } A : \langle msg, \omega_A\rangle,\ B : \langle\omega_B\rangle \quad \textbf{by } A : \langle\omega_A\rangle,\ B : \langle msg, \omega_B\rangle \textbf{ if } Cond.$$

The $send$ molecule sends the messages from the client to the pool, $recv$ gets the messages from the pool and places them inside the message box of the client, put forwards messages to the network, get receives messages from the network.

$$send_{d_i} = \textbf{replace } ToSend_{d_i} : \langle msg, \omega_t\rangle, Pool_d : \langle\omega_p\rangle \quad \textbf{by } ToSend_{d_i} : \langle\omega_t\rangle, Pool_d, : \langle msg, \omega_p\rangle$$

$$recv_{d_i} = \textbf{replace } Pool_d : \langle msg, \omega_p\rangle, MBox_{d_i} : \langle\omega_b\rangle \quad \textbf{by } Pool_d : \langle\omega_p\rangle, MBox_{d_i} : \langle msg, \omega_b\rangle \quad \textbf{if } recipient(msg) = i$$

$$put_d = \textbf{replace } Pool_d : \langle msg, \omega_p, \rangle, Network : \langle\omega_n\rangle \quad \textbf{by } Pool_d : \langle\omega_p\rangle, Network : \langle msg, \omega_n\rangle \quad \textbf{if } recipientDomain(msg) \neq d$$

$$get_d = \textbf{replace } Network : \langle msg, \omega_n\rangle, Pool_d : \langle\omega_p\rangle \quad \textbf{by } Network : \langle\omega_n\rangle, Pool_d : \langle msg, \omega_p\rangle \quad \textbf{if } recipientDomain(msg) = d$$

The solution representing the mail system contains the above described molecules together with sub-solutions representing the messages to be sent and received (called $ToSend$ and $MBox$, respectively) for each user, A_1, A_2, A_3, and B_1, B_2.

MailSystem:

$$\begin{aligned}\langle &send_{A_1}, recv_{A_1}, ToSend_{A_1} : \langle\ldots\rangle, MBox_{A_1} : \langle\ldots\rangle,\\ &send_{A_2}, recv_{A_2}, ToSend_{A_2} : \langle\ldots\rangle, MBox_{A_2} : \langle\ldots\rangle,\\ &send_{A_3}, recv_{A_3}, ToSend_{A_3} : \langle\ldots\rangle, MBox_{A_3} : \langle\ldots\rangle,\\ &put_A, get_A, Pool_A, Network, put_B, get_B, Pool_B,\\ &send_{B_1}, recv_{B_1}, ToSend_{B_1} : \langle\ldots\rangle, MBox_{B_1} : \langle\ldots\rangle,\\ &send_{B_2}, recv_{B_2}, ToSend_{B_2} : \langle\ldots\rangle, MBox_{B_2} : \langle\ldots\rangle\ \rangle\end{aligned}$$

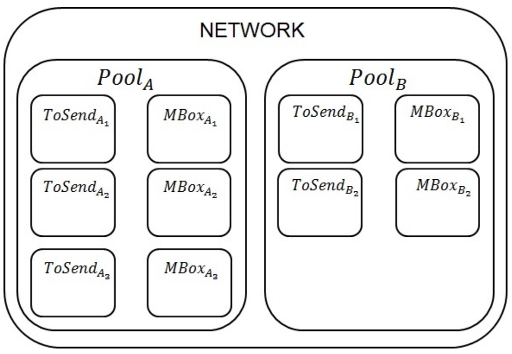

Fig. 3. The membrane mail system corresponding to the system of Figure 2

This chemical solution can be represented by a membrane system where message stores are represented by membranes, active molecules (or reactions) are represented by evolution rules. If we denote the regions as in Figure 3, and messages addressed to recipient d_i are represented by objects msg_{d_i}, we need the following rules:

- $R_{ToSend_{d_i}} = \{msg_{d_j} \to (msg_{d_j}, out)\}$,
- $R_{Pool_{d_i}} = \{msg_{d'_j} \to (msg_{d'_j}, out)$,
  $msg_{d_i} \to (msg_{d_i}, in_{MBox_i})\}$, and
- $R_{Network} = \{msg_{d_i} \to (msg_{d_i}, in_{Pool_d})\}$.

The rules corresponding to the "outbox" of users send the messages to their pool, and the rules corresponding to the pools, $Pool_A$ and $Pool_B$, place them into the $MBox$ of the user. If a message is addressed to a user belonging to the other pool, then it is sent to the network which forwards it to the corresponding message pool.

B. Describing membrane systems by chemical programs

Let us know continue with considerations in the "opposite direction", namely, with the study of how membrane computations could be described with a chemical program. To this aim, we summarize the results contained in [8].

First we introduce some elements of the γ-calculus of Banâtre and his coauthors, see for example [4]. Similarly to the chemical programming language used in the previous section, it is a higher order extension of the Gamma formalism. We need it in order to be able to have a calculus, a mathematically precise description of chemical computations.

The main rule of the calculus is the reaction rule

$$\gamma(P)[C].M, N \to \phi M$$

where P is a pattern, C is a condition, M is the result, and N is the multiset to which the rule is applied. Its application produces ϕM, where $match(P, N) = \phi$ and ϕ assigns values to variables in such a way that $\phi(C)$ is true.

Without further clarifications, let us look at the following example. The γ-term

$$\gamma(x, y)[x \leq y].y, (3, 4)$$

reduces to the multiset (4) since in order for $\phi(x \leq y)$ to hold, $match((x, y), (3, 4)) = \phi$ should be such, that $\phi = \{x \mapsto 3, y \mapsto 4\}$. This means that $\phi(y) = 4$, thus, the result of the reaction $\gamma(x, y)[x \leq y].y, (3, 4)$ is the multiset (4).

There calculus has several other reduction rules which we do not discuss here, but we do recall that a $replace$ operator can also be defined, which behaves similarly to the instruction with the same name used in the chemical programming language we saw in the previous sections. This is used in the following example which calculates the largest prime which is less than or equal to 6.

Example 6:

$largestprime(6) =$
let $sieve = replace\ (\langle x\rangle, \langle y\rangle)$ by $\langle x\rangle$ if x div y in
let $max = replace\ (\langle x\rangle, \langle y\rangle)$ by $\langle x\rangle$ if $x \leq y$ in
$\langle\ \langle\langle 2\rangle, \langle 3\rangle, \ldots, \langle 6\rangle, sieve\rangle, \gamma(\langle x\rangle)[true](x, max)\ \rangle$

The pattern standing in the last term $\gamma(\langle x\rangle)[true](x, max)$ is a solution $\langle x\rangle$, which can be only matched by inert solutions, thus, first the prime numbers are selected with the term $sieve$, producing the inert solution $\langle\langle 2\rangle, \langle 3\rangle, \langle 5\rangle, sieve\rangle$ which matches the pattern $\langle x\rangle$ in the γ-term, resulting in $\langle\langle 2\rangle, \langle 3\rangle, \langle 5\rangle, sieve, max\rangle$, and now $\max$ chooses the maximum among them, ending up with $\langle\langle 5\rangle, sieve, max\rangle$.

Using these ingredients, we can define a γ-term which is able to "simulate" the computations of a membrane system. More precisely, for each configuration C of a membrane system Π, we can define a term which contains the objects of the configuration together with *replace* operators corresponding to the rules of Π (and several technical details which we don't discuss) which enables the reduction in the calculus to proceed in such a way that it reproduces the results of the maximally parallel rule application of the P system. Namely, we have the following theorem, see [8].

Theorem 1: Let Π be a membrane system, and $C_0, C_1, \ldots, C_m$ be a sequence of configurations denoting a terminating computation.

Then there exists a γ-term $M(\Pi)$, and a reduction sequence $M(\Pi) \to M_1 \to \ldots \to M_s$, such that M_s cannot be further reduced, and if $C_m = (w_1, \ldots, w_n)$, then for all objects $a \in O$ and regions i, $1 \leq i \leq n$, M_s contains the same number of copies of (a, i), as w_i contains a.
In effect, the theorem establishes for the sequence $\Pi_0, \Pi_1, \ldots, \Pi_m$ of P-systems corresponding to the computation $C_0, C_1, \ldots, C_m$ starting from $\Pi = \Pi_0$ a sequence $M(\Pi) \to M_1 \to \ldots \to M_s$ of γ-terms such that there is an index set $0 < k_1 < \ldots < k_n = s$ with the property $M(\Pi_j) = M_{k_j}$ for $1 \leq j \leq n$.

VI. CONCLUSION

First we have briefly reviewed the chemical computing paradigm and the notion of membrane systems, then we have discussed their relationship by describing results from [10] and [8]. These results represent the first steps in the direction of establishing the relationship of the two paradigms. This approach could be interesting from several points of view. By being able to translate chemical programs to membrane systems, we could obtain a high level programming language for the description of membrane algorithms. By being able to describe membrane computations with a mathematically precise chemical calculus, we could use it to reason about the properties of membrane systems in a mathematical way.

ACKNOWLEDGMENT

The authors acknowledge the support of the European Union through the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project which is co-financed by the European Social Fund.

REFERENCES

[1] J.P. Banâtre, P. Fradet, and Y. Radenac, Higher-order chemical programming style. In: [2], 84–95.
[2] J.P. Banâtre, P. Fradet, J.L. Giavitto, and O. Michel, editors, *Unconventional Programming Paradigms, International Workshop UPP 2004, Le Mont Saint Michel, France, September 15-17, 2004, Revised Selected and Invited Papers*. Volume 3566 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, 2005.
[3] J.-P. Banâtre, P. Fradet, D. Le Métayer: Gamma and the chemical reaction model: Fifteen years after. In *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View*. Volume 2235 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2001, 17–44.
[4] J.P. Banâtre, P. Fradet, Y. Radenac, Principles of chemical computing. *Electronic Notes in Theoretical Computer Science* 124 (2005) 133–147.
[5] J.P. Banâtre, P. Fradet, Y. Radenac, Generalized multisets for chemical programming. *Mathematical Structures in Computer Science* 16(4) (2006), 557 – 580.
[6] J.P. Banâtre, D. Le Métayer, A new computational model and its discipline of programming. *Technical Report RR0566*, INRIA (1986).
[7] J.P. Banâtre, D. Le Métayer, Programming by multiset transformation. *Communications of the ACM* 36 (1993), 98–111.
[8] P. Battyányi, Gy. Vaszil, Describing membrane computations with a chemical calculus. *Fundamenta Informaticae* 134 (2014), 39–50.
[9] G. Berry, G. Boudol, The chemical abstract machine. *Theoretical Computer Science* 96 (1992), 217–248.
[10] M. Fésüs, Gy. Vaszil, Chemical programming and membrane systems. In: *Proc. 14th International Conference on Membrane Computing*, Institute of Mathematics and Computer Science, Academy of Moldova, Chişinău, 2013, 313–316.
[11] Gh. Păun, Computing with membranes. *Journal of Computer and System Sciences* 61 (2000), 108–143.
[12] Gh. Păun, Membrane Computing - An Introduction. Springer-Verlag, Berlin 2002.
[13] Gh. Păun, G. Rozenberg, A. Salomaa (eds): The Oxford Handbook of Membrane Computing, Oxford University Press (2010)
[14] G. Rozenberg, A. Salomaa (eds): Handbook of Formal Languages, Springer Berlin (1997)
[15] A. Salomaa: Formal Languages, Academic Press, New York (1973)

Péter Battyányi received his MSc in mathematics in 1997 at the University of Debrecen. He conducted his PhD studies in 2005-2007 at the Université de Savoie, France. Since 2008, he is an assistant professor at University of Debrecen. His main fields of interest are mathematical logic, logical calculi, computability, artificial intelligence, formal verification of programs, models of parallel programming.

György Vaszil is head of the Department of Computer Science at the Faculty of Informatics of the University of Debrecen. He received his PhD at the Eötvös Loránd University, Budapest, in 2001, and the title Doctor of the Hungarian Academy of Sciences in 2014. His research interests include formal languages and automata theory, unconventional and nature motivated computational models and architectures.