Computer Science Journal of Moldova, vol.**20**, no.3(60), 2012

Static and Dynamic Membrane Structures

Sergiu Ivanov

Abstract

While originally P systems were defined to contain multiset rewriting rules, it turned out that considering different types of rules may produce important results, such as increasing the computational power of the rules. This paper focuses on factoring out the concept of a membrane structure out of various P system models with the goal of providing useful formalisations. Both static and dynamic membrane structures are considered.

Keywords: Computing model, P system, membrane structure, semi-lattice, active membranes.

1 Introduction

P systems are computational models inspired from the structure of living cells, introduced by Gh. Păun in 1998 [1]. The principal idea behind this model is that the chemical reactions happening in a biological cell can be interpreted as applications of rewriting rules to multisets of objects. Since formal grammars can be treated as computational devices, a cell can be basically viewed as a collection of compartments, each hosting computation. Further, communication between compartments is allowed, which binds the computing devices into a network where information is produced and consumed to be eventually combined into the final result. For a more thorough introduction to the subject the reader may turn to [2].

One of P systems types which is commonly brought about in examples is the transitional P systems [3]. In transitional P systems, the compartments of P systems hold multiset rewriting rules. It has been shown (see Chapter 4 of [4] for a summary) that membrane structure

does not add any computational power to what is already provided by the class of multiset rewriting rules in use. The idea behind this result is simple: since a membrane structure is finite and static in this case, it can very well be dropped by considering "labelled" symbols: instead of having a in compartment 1, have the symbol a_1, for example. In this way, one can simulate any communication between the computing compartments which can happen in a transitional P system and which could enhance the overall computational power.

While this conclusion may look rather disconcerting in what concerns the utility of transitional P systems, static membrane structures may actually be rather significant in certain situations. The authors of [5] show that, if one places insertion-deletion rules in the compartments of a membrane structure, one obtains a computational device which is more powerful than the class of insertion-deletion rules in use. In fact, this is not the only well-known example of placing other types of rules in compartments of membrane structures; consider, for example, splicing P systems (Chapter 8 of [4]) and P systems with string objects (Chapter 7 of [4]). Note that in these cases, the rules placed in the compartments of the membrane system do make sense outside of the context of membrane structures. I find it necessary to explicitly contrast this with communication P systems (Chapter 5 of [4]) and P systems with active membranes (Chapter 11 of [4]), in which cases the investigated rules seem to be very intimately connected with the membrane structure itself.

The reasoning exposed in the previous paragraph brings attention to the membrane structure, rather than to the P system that results from combining a membrane structure and rules. Some basic formal representations are widely used in which membrane structures are considered as rooted trees [3, 4]. However, as this paper shows, the underlying tree of a membrane structure is a skeleton which, while being essential, is far from covering all the features associated with the membranes. Further note that, while formalising static membrane structures is an interesting and useful task in itself, it is the dynamic membrane structures arising in different flavours of P systems with active membranes that are the most attractive object of formalisation.

This paper focuses on studying membrane structures as separate objects, apart from the containing context of P systems. An approach to formalising static and dynamic membrane structures as algebraic structures is suggested, and then applications of the obtained formalisation are shown.

2 Preliminaries

2.1 Multisets

Given a finite set A, by $|A|$ one understands the number of elements in A.

Let V be a finite alphabet; then V^* is the set of all finite strings of a V, and $V^+ = V^* - \{\lambda\}$, where λ is the empty string. By $\mathbb{N}$ one denotes the set of all non-negative integers, by $\mathbb{N}^k$ – the set of all vectors of non-negative integers.

Let V be a finite set, $V = \{a_1, \ldots, a_k\}$, $k \in \mathbb{N}$. A *finite multiset* M over V is a mapping $M : V \to \mathbb{N}$. For each $a \in V$, $M(a)$ indicates the number of "occurrences" of a in M. The value $M(a)$ is called the *multiplicity* of a in M. The size of the multiset M is $|M| = \sum_{a \in V} M(a)$, i.e., the total count of the entries of the multiset. A multiset M over V can also be represented by any string x which contains exactly $M(a_i)$ instances of a_i, $1 \leq i \leq k$. The support of M is the set $supp(M) = \{a \in V \mid M(a) \geq 1\}$, which is the set which contains all elements of the multiset. For example, the multiset over $\{a, b, c\}$ defined by the mapping $\{(a, 3), (b, 1), (c, 0)\}$ can be written as a^3b. The support of this multiset is $\{a, b\}$.

Let x, y be two multisets over V. Then x is called a *submultiset* of y, written as $x \subseteq y$, if and only if $\forall a \in V \,.\, x(a) \leq y(a)$. The union of x and y, denoted by $x \uplus y$ is defined in the following way: $\forall a \in V \,.\, (x \uplus y)(a) = x(a) + y(a)$. The difference of x and y, denoted by $x \backslash y$, is defined similarly: $\forall a \in V \,.\, (x \backslash y)(a) = x(a) - y(a)$.

2.2 P Systems

A *transitional* membrane system is defined by a tuple (Chapter 1 of [4])

$$\begin{array}{lcl} \Pi & = & (O, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m, i_0), \text{ where} \\ O & & \text{is a finite set of objects,} \\ \mu & & \text{is a hierarchical structure of } m \text{ membranes, bijectively} \\ & & \text{labelled with } 1, \ldots, m, \\ w_i & & \text{is the initial multiset in region } i, 1 \leq i \leq m, \\ R_i & & \text{is the set of rules of region } i, 1 \leq i \leq m, \\ i_0 & & \text{is the output region.} \end{array}$$

The rules have the form $u \to v$, where $u \in O^+$, $v \in (O \times Tar)^*$. The target indications from $Tar = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$, where j are the labels of the corresponding inner membranes. The target *here* is typically omitted. In case of non-cooperative rules, $u \in O$.

The rules are applied in a maximally parallel way: no further rule should be applicable to the idle objects. In the case of non-cooperative systems, all objects evolve by the associated rules in the corresponding regions (except objects a in regions i such that R_i does not contain any rule $a \to u$, but these objects do not contribute to the result). Rules are non-deterministically chosen at each moment in time when a change occurs in the configuration of the P system. The process of choosing which rules should be applied does not take any time.

A P system with *active membranes* is defined by a tuple (Chapter 11 of [4]):

$$\begin{array}{lcl} \Pi & = & (O, H, E, \mu, w_1, w_2, \ldots, w_m, R, i_0), \text{ where} \\ O & & \text{is a finite set of objects,} \\ H & & \text{is the alphabet of names of membranes,} \\ E & & \text{is the set of electrical charges,} \\ \mu & & \text{is the initial hierarchical structure of } m \text{ membranes,} \\ & & \text{bijectively labelled by } 1, \ldots, m; \end{array}$$

w_i	is the initial multiset in region $i, 1 \leq i \leq m$,
R	is the set of rules,
i_0	is the output region.

The rules in P systems with active membranes can be of the following five basic types:

(a) $[a \to v]^e_h$, $h \in H$, $e \in E$, $a \in O$, $v \in O^*$;

(b) $a[\,]^{e_1}_h \to [b]^{e_2}_h$, $h \in H$, $e_1, e_2 \in E$, $a, b \in O$;

(c) $[a]^{e_1}_h \to [\,]^{e_2}_h b$, $h \in H$, $e_1, e_2 \in E$, $a, b \in O$;

(d) $[a]^e_h \to b$, $h \in H \backslash \{s\}$, $e \in E$, $a, b \in O$;

(e) $[a]^{e_1}_h \to [b]^{e_2}_h [c]^{e_3}_h$, $h \in H \backslash \{s\}$, $e_1, e_2, e_3 \in E$, $a, b, c \in O$.

It is often considered that $E = \{0, -, +\}$. The rules apply to elementary membranes, i.e., membranes which do not contain other membranes inside.

The rules are applied in the usual non-deterministic maximally parallel manner, with the following details: any object can be subject of only one rule of any type and any membrane can be subject of only one rule of types (b)–(e). Rules of type (a) are not counted as applied to membranes, but only to objects. This means that when a rule of type (a) is applied, the membrane can also evolve by means of a rule of another type. If a rule of type (e) is applied to a membrane, and its inner objects evolve at the same step, it is assumed that first the inner objects evolve and then the division takes place, so that the result of applying rules inside the original membrane is replicated in the two new membranes.

2.3 Semilattices

A binary relation $\leq$ is a partial order if it is reflexive, symmetric, and transitive. A set $(S, \leq)$ endowed with such a binary relation is called

a partially ordered set. If $x, y \in S$ such that $(x, y) \notin \leq$, the elements x and y are called incomparable; this is written as $x \not\leq y$. The *interval* between two comparable elements $x, y \in L$, denoted by $[x, y]$ is the set of all elements in L which are "between" x and y:

$$\forall x, y \in S \,.\, x \leq y \,.\, [x, y] \stackrel{def}{=} \{a \in L \mid x \leq a \text{ and } a \leq y\}$$

An interval is called simple if it only includes its "endpoints":

$$\forall x, y \in L \,.\, [x, y] \text{ – simple } \stackrel{def}{\iff} [x, y] = \{x, y\}.$$

In this case x is called the predecessor of y (or y – the successor of x), which is denoted by $x \prec y$.

A partially ordered set $(S, \leq)$ is a *meet-semilattice*, if for any $x, y \in S$ the greatest lower bound $x \wedge y$ (the meet) of the two exists:

$$\forall x, y \in S \,.\, \exists x \wedge y \in S \,.\, \forall z \in S \,.\, (z \leq x \text{ and } z \leq y) \Longrightarrow z \leq x \wedge y.$$

Dually, one defines the join-semilattice. A partially ordered set $(S, \leq)$ is a *join-semilattice*, if for any $x, y \in S$ the least upper bound $x \vee y$ (the join) of the two exists:

$$\forall x, y \in S \,.\, \exists x \vee y \in S \,.\, \forall z \in S \,.\, (x \leq z \text{ and } y \leq z) \Longrightarrow x \vee y \leq z.$$

Any of these two can be defined as an algebraic structure. For example, a meet-semilattice is the structure $(S, \wedge)$ in which the binary operation is idempotent, commutative, and associative:

$$(S, \wedge)\text{– semilattice} \stackrel{def}{\iff} \begin{array}{ll} \forall x, y, z \in S \,. & x \wedge x = x \\ \text{and} & x \wedge y = y \wedge x \\ \text{and} & x \wedge (y \wedge z) = (x \wedge y) \wedge z. \end{array}$$

3 Static Membrane Structures

3.1 Construction of Static Membrane Structures

Consider a finite meet-semilattice $(L, \wedge)$ with the properties that the semilattice includes the minimal element, denoted by 0:

$$\exists 0 \in L \,.\, \forall x \in L \,.\, 0 \leq x, \tag{1}$$

and that any element of L except 0 has only one predecessor:

$$\forall x \in L\backslash\{0\} \,.\, \exists! y \in L \,.\, y \prec x. \tag{2}$$

The following lemma shows that finite semilattices with these two properties are essentially trees.

Lemma 1. *Let $(L, \wedge)$ be a finite meet-semilattice. Consider the graph $G = (V, E)$ with vertexes all elements of L and edges all corresponding simple intervals:*

$$V \;=\; L, \qquad E \;=\; \{(x, y) \in L \times L \mid x \prec y\},$$

If $(L, \wedge)$ has the properties (1) *and* (2)*, then G is a tree.*

Proof. Let $n = |L| = |V|$ be the number of elements in the set $L = V$. Since any element $a \in L\backslash\{0\}$ has exactly one predecessor, the count of edges in G is $|E| = n - 1$. Further, G is connected, because $\forall x \in L = V \,.\, 0 \leq x$, which means that there exists a sequence of elements $(x_i)_{i=1}^{m} \subseteq L$, $m \in \mathbb{N}$, such that

$$0 = x_1 \prec x_2 \prec \ldots \prec x_m = x,$$

which gives the path in G connecting 0 and x. Since G is a connected graph in which $|E| = |V| - 1$, G is a tree [7]. □ □

In particular, if L satisfies the properties (1) and (2), then L contains no meets for any incomparable elements: $\forall x, y \in L \,.\, x \not\leq y \Longrightarrow x \wedge y \notin L$.

For a set S and a set H, a mapping $l : S \to H$ will be called a *labelling* of S with the label set H. Note that l is not required to be injective, which means that several objects in S may have the same label.

Definition 1. *The following tuple will be called a* membrane structure*:*

$$\begin{array}{rcl} M & = & ((L, \wedge), H, l)\,, \textit{ where} \\ (L, \wedge) & & \textit{is a meet-semilattice with the properties } (1) \textit{ and } (2), \\ H & & \textit{is a set of labels}, \\ l & & \textit{is a labelling of } L \textit{ with } H. \end{array}$$

The elements of L will be called membranes*.*

It is easy to see that a membrane structure in this definition is exactly the same thing as what is defined in numerous articles on P systems (for example, Chapter 1 of [4]). The important part is that the meet-semilattice $(L, \wedge)$ was shown to be a tree. The set of labels H and the corresponding labelling l obviously corresponds to the usual labelling of membranes.

Example 1. *Consider the structure*

$$[\,[\,[\;\;]_3\,]_2[\;\;]_4\,]_1,$$

in which the membrane with label 1 *contains a membrane with label* 4 *and a membrane with label* 2*, which, in its turn, contains a membrane with label* 3*, will be translated to the membrane structure* $M = ((L = \{a, b, c, d\}, \wedge), H = \{1, 2, 3, 4\}, l, \varnothing)$*, where* $l(a) = 1$*,* $l(b) = 2$*,* $l(c) = 3$*,* $l(d) = 4$*, and the partial order on* L *is given by the following set of pairs:*

$$\leq = \{(a, b), (a, d), (a, c), (b, c)\}\,.$$

$(L, \wedge)$ *satisfies the properties* (1) *and* (2)*. Indeed,* $\forall x \in L \,.\, a \leq x$*, thus* $a = 0$ *in the terminology introduced in this section. Further, it is easy to check that each element in* L*, except for* a*, has exactly one predecessor. Thus,* M *is a valid membrane structure.*

It should be clear now that, if $x, y \in L$ and $x \prec y$, then x is the parent membrane of y.

Note that while the definition given in this paper generalises the majority of other definitions of tree-like membrane structures, it does not cover much more than what is covered by the said definitions. Thus, the notion of membrane structure as introduced in the present paper is sufficiently narrow.

Remark that there has not been any mentioning of the environment, which is sometimes considered as a compartment with some limitations (Chapter 1 of [4]). It is easy, however, to extend the semi-lattice $(L, \wedge)$ by adding an element $0'$ with the property that $\forall x \in L \,.\, 0' \leq x$ to represent the environment.

Also note that a join-semilattice could have been chosen instead of a meet-semilattice. Obviously, any reasoning about membrane structures

considered as meet-semilattices can be converted to join-semilattices by substituting the word "meet" for "join", $\wedge$ for $\vee$ and reversing the direction of comparisons.

Finally, I would like to discuss the usefulness of the new formalisation. While it has been shown that the principal component of a membrane structure, the semilattice $(L, \wedge)$, is always a tree, the advantage of this approach is that a membrane structure is defined as an algebraic structure, which makes it easier to define morphisms, as will be shown in the concluding sections of this paper.

3.2 Construction of P Systems with Static Membrane Structures

Consider a finite set O and a set of rules R over this alphabet. No other restrictions on the two sets are imposed, i.e., any type of rules over O is allowed. Define the application $\sigma : R \times O \to O \cup \{\perp\}$, $\perp \notin O$, in the following way: if a rule $r \in R$ is applicable to an object $o \in O$, then $\sigma(r, o)$ is the result of application of r to o. If r is not applicable to o, $\sigma(r, o)$ is defined to be $\perp$. The terms "applicable" and "application" are expected to be defined during the construction of the sets O and R. For the purposes of this article, the inner structure of the rules and objects is inessential, as long as some basic statements can be asserted about either of them.

Consider a membrane structure $M = ((L, \wedge), H, l, A, a)$ and two labellings of L: $object : L \to O$ and $rules : L \to 2^R$, where 2^X is the set of all subsets (the power set) of X. Setting up such labellings can be intuitively perceived as creating a system of nested compartments, with an object and rules in each compartment. Note that since no restriction has been imposed on O, an object may be anything, including a set, a multiset, a string, a set of strings, etc.

Further, introduce the function $outer : L \to L \cup \{\perp\}$, which yields the containing membrane for the given membrane, or $\perp$ if the argument is 0:

$$\begin{aligned} m \in L \backslash \{0\} \Longrightarrow \quad & outer(m) \stackrel{def}{=} p, p \prec m; \\ & outer(0) \stackrel{def}{=} \perp . \end{aligned}$$

Similarly, the function $inner : L \to 2^L$ yields the immediately inner membranes of the given membrane:

$$inner(m) \stackrel{def}{=} \{c \in L \mid m \prec c\}.$$

To simplify further expressions, the convenience function $adjacent : L \to 2^L$ will be introduced:

$$\begin{array}{rcl} m \in L\backslash\{0\} \Longrightarrow adjacent(m) & \stackrel{def}{=} & inner(m) \cup outer(m); \\ adjacent(0) & \stackrel{def}{=} & inner(0). \end{array}$$

Now define two applications $iLabels, oLabels : L \times R \to 2^H$ in the following way: if $m \in L$ and $r \in rules(m)$, then $iLabels(m, r)$ is the set of input labels for the rule r in membrane m, and $oLabels(m, r)$ is the set of output labels for r. If $r \notin rules(m)$, then $iLabels(m, r) \stackrel{def}{=} oLabels(m, r) \stackrel{def}{=} \varnothing$. These functions annotate a rule with the information about the labels of the membranes whose contents it may use or modify. To ensure the validity of the labels in the context of the membrane structure, one defines the function $validLabels : L \times 2^H \to 2^H$ in the following way:

$$validLabels(m, H') \stackrel{def}{=} H' \cap \{l(b) \mid b \in adjacent(m)\}.$$

Thus, *validLabels* insures that a set of labels only contains the labels of the outer and inner membranes of m, enforcing the well-known pattern of communication along the tree in P systems.

Finally, define the applications

$$\begin{array}{rcccll} buildInput & : & L \times R & \to & O & \cup\{\perp\}, \\ outputBuilder & : & L \times R & \to & \hom_{\mathbf{Set}}(O \times L, \quad O) & \cup\{\perp\}. \end{array}$$

where $\hom_{\mathbf{Set}}(A, B)$ is the set of applications between the sets A and B.

To understand the meaning of the last two applications, consider again a membrane $m \in L$, and a rule r in the associated set of rules

$rules(m)$. $buildInput(m, r)$ constructs the objects belonging to the compartments the rule r depends on:

$$\{object(m) \mid m \in adjacent(r) \text{ and } l(m) \in iLabels(m, r)\},$$

then "combines" these objects and $object(m)$. The meaning of the verb "combine" should be defined in the description of the rules R and how they act on the objects in O.

The value $outputBuilder(m, r)$ is a function $f : O \times O \to O$ which, for an object o and a membrane $b \in adjacent(m)$, returns the "combination" of the object o with $object(b)$, or produces other modifications to $object(b)$. Again, the term "combination" should be defined in the description of the rules R and of how they act on the objects in O.

In the case when r does not belong to the set of rules associated with m, the last two applications take the value $\perp$:

$$\begin{array}{lll} r \notin rules(m) \implies & buildInput(m, r) & \stackrel{def}{=} \perp \\ \text{and} & outputBuilder(m, r) & \stackrel{def}{=} \perp \end{array}$$

If some input conditions are not satisfied in $buildInput$, this function should take the value $\perp$.

Definition 2. *The following construction will be referred to as a* P system with static (tree-like) membrane structure*:*

$$\begin{array}{rcl} \Pi & = & (M, O, R, \sigma, object, rules, iLabels, \\ & & oLabels, buildInput, outputBuilder, i_0), \end{array}$$

where $i_0 \in H$ is the label of the output membrane (s).

Similarly to the usual definition, a configuration $C : L \to O$ of Π is the collection of the contents of the compartments, indexed by membranes: $C(m) = object(m)$.

Before proceeding to extending the formalisation to the semantics of the P systems, an example would be helpful in showing how the static structure of familiar constructs of P systems maps to the definition given in the current paper.

Example 2. *Consider a transitional P system*

$$\Pi' = (O', \mu, w_1, w_2, \ldots, w_n, R_1, R_2, \ldots, R_n, i_0).$$

In the previous sections it has already been shown how μ maps to the semilattice $(L, \wedge)$. The set of labels H is the set of numbers 1 through n: $H = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$ and the (bijective) labelling l is defined in the obvious way.

The set of objects O is the set of multisets over $O'' = O' \cup \{(o,t) \mid o \in O' \text{ and } t \in Tar\}$. The set of rules R contains all multiset rewriting rules over the alphabet O'', whose left-hand sides do not include target indications:

$$R = \{u \to v \mid u \in O'^* \text{ and } v \in O''^*\},$$

where X^* was used to denote the set of multisets over X. The application σ carries out the usual application of a multiset rewriting rule to a multiset. The labelling object associates to the membrane labelled with i, $1 \leq i \leq m$, the multiset w_i. Similarly, the labelling rules associates to the membrane with label i, $1 \leq i \leq n$, the set of rules R_i.

The application $iLabels$ takes the value $\varnothing$ for any valid combination of arguments. For $m \in L$ and $r \in rules(m)$, the function $oLabels(m, r)$ is the set of labels mentioned in target indications of the right-hand side of the rule r, excluding the label of m. The application $buildInput$ is trivially defined as $buildInput(m, r) = object(m)$.

The value $f : O \times L \to O$ of $outputBuilder(m, r)$ is defined in the following way. For every $b \in \{b \in L \mid b \in adjacent(m) \text{ and } l(b) \in oLabels(m, r)\}$, and an object $o \in O$, $f(o, b)$ will result in multiset union of $object(b)$ and the multiset of all objects of o with target indications $l(b)$. The value $f(o, m)$ will result in constructing a multiset o' by subtracting the left-hand side of r from $object(m)$ and then performing multiset union of o' and the multiset of objects of o which have no target indications or have the indication here. For all other membranes x, the value of the function is trivially defined as $f(o, x) = object(x)$. Thus, $buildObject$ distributes the symbols across the corresponding membranes.

3.3 Computation in P Systems with Static Membrane Structure

With the necessary tools set up, it is now possible to completely describe how a P system with static membrane structure, as defined in this paper, transitions from one configuration into another configuration. This will eventually make it possible to define computation.

The reasoning exposed in this section is loosely based on the considerations in [8], which provides a different approach to generalising P systems with static membrane structures, whereby the tree-like membrane structure is almost wholly dismissed.

Consider a P system Π, as defined in the previous section. Remark that different configurations of Π are given by different mappings $C = object$. To avoid confusion, as well as to specify the origin of the corresponding functions, subscripts will be henceforth supplied which show which P system and which configuration thereof is being considered.

Define the function $applyRule_{\Pi,C} : L \times R \to \hom_{\mathbf{Set}}(L, O) \cup \{\perp\}$. Its purpose is to produce a new configuration by applying a rule associated to a membrane. For $m \in L$, $r \in rules_{\Pi}(m)$, under the conditions that $buildInput_{\Pi,C}(m, r) \neq \perp$ and $\sigma(r, buildInput_{\Pi}, C(m, r)) \neq \perp$, $applyRule_{\Pi,C}$ is defined as follows:

$$\begin{aligned} applyRule_{\Pi,C}(m, r)(b) &\stackrel{def}{=} doOutput(result, b), \text{ where} \\ result &\stackrel{def}{=} \sigma(r, buildInput_{\Pi,C}(m, r)), \\ doOutput &\stackrel{def}{=} outputBuilder_{\Pi,C}(m, r). \end{aligned}$$

If the enumerated conditions are not satisfied, $applyRule_{\Pi,C}(m, r) = \perp$.

According to this definition, $applyRule_{\Pi,C}(m, r)$ is a function which maps every membrane to the objects contained within, after the application of the rule $r \in rules_{\Pi}(m)$. If applying the rule is not possible, $applyRule_{\Pi,C}(m, r)$ takes the special signal value $\perp$.

Note that, while the description of the process of applying a rule by σ is done rather generally and informally, quite a bit of effort is invested into specifying the modifications induced by the associated membrane structure in as detailed a way as possible.

Definition 3. *A rule $r \in rules_\Pi(m)$, for an $m \in L$, is said to be applicable in the configuration C if $applyRule_{\Pi,C}(m,r) \neq \perp$.*

In a given configuration given by the mapping $C = object$, the set of applicable rules is defined as

$$\begin{aligned} applicableRules(\Pi, C) &\stackrel{def}{=} \{r \in rules_\Pi(m) \mid m \in L \\ &\qquad \text{and } applyRule_{\Pi,C}(m,r) \neq \perp\}. \end{aligned}$$

It is would now be desirable to construct the analog of the marking algorithm introduced in [8]. To do this, it should be remarked that an application of a rule $r \in R$ is made possible because certain "premises" are satisfied. The action of applying r may entail removal of some of these premises. To account for this, define the application

$$premisesEraser_{\Pi,C} : L \times R \to \hom_{\mathbf{Set}}(O \times L, O) \cup \{\perp\},$$

which, in parallel to $outputBuilder_{\Pi,C}$, produces a function which removes, if possible, the premises which made the rule $r \in rules_\Pi(m)$, $m \in L$, applicable. These considerations lead to the definition of the application $erasePremises_{\Pi,C} : L \times R \to \hom_{\mathbf{Set}}(L, O) \cup \{\perp\}$, in parallel to $applyRule_{\Pi,C}$:

$$\begin{aligned} erasePremises_{\Pi,C}(m,r)(b) &\stackrel{def}{=} doErase(result, b), \text{ where} \\ result &\stackrel{def}{=} \sigma(r, buildInput_{\Pi,C}(m,r)), \\ doErase &\stackrel{def}{=} premisesEraser_{\Pi,C}(m,r). \end{aligned}$$

This definition is valid when $r \in rules_\Pi(m)$, $m \in L$. If this does not hold, or if $buildInput_{\Pi,C}(m,r) = \perp$, or if $\sigma(r, buildInput_{\Pi,C}(m,r)) = \perp$, then $erasePremises_{\Pi,C} \stackrel{def}{=} \perp$.

The are now sufficient instruments to construct the marking algorithm. Consider a multiset ρ of pairs rules and the corresponding membranes:

$$\rho = \{((m,r), n) \mid m \in L \text{ and } r \in rules_\Pi(m) \text{ and } n \in \mathbb{N}\}.$$

Define the function

$$isApplicableMultiset_{\Pi} : \hom_{\mathbf{Set}}(L, O) \times (L \times R)^* \to \{\texttt{true}, \texttt{false}\}$$

to be `true` if all rules in ρ can be applied the corresponding number of times in the supplied configuration and to be `false` otherwise:

$$\begin{aligned} isApplicableMultiset_{\Pi}(C, \lambda) &\stackrel{def}{=} \texttt{true} \\ (m, r) \in \rho \Longrightarrow & \\ isApplicableMultiset_{\Pi}(C, \rho) &\stackrel{def}{=} r \in applicableRules(\Pi, C) \\ \text{and} \quad & isApplicableMultiset_{\Pi}(C', \rho'), \\ \text{where } C' &\stackrel{def}{=} erasePremises_{\Pi,C}(m, r), \\ \rho' &\stackrel{def}{=} \rho \backslash \{(m, r)\}. \end{aligned}$$

Here λ was used to denote the empty multiset.

The function $isApplicableMultiset_{\Pi}$ essentially performs the same procedure as does the marking algorithm in [8]. It checks the applicability of every rule in the multiset ρ and removes the rules found applicable one by one. If the multiset becomes empty, the conclusion is drawn that all rules in ρ can be applied the corresponding number of times in the current configuration. Otherwise, the function is `false`.

Once the multiset of membranes and rules ρ has been decided to be applicable, the rules in ρ may obviously be applied one by one, by invoking $applyRule_{\Pi,C}$ for all of them. Thus, the basic semantics has been constructed. Further definitions provided in [8] like, for example, derivation modes, halting conditions, etc., can be easily adapted to the algorithms described in this section, which eventually completes the formalisation of P systems with static (tree-like) membrane structure.

4 Dynamic Membrane Structures

4.1 Construction of P systems with Dynamic Membrane Structure

In this section the definition of a membrane structure will be extended to cover the dynamic membrane structures arising in P systems with active membranes, for example.

Definition 4. *The following tuple will be called a (dynamic)* membrane structure*:*

$$\begin{array}{rcl} M & = & ((L,\wedge),H,l,A,a)\text{, where} \\ (L,\wedge) & & \text{is a meet-semilattice with the properties (1) and (2),} \\ H & & \text{is a set of labels,} \\ l & & \text{is a labelling of } L \text{ with } H, \\ A & & \text{is a set of attributes,} \\ a & & \text{is a labelling of } L \text{ with } A. \end{array}$$

If $A = \varnothing$, by convention, the last two components of the tuple will not be written. Thus, the definition introduced in Subsection 3.1 can be regarded as a special case of this definition.

The need for the set of attributes arises from the fact that, in P systems with active membranes, the membranes sometimes carry charge (Chapter 11 of [4]). To model this feature, one can define $A = \{0, -, +\}$; then, for a membrane $m \in L$, $a(m) \in A$ will give the charge.

In the previous parts of the paper it has been shown how the membrane structure M, together with the sets yielded by $iLabels$ and $oLabels$, directs the way rule applications happen. However, as it can be seen in Subsection 2.2, rules that influence the membrane structure itself are in very tight connection with the membranes, which makes it quite difficult to construct the parallel to the mappings $iLabels$ and $oLabels$ which would indicate how a rule acts on the membrane structure. A possible solution is to even further decouple the action of a rule on a membrane structure for the nature of the rule itself. More concretely, a rule in a P system with dynamic membrane structure will be written as two rules: a rule which works as described in the Subsection 3.3, and another rule, acting on the membrane structure. The coming paragraphs will provide further details, as well as a formal explanation.

In order to better describe the semantics of dynamic membrane structure, the reasoning will start in the frame of a P system with the (yet static) membrane structure M, as defined in Subsection 3.2. Thus,

consider the P system

$$\begin{aligned}\Pi \;=\; & (M, O, R, \sigma, object, rules, iLabels,\\ & oLabels, buildInput, outputBuilder).\end{aligned}$$

To benefit from the attributes in M, for a membrane $m \in L$ and an associated rule $r \in rules(m)$, define the application $contextChecker : L \times R \to \hom_{\mathbf{Set}}(A, \{\texttt{true}, \texttt{false}\})$ in the following way: $contextChecker(m, r)$ is a function, which checks the attributes of the membrane r, and decides whether the context is "suitable" or not. The meaning of this function will become clearer in the next section.

Fix a membrane $m \in L$ and a rule $r \in rules(m)$ associated with it. Consider the set of pairs of labels and attributes, valid in the context of the membrane m:

$$labsAttrs(m) \stackrel{def}{=} \left\{ \big(l(m'), attr\big) \mid m' \in adjacent(m) \text{ and } attr \in A \right\}.$$

Definition 5. *A* membrane structure rule *in the context of a membrane m is a multiset rewriting rule of the form $u \to v$, where $u, v \in labAttrs(m)^*$, where X^* was used to denote the set of all multisets over X.*

The set of membrane structure rules valid in the context of a given membrane m is given by the application $validMSRules : L \to labAttrs(m)^*$ naturally defined as

$$validMSRules(m) \stackrel{def}{=} \{u \to v \mid u, v \in labAttrs(m)^*\}.$$

The set of all valid membrane structure rules is defined in the following way:

$$allMSRules \stackrel{def}{=} \bigcup_{m \in L} validMSRules(m).$$

What a membrane structure rule is should become clear from an informal example.

Example 3. *Consider the construction* $[\ [\ \ ]_2^+[\ \ ]_3^-\ ]_1^+$. *Then*

$$(2,+)(3,-) \to (2,+)(3,-)(2,-)$$

is a valid membrane structure rule for the membrane with label 1. *From the notation, it should be intuitively understood that this rule produces a new membrane with label* 2 *and charge "−" if the membrane* 1 *contains a membrane* 2 *with charge "+" and a membrane* 3 *with charge "−". What exactly is the action of such a rule, in particular, how it acts upon the inner membranes of the involved membrane and the corresponding rules and objects, will be defined in the next section.*

A conclusion to this subsection is the definition of a P system with dynamic (tree-like) membrane structure.

Definition 6. *The following construct will be referred to as* P system with dynamic (tree-like) membrane structure*:*

$$\begin{aligned}\Pi \ = \ & (M, O, R, \sigma, object, rules, iLabels, oLabels, buildInput,\\ & outputBuilder, contextChecker, msRule, \lambda_O, i_0),\end{aligned}$$

where $\lambda_O \in O$ *is a "default" object to be attached to newly created membranes,* $i_0 \in H$ *is the label of the output membrane, and* $msRule : L \times R \to allMSRules \cup \{\perp\}$ *is defined to be the membrane structure rule associated with the rule* r *associated in its turn with a membrane* m.

If $r \notin rules(m)$, then $msRule(m, r) \stackrel{def}{=} \perp$. If the rule r does not influence the membrane structure, $msRule(m, r) \stackrel{def}{=} \perp$.

4.2 Computation in P Systems with Dynamic Membrane Structure

It is now possible to describe the computation in P systems with dynamic membrane structures.

Among the first things, the exact semantics of membrane structure rules should be described. Consider a P system Π with dynamic membrane structure as defined in the previous section, a membrane $m \in L$,

a rule $r \in rules_{\Pi}(m)$, and the corresponding membrane structure rule $g = msRule_{\Pi}(m, r)$ (assume that it exists, for the purposes of this explanation). Define the utility functions $lhs_{\Pi}, rhs_{\Pi} : allMSRules_{\Pi} \to (H \times A)^*$ as $lhs_{\Pi}(u \to v) = u$ and $rhs_{\Pi}(u \to v) = v$.

Before making this visible from the formal description of semantics, it will be helpful to state that a configuration of P system with dynamic membrane structure includes the mappings between membranes and objects, labels, attributes, as well as the relations between the membranes

$$C = (object, (L, \wedge), l, a).$$

Next define the function $labAttrsMultiset_{\Pi,C} : L \to (H \times A)^*$ to return the pairs of labels and attributes the number of times they occur in inner membranes of a given membrane a:

$$\begin{aligned} labAttrsMultiset_{\Pi,C}(m) &\stackrel{def}{=} buildMultiset(inner_{\Pi}(m)), \\ \text{where } buildMultiset(\varnothing) &\stackrel{def}{=} \lambda, \\ b \in adj \Longrightarrow buildMultiset(adj) &\stackrel{def}{=} (l(m), a(m)) \\ &\quad \uplus buildMultiset(adj'), \\ adj' &\stackrel{def}{=} adj \backslash \{b\}. \end{aligned}$$

Here $\uplus$ was used to denote multiset union. Note the similarity between this function and the notation $labsAttrs$, introduced in the previous section.

Proceed now with defining the function $msRuleApplicable_{\Pi,C} : L \times allMSRules \to \{\texttt{true}, \texttt{false}\}$ to decide whether a membrane structure rule g is applicable to the membrane m or not:

$$msRuleApplicable_{\Pi,C}(m, g) \stackrel{def}{=} lhs_{\Pi}(g) \subseteq labAttrsMultiset_{\Pi,C}(m),$$

where $\subseteq$ was used to denote multiset inclusion.

Now that applicability of a membrane structure rule can be decided, it is time to describe how such a rule is applied. Define the function $labelMembranesMap_{\Pi,C} : L \times H \to \hom_{\mathbf{Set}}(H, 2^L)$ to produce a mapping between some labels in H and the corresponding inner

membranes of a membrane:

$$labelMembranesMap_{\Pi,C}(m, H')(h) \stackrel{def}{=} l^{-1}(h) \cap inner(m),$$

where $l^{-1} : H \to 2^L$ provides the set of membranes labelled with a given label: $l^{-1}(h) \stackrel{def}{=} \{m \in L \mid l(m) = h\}$.

Again, consider a membrane $m \in L$, one of its rules $r \in rules_\Pi(m)$, and the corresponding membrane structure rule $g \in msRule_\Pi(m, r)$. Define the function $involvedMembranes_{\Pi,C} : L \times allMSRules \to 2^L$ to produce the set of membranes involved by the labels in left-hand side of the membrane structure rule:

$$\begin{aligned} involvedMembranes(m, g) &\stackrel{def}{=} \bigcup_{(h,attr)\in I} map(h), \text{ where} \\ map &\stackrel{def}{=} labelMembranesMap(m, labels), \\ labels &\stackrel{def}{=} \{h \in H \mid \exists attr \in A \,.\, (h, attr) \in I\}, \\ I &\stackrel{def}{=} lhs_\Pi(g). \end{aligned}$$

If g is not a membrane structure rule associated with one of the rules of the membrane m, the function is defined to take the value $\perp$:

$$\begin{aligned} (\nexists r \in rules_\Pi(m) \,.\, g = msRule(r)) \Longrightarrow \\ involvedMembranes_{\Pi,C}(m, g) \stackrel{def}{=} \perp \,. \end{aligned}$$

Suppose $msRuleApplicable_{\Pi,C}(m, g) = \texttt{true}$. In this case, define the function $applyMSRule_{\Pi,C}(m, g)$ in the following way:

$$applyMSRule_{\Pi,C}(m, g) \stackrel{def}{=} (object', (L', \wedge'), l', a') = C'.$$

If $msRuleApplicable_{\Pi,C}(m, g) = \texttt{false}$, or $\nexists r \in rules_\Pi(m) \,.\, g = msRule(r)$, $applyMSRule_{\Pi,C}(m, g) \stackrel{def}{=} \perp$.

The underlying set L' of the new semilattice $(L', wedge)$ is obtained by removing first all the membranes involved in the left-hand side of the rule g, and all their inner membranes:

$$\begin{aligned} L'_1 &\stackrel{def}{=} L \setminus (\{b \in L \mid \exists b' \in I \,.\, b' \leq b\}), \\ \wedge'_1 &\stackrel{def}{=} \wedge \setminus \{(b', b'') \mid b' \in I \text{ or } b'' \in I\}, \quad \text{where} \\ I &\stackrel{def}{=} involvedMembranes_{\Pi,C}(m, g). \end{aligned}$$

The symbol $removeMSRuleLHS$ will be used to refer to this operation, i.e.,

$$(L'_1, \wedge'_1) = removeMSRuleLHS((L, \wedge)).$$

Now, define the function $reAddMembranes$ in the following way:

$$reAddMembranes(\lambda, i, (P, \wedge)) \stackrel{def}{=} (P, \wedge),$$

and the value $i \in \mathbb{N}$ is not used in this case. If the first argument α of the function is not an empty multiset and $(h, a) \in \alpha$, then

$$\begin{array}{l} reAddMembranes(\alpha, i, (P, \wedge)) \stackrel{def}{=} \\ reAddMembranes(\alpha', i+1, (P', \wedge')), \text{where} \end{array}$$

$$\begin{array}{rcl} \alpha' & \stackrel{def}{=} & \alpha \backslash ((h, a), 1), \\ P' & \stackrel{def}{=} & P \cup S', \\ \wedge' & \stackrel{def}{=} & \wedge \cup \{(m', b) \in L \times S \mid m' \leq m\}, \\ S' & \stackrel{def}{=} & \{b'_i \in L \mid \exists b \in S \,.\, b \leq b'\}, \\ S & \stackrel{def}{=} & l^{-1}(h) \cup inner(m). \end{array}$$

Thus, according to the definition,

$$(L'_2, \wedge'_2) = reAddMembranes(rhs_\Pi(g), 0, L'_1)$$

reintroduces to the membrane structure all those membranes which have been removed during the construction of L'_1 and which are mentioned in the right-hand side of g, together with all their inner membranes. However, in the process, unique labels are attached to each of the new membranes, which makes it possible to actually duplicate a membrane together with all its inner membranes.

To keep the new labellings synchronised, along with all other identities in $reAddMembranes$, consider the following included among the

definitions in this function:

$$\begin{aligned} b_i \in S \Longrightarrow l'(b_i) &\stackrel{def}{=} h \\ \text{and } l'(b_i) &\stackrel{def}{=} a, \\ b_i \in S' \backslash S \Longrightarrow l'(b_i) &\stackrel{def}{=} l(b_i) \\ \text{and } a'(b_i) &\stackrel{def}{=} a(b_i), \\ object'(b_i) &\stackrel{def}{=} object(b). \end{aligned}$$

Thus, $reAddMembranes$ also updates the labellings for the immediately inner membranes of m which are involved with the membrane structure rule, but leaves the labellings intact for the membranes further down the tree.

Lastly, define the function $addMembranes$ in the following way:

$$\begin{aligned} addMembranes(\lambda, (P, \wedge)) &\stackrel{def}{=} (P, \wedge) \\ (h, a) \in \alpha \Longrightarrow & \\ addMembranes(\alpha, (P, \wedge)) &\stackrel{def}{=} addMembranes(\alpha', (P', \wedge')), \\ \text{where } \alpha' &\stackrel{def}{=} \alpha \backslash \{((h, l), 1)\}, \\ P' &\stackrel{def}{=} P \cup \{m_h\}, \\ \wedge' &\stackrel{def}{=} \wedge \cup \{(m', m_h) \mid m' \in L \\ & \qquad \text{and } m' \leq m\}, \\ m_h &\notin P. \end{aligned}$$

This function adds a new symbol m_h for each new label h in the right-hand side of the membrane structure rule g. Consequently,

$$(L', \wedge') = addMembranes(newMems, (L'_2, \wedge'_2))$$

is the new semilattice, representing the underlying tree of the dynamic membrane structure.

Again, to update the labellings of the membrane structure, the following definitions should be added to the definition of $addMembranes$:

$$\begin{aligned} l'(m_h) &\stackrel{def}{=} h, \\ a'(m_h) &\stackrel{def}{=} a, \\ object'(m_h) &\stackrel{def}{=} \lambda_O. \end{aligned}$$

Thus, the newly added membranes are labelled with default objects, specified in the definition of the P system.

Finally, to complete the definitions of the new labellings of the membrane structures, the following is stated:

$$m \in L_0 \Longrightarrow l'(m) \stackrel{def}{=} l(m) \text{ and } a'(m) \stackrel{def}{=} a(m) \text{ and } object'(m) \stackrel{def}{=} object(m).$$

The conclusion at this point is that $applyMSRule_{\Pi,C}$ formally describes the semantics of a membrane structure rule by constructing a new configuration C' in the context of a P system with dynamic membrane structure Π and a reference configuration C of it.

No types have been provided for $applyMSRule_{\Pi,C}$ and utilities used to construct it, because it returns functions whose domains belong to proper classes (for example, the function $object'$ whose domain is a lattice) and the notations introduced in this paper are insufficient to express this fact. This is irrelevant to the present formalisation, though.

It is now possible to move to defining an evolution step of a P system with dynamic membrane structure. As in Subsection 3.3, only the marking algorithm and one step of evolution will be described in detail. This will create sufficient foundation for continuing the reasoning along the lines shown in [8] and presents little interest in the context of this paper.

Before describing the marking algorithm itself, note that the set of rules employed in P system with dynamic membrane structure is partitioned into two sets: the rules that do not have membrane structure rules associated, and the rules that have:

$$\begin{aligned} R_{\neg\mu} &\stackrel{def}{=} \{r \in R \mid \exists m \in L \,.\, r \in rules_\Pi(m) \text{ and } msRule_\Pi(r) = \perp\}, \\ R_{\mu} &\stackrel{def}{=} \{r \in R \mid \exists m \in L \,.\, r \in rules_\Pi(m) \text{ and } msRule_\Pi(r) \neq \perp\}. \end{aligned}$$

These two types of rules will always be treated in certain order: the rules in $R_{\neg\mu}$ will always be analysed first.

Consider a multiset ρ of pairs rules and the corresponding membranes:

$$\rho = \{((m, r), n) \mid m \in L \text{ and } r \in rules_\Pi(m) \text{ and } n \in \mathbb{N}\}.$$

According to the classification of rules above, split ρ into two multisets:

$$\begin{aligned} \rho_{\neg\mu} &\stackrel{def}{=} \{((m,r),n) \in \rho \mid r \in R_{\neg\mu}\}, \\ \rho_{\mu} &\stackrel{def}{=} \{((m,r),n) \in \rho \mid r \in R_{\mu}\}. \end{aligned}$$

While it is tempting to declare that the function $isApplicableMultiset_{\Pi,C}$ can be used to decide the applicability of $\rho_{\neg\mu}$, it is not exactly so since, in P systems with dynamic membrane structures, the attributes of the membrane a rule is associated with must also be checked. Therefore, define the following simple function $ruleApplicable_{\Pi,C} : L \times R \to \{\texttt{true}, false\}$:

$$\begin{aligned} ruleApplicable_{\Pi,C}(m,r) \stackrel{def}{=} & \\ & applyRule_{\Pi,C}(m,r) \neq \perp \\ \text{and} \quad & contextChecker(m,r)(a(m)) = \texttt{true}. \end{aligned}$$

As usual, for $r \notin rules(m)$, $ruleApplicable_{\Pi,C}(m,r) = \perp$. Now, if one redefines $isApplicableMultiset_{\Pi}$ to use $ruleApplicable_{\Pi,C}$ instead of checking the condition $r \in applicableRules(\Pi, C)$, one may use $isApplicableMultiset_{\Pi}$ to check the applicability of $\rho_{\neg\mu}$ in a P system with dynamic membrane structure.

The current question is how to decide the applicability of ρ_{μ}. The answer to this question is constructed pretty much along the same line as is $isApplicableMultiset_{\Pi,C}$. Define the function $ruleApplicableG_{\Pi,C} : L \times R \to \{\texttt{true}, \texttt{false}\}$ to return `true` if, for a membrane $m \in L$ and its rule $r \in rules_{\Pi}(m)$, both r and $msRule(r)$ are applicable (here G stands for "generalised"):

$$\begin{aligned} ruleApplicableG_{\Pi,C}(m,r) \stackrel{def}{=} & \; msRuleApplicable(m, msRule(r)) \\ \text{and} \quad & ruleApplicable_{\Pi,C}(m,r). \end{aligned}$$

As usually defined in the situations when r is not a rule associated with the membrane m:

$$r \notin rules(m) \Longrightarrow ruleApplicableG_{\Pi,C}(m,r) \stackrel{def}{=} \perp .$$

Further, if $msRule(r) = \perp$, for consistency,

$$ruleApplicableG_{\Pi,C} \stackrel{def}{=} ruleApplicable_{\Pi,C}(m, r).$$

Now define the function $eraseMSRulesG$ which, quite in parallel to $eraseMSRules$ and $applyMSRule$, produces a configuration without the premises which made the rule r and the corresponding $msRule(r)$ applicable:

$$\begin{array}{rcl} eraseMSRulesG_{\Pi,C}(m, r) & \stackrel{def}{=} & (objects', (L', \wedge'), l, a), \text{ where} \\ objects' & \stackrel{def}{=} & eraseMSRules_{\Pi,C}(m, r), \end{array}$$

and $(L', \wedge')$ is defined as follows:

$$(L', \wedge') \stackrel{def}{=} removeMSRulesLHS_{\Pi,C}(m, msRule(r)).$$

Again, $r \notin rules(m) \Longrightarrow eraseMSRulesG_{\Pi,C}(m, r) \stackrel{def}{=} \perp$.

Now, define $isApplicableMSMultiset_{\Pi}$ to decide whether ρ_μ is applicable in the supplied configuration. For an empty multiset, the definition is trivial:

$$isApplicableMSMultiset_{\Pi}(C, \lambda) \stackrel{def}{=} \texttt{true}.$$

For a nonempty multiset ρ_μ and $(m, r) \in \rho_\mu$:

$$\begin{array}{l} isApplicableMSMultiset_{\Pi}(C, \rho_\mu) \stackrel{def}{=} \\ ruleApplicableG_{\Pi,C}(m, r) \text{ and } isApplicableMSMultiset_{\Pi}(C', \rho'_\mu), \\ \text{where } C' \stackrel{def}{=} eraseMSRulesG_{\Pi,C}(m, r), \rho'_\mu \stackrel{def}{=} \rho_\mu \backslash \{(m, r)\}. \end{array}$$

Finally, the function $isApplicableMultisetG_{\Pi,C}$ decides whether the multiset of rules ρ is applicable in the given configuration:

$$\begin{array}{l} isApplicableMultisetG_{\Pi,C}(\rho) \stackrel{def}{=} \\ isApplicableMultiset_{\Pi}(C, \rho_{\neg\mu}) \\ \quad \text{and } isApplicableMSMultiset_{\Pi}(C', \rho_\mu), \end{array}$$

where $C' = (object', (L, \wedge), l, a)$ and $object'$ is the labelling of the membrane structure with objects at which $isApplicableMultiset_{\Pi}(C, \lambda)$ has arrived.

Now, the application of an applicable multiset of rules ρ to configuration of P system with dynamic membrane structure is performed in two stages. First, the multiset of rules $\rho_{\neg\mu}$ is applied as described in Subsection 3.3. Then, the rules in ρ_{μ} are applied one by one, using the function $applyRuleG_{\Pi,C}(m, r)$, defined in the following way. For $r \in rules(m)$ and $g = msRule(r) \neq \perp$,

$$\begin{aligned} applyRuleG_{\Pi,C}(m,r) &\stackrel{def}{=} applyRule_{\Pi,C'}(m,r), \text{ where} \\ C' &\stackrel{def}{=} applyMSRule_{\Pi,C}(m,g). \end{aligned}$$

When $msRule(r) = \perp$, $applyRuleG_{\Pi,C} = applyRule_{\Pi,C}$. For $r \notin rules(m)$, $applyRuleG_{\Pi,C} \stackrel{def}{=} \perp$.

4.3 P Systems With Active Membranes

This section will show how the five types of rules in P systems with active membranes are translated into the suggested formalism.

The rules of type (a), $[a \to v]^e_h$, will be translated to rules in $R_{\neg\mu}$, whose context checkers will assure check the charge of the containing membrane.

The rules of type (b), $a[\,]^{e_1}_h \to [b]^{e_2}_h$, will be modelled in the following way. All parent membranes of h will have a rule which will take an instance of a and will place it into the membrane h: $a \to (a, h)$. The corresponding membrane structure rule will be $(h, e_1) \to (h, e_2)$.

Similarly, for the rules of type (c), $[a]^{e_1}_h \to [\,]^{e_2}_h b$, the parent membrane of h will contain a rule $(a, h) \to b$, with the corresponding membrane structure rule being again $(h, e_1) \to (h, e_2)$.

For the dissolution rules of type (d), $[a]^e_h \to b$, the system will include a rule $a \to b$, for which *buildInput* will fetch the whole multiset contained in the inner membrane h, so that the contents of this membrane get merged with the contents of the parent membrane. The associated membrane structure rule will be $(h, e) \to \lambda$.

Finally, for the division rules of type (e), $[a]_h^{e_1} \rightarrow [b]_h^{e_2}[c]_h^{e_3}$, the parent membrane of h will contain a rule $(a, h) \rightarrow (b, h)(c, h)$ with the corresponding membrane structure rule $(h, e_1) \rightarrow (h, e_2)(h, e_3)$. The function provided by *outputBuilder* for such a rule will take care of distributing the symbols b and c across the compartments in the correct order.

Note that, in this setup, the rules which do not have membrane structure rules associated, are applied first, just required by the definition of a P system with active membranes (Chapter 11 of [4]).

5 Conclusion

Instead of focusing on certain kinds of P systems, constructed by combining membrane structures with a certain type of rules, this paper has brought attention to the membrane structures themselves, as separate objects of study. This approach was motivated by the observation that it has become quite popular with researchers in the domain of computational devices to combine a known type of rules with membrane structures. A generalisation of membrane structures was provided in terms of algebraic structures and mappings and a number of known concrete P systems models were shown to be covered by the introduced formalisation.

Importantly enough, the constructs suggested in this paper do not focus on the nature of the rules on which the membrane structure acts. In fact, only some basic statements are made about the rules and the objects placed in the compartments of the membrane structure. This makes it possible to fit the majority of known P system models in the suggested formalisation.

Even more importantly, it turns out that membrane structures can indeed be quite easily separated from the rules associated with the membranes. Static membrane structures turned out to be simpler to factor out than dynamic membrane structures, a lot less additional constructions are required in the former case. However, as visible in Subsection 4.3, actually fitting a P system model with active membranes in the suggested formalisation is fairly straightforward. In fact,

the majority of rules shown in [9] can be fit into the constructions shown in this paper.

A remarkable feature of the formalised models suggested in the present work is that they are rather considerably narrowed down to cover as little as possible extra capabilities. As different from the powerful, generalised interaction rules shown in [8], the constructions in this paper only allow for tree-like membrane structures, with communication limited to the parent membranes and the immediately inner membranes.

While the formalisations exposed in this paper may not themselves come to know wide usage, the point of view will hopefully make more researchers consider membrane structures without the context of concrete P system models. There are two major reasons motivating such a shift of perspective. The first reason is that membrane structures are not just trees, as it has been shown in this paper, and have the full right to be studied on their own. The second reason is that such a view on membrane structures opens further possibilities for placing different types of rules in compartments and thus obtaining a potential plethora of results.

References

[1] Gh. Păun. *Computing with membranes.* TUCS Report 208, Turku Center for Computer Science, 1998.

[2] Gh. Păun. *Membrane Computing. An Introduction.* Springer-Verlag, 2002.

[3] M. J. Pérez-Jiménez, F. Sancho-Caparrini. *A formalization of transition P Systems.* Fundamenta Informaticae – Membrane computing, Volume 49 Issue 1, January 2002.

[4] Gh. Păun, G. Rozenberg, A. Salomaa, Eds. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.

[5] A. Krassovitskiy, Yu. Rogozhin, S. Verlan. *Computational power of insertion-deletion (P) systems with rules of size two*. Journal Natural Computing, Volume 10 Issue 2, June 2011.

[6] B. A. Davey, H. A. Priestley. *Introduction to Lattices and Order (second ed.)*. Cambridge University Press, 2002.

[7] Eric W. Weisstein. *Tree*. From MathWorld – A Wolfram Web Resource, http://mathworld.wolfram.com/Tree.html.

[8] R. Freund and S. Verlan. *A Formal Framework for Static (Tissue) P Systems*. G. Eleftherakis, P. Kefalas, G. Paun, G. Rozenberg, A., Salomaa, eds., 8th International Workshop on Membrane Computing, WMC2007. vol 4860 of LNCS, 2007.

[9] E. Csuhaj-Varjú, A. Di Nola, Gh. Păun, M. J. Pérez-Jiménez, G. Vaszil. *Editing Configurations of P Systems*. Fundamenta Informaticae, Volume 82 Issue 1-2, July 2008.

[10] The P systems web page. `http://ppage.psystems.eu/`

Sergiu Ivanov,

Received July 6, 2012

Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E–mail: *sivanov@math.md*

University of Academy of Sciences of Moldova
Faculty of Real Sciences
Academiei 3/2, MD-2028 Chişinău, Republic of Moldova