A formalization of membrane systems with dynamically evolving structures

Rudolf Freund[a], Ignacio Pérez-Hurtado[b], Agustín Riscos-Núñez[b] and Sergey Verlan[c]

[a]Faculty of Informatics, Vienna University of Technology, Favoritenstr. 9, 1040 Vienna, Austria; [b]Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012, Sevilla, Spain; [c]Département Informatique, Université Paris Est, 61, av. Général de Gaulle, 94010 Créteil, France

This article introduces a general formalism/framework flexible enough to cover descriptions of different variants of P systems having a dynamic membrane structure. Our framework can be useful for the precise definition of new variants of P systems with a dynamic structure and for the comparison of existing definitions as well as for their extension. We give a detailed definition of the formalism and we present some examples of how to translate several existing variants of P systems with a dynamic structure.

Keywords: P systems; dynamic structure; formal definition

2000 AMS Subject Classification: 68Q05

1. Introduction

The goal of this article is to define a formal framework that captures the essential properties of P systems with a *dynamic* structure (i.e. their structure can change during the computation, e.g. by dissolving, moving or generating membranes or cells). This framework can be seen as a kind of meta-language that permits to describe a P system and its evolution. Our main goal is to provide a simple tool for the analysis of different models of P systems with a dynamic structure. There are numerous possible applications of the results of such an analysis, for example, the comparison and the extension of existing models and the creation of new models of P systems with a dynamic structure.

There are other approaches having a general point of view rather than focusing on a specific model of P systems, mainly related to research lines involving the development of simulation software. For example, *kernel P systems* [8] can be seen as an intersection (or merging) of several models, bringing together enough ingredients to ensure a high computing power, but trying to keep

the model as simple as possible in order to allow for automatic reasoning and model-checking tools to be applied on this setup. Another significant example is *P-Lingua* [7,14], a software framework that includes a definition language for P systems (also called P-Lingua) and a library (pLinguaCore) that is able to parse P-Lingua files and simulate P system computations. It is important to remark that P-Lingua covers many models of P systems, but distinguishes between them by including model-specific instructions for the parser to handle syntactical ingredients such as charges, labels and structure of membranes. Moreover, the behaviour of the P systems (semantics) is reproduced by implemented simulation algorithms in pLinguaCore. Thus, pLinguaCore includes one or more simulation algorithms for each supported P system model in the language. This software framework has been successfully used to simulate, for example, P systems with active membranes and division rules [7], tissue-like P systems [9] and models of real ecosystems by means of probabilistic P systems [2].

The article is organized as follows. Section 2 recalls some standard notions that will be used throughout the article. Section 3 gives the definition of the framework and presents the related algorithms. Section 5 presents a taxonomy that permits to define shortcuts for the commonly used cases. Section 6 gives examples of the translation of several well-known types of P systems with a dynamic structure. Finally, the perspectives of the presented approach are discussed.

This article is an extended version of [4].

2. Preliminaries

We assume that the reader is familiar with standard definitions in formal language theory (e.g. we refer to [12] for all details).

2.1 *P systems*

Membrane computing considers classes of computing models abstracting from the structure and functioning of a living cell or a group of cells. These computational devices consist of a compartmentalized structure, in which multisets of objects can be placed, as well as a set of rules that determine the evolution of the system along a computation, acting over objects and/or membranes. Generically, we call the obtained devices P systems (in the case of a tree structure) and tissue P systems (in the case of a graph structure). There are many defined classes of (tissue) P systems and most of them are equivalent to Turing machines, hence computationally complete/universal [11].

There exist a large number of classes (or models) of P systems in the literature, containing different syntactical/semantical ingredients (we refer to the handbook of Păun *et al.* [11] for an overview); for example, symport/antiport P systems restrict themselves to using only communication rules. Another classical model is given by P systems with active membranes, where the membranes have electrical charges associated with them, and such charges play an important role in the application of rules. This article tries to cover most of the existing classes of (tissue) P systems, in particular, those that allow changes on their membrane structure during the computation (e.g. by creation, division or dissolution of compartments).

Informally, a definition of a P system consists of

- a description of the initial structure (indicating the graph relation between the compartments and any additional information such as labels and charges),
- a list of the initial multisets of objects placed in each compartment at the beginning of the computation and
- a set of rules, acting over objects and/or over the structure.

A computation of a P system Π can be defined as a sequence of transitions between configurations, eventually ending in some halting configuration. To give a more precise description of the semantics, we define the following four notions (functions):

- Applicable$(\Pi, \mathcal{C}, \delta)$ – the set of multisets of rules of Π applicable in the configuration $\mathcal{C}$, according to the derivation mode δ.
- Apply$(\Pi, \mathcal{C}, R)$ – the configuration obtained by the (parallel) application of the multiset of rules R to the configuration $\mathcal{C}$.
- Halt$(\Pi, \mathcal{C}, \delta)$ – a predicate that yields true if $\mathcal{C}$ is a halting configuration of the system Π evolving in the derivation mode δ. Usually, the halting condition is that no rule can be applied anymore, but one could define more sophisticated conditions.
- Result$(\Pi, \mathcal{C})$ – a function giving the result of the computation of the P system Π, when the halting configuration $\mathcal{C}$ has been reached. Generally, this is an integer function; however, it is possible to generalize it, allowing, for example, Boolean or vector functions.

The transition of a P system Π according to the derivation mode δ (e.g. the maximally parallel mode) is defined as follows: we pass from a configuration $\mathcal{C}$ to $\mathcal{C}'$ (written as $\mathcal{C} \Rightarrow_\delta \mathcal{C}'$) if and only if

$$\mathcal{C}' = \text{Apply}(\Pi, \mathcal{C}, R), \quad \text{for some } R \in \text{Applicable}(\Pi, \mathcal{C}, \delta).$$

In general, the result of the computation of a P system is interpreted as the union of the results of all possible computations (in the same way as the language generated by a grammar is defined, in formal language theory, as gathering the results of all possible derivations).

The precise definition of the above four functions depends on the selected model of P systems. The goal of this article is to provide a concrete variant of P systems (hence with concrete definitions of these functions), called a *formal framework*, such that most of the existing models of P systems could be obtained by a restriction (eventually using a simple encoding) of this formal framework with respect to different parameters.

For a comprehensive presentation of the standard notions of P systems, interested readers can refer to the books of Păun [10] and Păun *et al.* [11] (see also references listed on the web page [13]).

2.2 *Graph transformations*

There exist several ways to define a graph transformation. We will define a graph transducer using the formalism from [3]. This formalism defines the graph transformation as a graph-controlled graph rewriting grammar with appearance checking using the following operations:

- $I(x)$: creation of a new node labelled by x;
- $D(x)$: deletion of a node labelled by x (only possible if the node is isolated, which means no edges are ending in or leaving from this node);
- $C(x, y)$: change the label of the node labelled by x to y;
- $I(l_1, \lambda, l_2; l_1', a, l_2')$: insert an edge labelled by a between two nodes labelled by l_1 and l_2; after the insertion, nodes are relabelled to l_1' and l_2', respectively;
- $D(l_1, a, l_2; l_1', \lambda, l_2')$: delete the edge labelled by a between two nodes labelled by l_1 and l_2; after the deletion, nodes are relabelled to l_1' and l_2', respectively;
- $C(l_1, a, l_2; l_1', a', l_2')$: rename to a' the label of the edge labelled by a between two nodes labelled by l_1 and l_2. After this operation, nodes are relabelled to l_1' and l_2', respectively.

It was proved in [3] that the above formalism is computationally complete.

In what follows, we use some particular graph transducers whose definitions are as follows:

- DELETE(x): $C(x,x')$, $D(x',a,y;x',\lambda,y)$ and $D(y,a,x';y,\lambda,x')$ (looping over a and y), $D(x')$
  Node x is deleted, together with all existing edges from x to any node y and from any node y to x.
- INSERT(x): $I(x)$
- INSERT-EDGE(x,y): $I(x,\lambda,y;x,a,y)$
- DELETE-EDGE(x,y): $D(x,a,y;x,\lambda,y)$

3. Definition of the framework

This article extends the approach used in [5] for P systems with a static structure. We recall that the framework for static P systems is mainly composed of five ingredients: the definition of the configuration of the system, the definition of rules, the definition of the applicability and of the application of a rule/multiset of rules, the derivation (transition) mode and the halting condition. A configuration is a list of multisets corresponding to the contents of membranes of a P system and rules generalize most kinds of rules used in the P system area. Based on this general form of rules, the applicability and application of a (group of) rule(s) are defined using an algorithm. This permits to compute the set of all applicable multisets of rules for a concrete configuration $\mathcal{C}$ (Applicable($\Pi,\mathcal{C}$)). Then, this set is restricted according to the derivation mode δ (Applicable($\Pi,\mathcal{C},\delta$)). For the transition, one of the multisets from this last set is non-deterministically chosen and applied, yielding a new configuration. The result of the computation is collected when the system halts according to the halting condition, which corresponds to a predicate that depends on the configuration and the set of rules.

In the case of P systems with a dynamic structure, the first three ingredients are to be changed in order to accommodate the fact that the structure of the system can change. Informally, a *configuration* contains a list of triples (i,h,w), where i is the unique identifier of a cell/membrane, h is its label and w is its contents. A configuration also contains the description of the structure of the system, which is given by a binary relation ρ on cell identifiers.

We assume that the set of rules is fixed (does not change in time). Rule actions are expressed in terms of 'virtual' cells (membranes). These virtual cells are identified by labels. The process of the application of rules first makes a correspondence between the current configuration and the virtual cells described in a rule; that is, it tries to 'match' the constraints of virtual cells (labels, relation, contents, etc.) against the current configuration. When a subset of cells from the current configuration (say $\mathcal{I}$) matches the constraints of a rule, we say that a rule can be *instantiated* by the instance $\mathcal{I}$. The instantiation of r by $\mathcal{I}$ is the couple $(r,\mathcal{I})$, denoted by $r\langle\mathcal{I}\rangle$, and it can then be treated as a rule that could be applied as in the static case. The rules also contain additional ingredients that permit to modify the structure (the relation ρ).

Instances of rules can further be used to compute the applicable set of multisets of rules and we provide an algorithm for this purpose. The derivation modes and halting conditions can easily be applied to this set exactly as in the static case.

3.1 *Formal definition of a configuration*

We start by defining a configuration of a P system. Since we deal with P systems with a dynamic structure, it should be taken into account that the number of cells (membranes) is not fixed (it may even be unbounded), and hence a list of variable length will be used.

DEFINITION 3.1 *A basic configuration C (of size n) over an alphabet of objects O is a finite list $(i_1, w_1) \cdots (i_n, w_n)$, with $i_j \in \mathbb{N}$ and $w_j \in O^*$, for $1 \leq j \leq n$, such that all the numbers appearing in the first components of the pairs are different from each other.*

- *We denote the size of C by* $\text{size}(C)$.
- *The set of all possible basic configurations of any size $n > 0$ is denoted by $\mathbb{C}$.*

DEFINITION 3.2 *Each element (i_j, w_j), $1 \leq j \leq n$, of a basic configuration is called a cell*:

- *the first component (i_j) is called the id of the cell and*
- *the second component (w_j) corresponds to the contents of the cell.*

Remark 1 (*Finiteness*) If not stated otherwise, we suppose that all multisets of a basic configuration are finite. If needed, for example, for the representation of the environment, the definitions that follow can be adapted to infinite multisets by adding corresponding constraints to the rule definition, in a similar way as done in [5].

Remark 2 (*Identification*) In what follows, the id's that will be used are natural numbers corresponding to the position of each cell in the list. However, if needed, one could define any special function $id : C \to \mathbb{N}$, provided that this function is injective (i.e. different cells from the basic configuration C must get different id's).

DEFINITION 3.3 *Given a P system having an alphabet of objects O and a set of membrane labels Lab, a configuration $\mathcal{C}$ is a couple (L, ρ), where L is a list of 'labelled cells' $(i_1, l_1, w_1) \cdots (i_n, l_n, w_n)$, with the id $i_j \in \mathbb{N}$, the label $l_j \in Lab$ and the contents $w_j \in O^*$, for $1 \leq j \leq n$, such that all the cell id's are different from each other. The second component $\rho \subseteq \mathbb{N} \times \mathbb{N}$ is a relation that represents the connections between cells (it can be seen as a graph where the nodes are the cell id's).*

The set of all possible configurations is denoted by $\mathfrak{C}$.

Note that in a configuration each cell has an id that is unique and a label that is not necessarily unique. Note also that multisets of objects are represented (as usual in membrane computing) as strings.

The interpretation of relation ρ may differ depending on the selected P system model, but its goal is to capture how cells (or membranes) are organized in the 'membrane structure'. In cell-like P systems, this corresponds to the parent relation, while in tissue P systems this corresponds to the communication graph of the system.

3.2 *Formal definition of a rule*

Now, we give the definition of a rule. As mentioned in the introduction, this definition is an abstract formalization that tries to cover most of the existing types of rules used in the membrane computing literature.

For the sake of simplicity, along the description of the rule, we avoid using variables for the id's of the cells to which the rule is going to be applied. We use instead a set of k *virtual cells* (where k is the length of $\text{Labels}(r)$) having $\mathbb{N}_k = \{1, \ldots, k\}$ as their id's. Such id's will be conveniently instantiated when applying the rule to a particular configuration, as will be explained later. We also use $\mathbb{C}_k$ to denote the set of basic configurations where only $\{1, \ldots, k\}$ are allowed as cell id's.

DEFINITION 3.4 *A rule r is a formal construct having the following 11 components*:

A. *Checking*
(1) $\text{Labels}(r) = (l_1, \ldots, l_k) \in \text{Lab}^k$ *is a list of cell labels that introduces k virtual cells.*
(2) $\rho(r) \subseteq \mathbb{N}_k \times \mathbb{N}_k$ *is a relation on the virtual cells* (*e.g. indicating parent relation*).
(3) $\text{Perm}(r) \subseteq \mathbb{C}_k$ *defines the permitting conditions.*
(4) $For(r) \subseteq \mathbb{C}_k$ *defines the forbidding conditions.*

B. *Modification of existing configuration/structure*
(5) $\text{Rewrite}(r) = U \to V$, *with* $U, V \in \mathbb{C}_k$, *is a general rewriting rule over basic configurations.*
(6) $\text{Label-Rename}(r) \in (\mathbb{N}_k \times \text{Lab})^*$ *allows us to specify new labels for the cells whose index* (*or id*) *appears in the list* (*same index cannot appear more than once in the list*).
(7) $\text{Delete}(r) \in \mathbb{N}_k^*$ *gives the indexes of virtual cells to be deleted.*
(8) $\text{Delete-and-Move}(r) \in (\mathbb{N}_k \times \mathbb{N}_k)^*$ *is a list of couples of indexes. Each pair* (i, j) *appearing in this component indicates that the virtual cell i should be deleted/dissolved and its contents should be moved to the virtual cell j.*

C. *Creation of new structures*
(9) $\text{Generate}(r) \in (\mathbb{N}' \times \text{Lab} \times O^*)^*$ *is a list of triples consisting of a* (*primed*) *index, a label and a multiset* (*e.g.* (j', h, u)). *This component introduces new cells to be created by the application of the rule.*
(10) $\text{Generate-and-Copy}(r) \in (\mathbb{N}' \times \text{Lab} \times \mathbb{N} \times (O^* \times O^*))^*$ *is a list of quadruplets consisting of a* (*primed*) *index, a label, an index and a rewriting rule over multisets of objects, where each tuple* $(j', h, i, u \to v)$ *means that a new cell should be created, having a new id* j' *and the label h, by first copying every object inside cell i and then applying the rewriting rule to the contents of the new cell.*

D. *Structure transformation*
(11) $\text{Change-Relation}(r)$ *is a graph transducer that updates the relation* ρ. *This transducer should be recursive and it can only add and remove edges* (*no node creation/removal is allowed*).

Before moving to the concept of applicability, let us define some auxiliary notions related to the way rules can be instantiated (replacing virtual id's by existing ones).

In what follows, given a vector $\boldsymbol{v} \in \mathbb{N}^k$, its *length* is denoted by $|\boldsymbol{v}|$, and $\boldsymbol{v}|_k$ represents the kth component of $\boldsymbol{v}$.

Given a vector of indexes $\boldsymbol{i} = (i_1, \ldots, i_n) \in \mathbb{N}^n$ and a basic configuration $C \in \mathbb{C}$, $C = (j_1, w_1) \cdots (j_k, w_k)$, such that $1 \leq j_1, \ldots j_k \leq n$, we define the *instantiation* of C by $\boldsymbol{i}$, denoted $C\langle \boldsymbol{i} \rangle$, as follows:

$$C\langle \boldsymbol{i} \rangle = (\boldsymbol{i}|_{j_1}, w_1), \ldots, (\boldsymbol{i}|_{j_k}, w_k).$$

In the above formula, we allow that $\{j_1, \ldots j_k\} \neq \{1, \ldots, k\}$. In particular, some of the id's may be greater than $\text{size}(C)$ (but always within the range $[1 \cdots n]$). We also remark that $\text{size}(C) \leq |\boldsymbol{i}|$.

It is clear that if C is defined in terms of virtual id's, then $C\langle \boldsymbol{i} \rangle$ permits to replace these virtual id's by the corresponding values from $\boldsymbol{i}$ (a virtual id k is replaced by $\boldsymbol{i}|_k$, which is i_k).

For a rule r as defined above and for a vector $\boldsymbol{i}$ such that $|\text{Labels}(r)| \leq |\boldsymbol{i}|$, we obtain the instantiation of r by $\boldsymbol{i}$, denoted by $r\langle \boldsymbol{i} \rangle$, by replacing all virtual id's k by $\boldsymbol{i}|_k$ in all components involving virtual id's (i.e. in all of them except in $\text{Labels}(r)$ and in $\text{Generate}(r)$).

3.3 *Applicability of a multiset of rules*

Now, we define the applicability of a multiset of rules. First, we define the set of vectors eligible for instantiation for a rule r in a configuration $\mathcal{C}$.

In what follows, $\text{lab}(x)$ denotes the label of the cell whose id is x.

We denote by $\mathcal{C}_m$ and $\mathcal{C}_\rho$ the first and the second components of the configuration $\mathcal{C}$, respectively. We also denote by $\bar{\mathcal{C}}_m \in \mathbb{C}$ the projection of $\mathcal{C}_m$ erasing the labels (yielding a basic configuration).

DEFINITION 3.5 *We say that a vector of indexes $\boldsymbol{i} = (i_1, \ldots, i_k) \in \mathbb{N}^k$ is eligible for instantiation for a rule r in a configuration $\mathcal{C} = (L, \rho)$ if it verifies the following three properties*:

- *all indexes i_j appearing in $\boldsymbol{i}$ are different from each other*;
- *$\boldsymbol{i}$ is consistent with* $\text{Labels}(r) = (l_1, \ldots, l_k)$, *that is*, $1 \leq i_j \leq \text{size}(\mathcal{C})$ *and* $\text{lab}(i_j) = l_j$, *for every* $1 \leq j \leq k$; *and*
- *$\boldsymbol{i}$ is consistent with the relation ρ, that is*, $(j, m) \in \rho(r) \Rightarrow (i_j, i_m) \in \mathcal{C}_\rho$.

We denote by $\mathcal{I}_\mathcal{C}(r)$ the set of all vectors eligible for instantiation for a rule r in a configuration $\mathcal{C}$.

For a multiset of rules R and a configuration $\mathcal{C}$, we define the set of multisets of instantiated rules from R that are *applicable* to $\mathcal{C}$ (denoted by $\text{Applicable}(R, \mathcal{C})$), as follows.

Let $R = \{r_1, \ldots, r_n\}$ (the rules are not necessarily different) and let $\mathcal{I}_\mathcal{C}(r_i) = \{\boldsymbol{v}_{i,1}, \ldots, \boldsymbol{v}_{ik_i}\}$, $1 \leq i \leq n$. Consider an arbitrary selection of one instantiation vector, $\boldsymbol{v}_{i,j_i}$, for each r_i, $1 \leq i \leq n$. The multiset $\{(r_1, \boldsymbol{v}_{1,j_1}), \ldots, (r_n, \boldsymbol{v}_{n,j_n})\}$ belongs to $\text{Applicable}(R, \mathcal{C})$ if

- $\forall P \in \text{Perm}(r_i) \cup \text{DPerm}(r_i)$, $P\langle \boldsymbol{v}_{i,j_i} \rangle \subseteq \bar{\mathcal{C}}_m$, for every $1 \leq i \leq n$, where $\text{DPerm}(r) = \{(i, u) : \exists (j', h, i, u \to v) \in \text{Generate-and-Copy}(r)\}$;
- $\forall Q \in \text{For}(r_i)$, $Q\langle \boldsymbol{v}_{i,j_i} \rangle \not\subseteq \bar{\mathcal{C}}_m$, for every $1 \leq i \leq n$;
- $\bigcup_{i=1}^{n} \text{Bound}(r_i\langle \boldsymbol{v}_{i,j_i} \rangle) \subseteq \bar{\mathcal{C}}_m$, where $\text{Bound}(r)$ denotes the left-hand side of the rewriting rule in the component $\text{Rewrite}(r)$;
- $\forall i, k, s$ if $((s, l_1) \in \text{Label-Rename}(r_i\langle \boldsymbol{v}_{i,j_i} \rangle) \wedge (s, l_2) \in \text{Label-Rename}(r_k\langle \boldsymbol{v}_{k,j_k} \rangle))$, then $l_1 = l_2$; and
- the consecutive application of graph transducers $\text{Change-Relation}(r_i)$ and $\text{Change-Relation}(r_j)$ yields the same result regardless of the order of the application, $1 \leq i, j \leq n$.

For a P system Π having a set of rules $\mathcal{R}$, we define

$$\text{Applicable}(\Pi, \mathcal{C}) = \bigcup_{\text{Applicable}(R,\mathcal{C}) \neq \emptyset} \text{Applicable}(R, \mathcal{C}),$$

where R are multisets (of rules) over $\mathcal{R}$. Note that this is a finite union, since the size of the eligible multisets for which $\text{Applicable}(R, \mathcal{C})$ is not empty is bounded.

Following [5], it is possible to define now the derivation modes as a restriction of this set. However, it should be noted that in the case of the present framework, the definition of the derivation modes can be extended by adding new types of restrictions based on instantiation vectors. Details are discussed in Section 4.

3.4 *Application of a multiset of rules*

Now, we are ready to define how the application of a multiset of rules is performed.

Before giving the algorithm, we remark that a rule is composed of three parts: the rewriting of objects and the label change (R), the membrane deletion (D) and the membrane creation (G). The order of the application of these parts is extremely important, for example, doing the rewriting before the membrane creation permits to copy the result of the rewriting to the new membranes. In this article, we consider that the application order is RGD, that is, rewriting, creation and then deletion. This order corresponds to the actual state of art in the area of P systems with active membranes. Other orders are also possible, and this can be an interesting topic for further research.

Let $\mathcal{C} = (\{(i_1, l_1, w_1) \dots (i_n, l_n, w_n)\}, \rho)$ be the current configuration and let $RI = \{(r_1, \boldsymbol{v}_1), \dots, (r_n, \boldsymbol{v}_n)\} \in \text{Applicable}(R, \mathcal{C})$ be a multiset of pairs (rule,instantiation vector). We now define $\text{Apply}(\text{RI}, \mathcal{C}) \in \mathfrak{C}$, which is the configuration obtained as the result of applying the rules from RI to C.

The algorithm for the computation of $\text{Apply}(\text{RI}, \mathcal{C})$ is the sequence consisting of the following steps:

(1) (rewriting application): $L_1 = \{(i_1, l_1, w'_1) \dots (i_n, l_n, w'_n)\}$, where

$$w'_j = w_j - \bigcup_{(r_k, \boldsymbol{v}_k) \in RI} U_k|_j + \bigcup_{(r_k, \boldsymbol{v}_k) \in RI} V_k|_j,$$

for each $j = 1 \cdots n$, where $(U_k \to V_k) = \text{Rewrite}(r_k\langle \boldsymbol{v}_k \rangle)$ and $U_k|_j$ (resp., $V_k|_j$) denotes the multiset associated with the cell from U_k (resp., from V_k) whose id is j or the empty multiset if none of the cells has j as id.

(2) (label change): $L_2 = \{(i_1, l'_1, w'_1) \cdots (i_n, l'_n, w'_n)\}$, where

$$l'_j = \begin{cases} e_s, & \text{if } \exists (r_k, \boldsymbol{v}_k) \in RI \text{ such that } (j, e_s) \in \text{Label-Rename}(r_k\langle \boldsymbol{v}_k \rangle) \\ l_j, & \text{otherwise.} \end{cases}$$

(3) (membrane creation): ($m_1 \cdots m_{t+s}$ are new id's). We define the lists of newly created cells L_c and L'_c:

$$L_c(r_k) = (m_1, h_1, u_1) \cdots (m_t, h_t, u_t), \quad (r_k, \boldsymbol{v}_k) \in RI \text{ and}$$
$$\text{Generate}(r_k) = \{(1', h_1, u_1) \cdots (t', h_t, u_t)\}.$$
$$L_c = L_c(r_1) \cdot \dots \cdot L_c(r_n).$$

$L'_c(r_k) = (m_{t+1}, e_1, w'_{n_1} - u_1 + v_1) \cdots (m_{t+s}, e_s, w'_{n_s} - u_s + v_s)$, where $(r_k, \boldsymbol{v}_k) \in RI$ and

$$\text{Generate-and-Copy}(r) = \{(1', e_1, n_1, u_1 \to v_1) \dots (s', e_s, n_s, u_s \to v_s)\},$$
$$(i_j, l'_j, w'_j) \in L_2, 1 \le j \le n.$$
$$L'_c = L'_c(r_1) \cdot \dots \cdot L'_c(r_n).$$

We also consider a graph transducer CREATE-NODES that creates nodes $m_1, \dots, m_{t+s}$. We put $L_3 = L_2 \cdot L_c \cdot L'_c$ (this is the $\mathcal{C}_m$ part of the result of the application of RI).

(4) (membrane deletion):

Consider a vector $\boldsymbol{p} = (p_1, \dots, p_n)$, where $p_j = p(i_j)$ is defined as follows:

$$p(i_j) = \begin{cases} *, & \text{if } \exists (r_k, \boldsymbol{v}_k) \in RI \text{ s.t. } i_j \in \text{Delete}(r_k\langle \boldsymbol{v}_k \rangle), \\ e, & \text{if } \exists (r_k, \boldsymbol{v}_k) \in RI \text{ s.t. } (i_j, e) \in \text{Delete-and-Move}(r_k\langle \boldsymbol{v}_k \rangle), \\ i_j, & \text{otherwise.} \end{cases}$$

The first two cases correspond to those id's i_j for which the corresponding cells should be deleted. Note that in the second case the contents of cell i_j are supposed to be moved to cell e, but such a cell may also be deleted. We remark that for any p_k such that $p_k \neq i_k$, there is a value $z \in \mathbb{N} \cup \{*\}$ such that there is a sequence $x_1, \dots, x_m$ with $x_1 = p_k$, $x_j = p(x_{j-1})$ for $2 \le j \le m$, and $x_m = z$, with either $z = *$ or $p(z) = z$. We denote this by $z = \text{last}(i_j)$. The above affirmation follows from the fact that the Delete-and-Move relation (considered as a

parent relation) induces a forest on the id's of the cells that should be deleted. The roots of the obtained trees are given by the function last and they will collect the objects from all the cells in the tree (except those having $p(i_j) = *$, whose contents are lost).

Next, we describe how the contents are moved:
$L_4 = \{(i_1, l'_1, w''_1) \cdots (i_n, l'_n, w''_n)\}$ where $(i_j, l'_j, w'_j) \in L_3$, $1 \leq j \leq n$, and

$$w''_j = w'_j + \bigcup_{\text{last}(i_k)=i_j} w'_k.$$

The deletion of cells induces changes to the relation ρ. We collect these modifications as a graph transducer DELETE-NODES that will be run after the Change-Relation transducer. This transducer deletes all vertexes i_j such that $p_j \neq i_j$ as well as all edges that are incoming to these deleted nodes.

We also remove the corresponding cells from L_4:
$L_5 = (i_1, l'_1, w''_1) \ldots (i_{n_1}, l'_{n_1}, w''_{n_1})$ where $(i_j, l'_j, w''_j) \in L_4$ and $p_j = i_j$.

(5) (relation change) The new relation $\mathcal{C}'_\rho$ is computed by running the graph transducers CREATE-NODES, Change-Relation$(r_k\langle \boldsymbol{v}_k \rangle)$ and DELETE-NODES for all $(r_k, \boldsymbol{v}_k) \in$ RI on $\mathcal{C}_\rho$.

The output of the algorithm is the configuration $\text{Apply}(\text{RI}, \mathcal{C}) = (L_5, \mathcal{C}'_\rho)$.

4. Derivation modes

As was pointed out in [5], the computation of $\text{Applicable}(\Pi, \mathcal{C}, \delta)$ can be dissociated from the corresponding P system by considering the function $\text{Applicable}(\Pi, \mathcal{C})$ that gives the set of all multisets of rules of Π that can be applied together in the configuration $\mathcal{C}$. Then, the result of $\text{Applicable}(\Pi, \mathcal{C}, \delta)$ is obtained by a fixed restriction of $\text{Applicable}(\Pi, \mathcal{C})$, called *derivation mode*. For example, the maximally parallel derivation mode (max) traditionally used in the P system area can be defined by the following restriction:

$$\text{Applicable}(\Pi, \mathcal{C}, \max) = \{R \in \text{Applicable}(\Pi, \mathcal{C}) \mid \nexists R' \in \text{Applicable}(\Pi, \mathcal{C}) : R' \supset R\}.$$

This restriction imposes that all multisets of rules present in $\text{Applicable}(\Pi, \mathcal{C}, \max)$ are not extensible.

In the case of the framework presented in this article, the set Applicable contains a multiset of pairs (rule,instance). Hence, it is possible to use the instance information for additional types of restrictions. For example, we can consider the following derivation mode 1max:

$$\text{Applicable}(\Pi, \mathcal{C}, 1\max) = \{R \in \text{Applicable}(\Pi, \mathcal{C}, \max) \mid \forall r, \nexists (r, \boldsymbol{v}) \in R, \text{ with } \boldsymbol{v}|_j = 1 \text{ for some } j \in \mathbb{N}\}.$$

The above condition further restricts a maximally parallel set of rules by disallowing those sets whose rules involve cell x with $\text{id}(x) = 1$.

Such conditions can be further extended by allowing to restrict rules involving cells with some properties on the id or the labels of the cells they involve, as well as on the structure relation between corresponding cells. As an example, we could define a derivation mode that disallows for a group of rules to use cells having a particular label and that are not in the relation ρ:

$$\text{Applicable}(\Pi, \mathcal{C}, 2h) = \{R \in \text{Applicable}(\Pi, \mathcal{C}) \mid \forall r_1, r_2 \ \nexists (r_1, \boldsymbol{v}_1), (r_2, \boldsymbol{v}_2) \in R, \exists j, k \in \mathbb{N} \text{ with } (j, k) \notin \mathcal{C}_\rho \text{ and } \text{lab}(\boldsymbol{v}_1|_j) = \text{lab}(\boldsymbol{v}_2|_k) = h\}.$$

5. Taxonomy

In order to simplify the notation, instead of using a long tuple bringing everything together, we consider several variants of rule notation, adapted for the following specific types of rules:

Simple rewriting rule (R-*rule*). An R-rule is defined by the following components:

$$r = (\text{Labels}(r), \rho(r), \text{Rewrite}(r))$$

Simple rewriting rule with label rename (LR-*rule*). An LR-rule is defined by the following components:

$$r = (\text{Labels}(r), \rho(r), \text{Rewrite}(r), \text{Label-Rename}(r))$$

Simple creation rule (C-*rule*). A C-rule is defined by the following components:

$$r = (\text{Labels}(r), \rho(r), \text{Generate}(r), \text{Generate-}\textit{and}\text{-Copy}(r), \text{Change-Relation}(r))$$

Simple creation rule with label rename (CL-*rule*). A CL-rule is defined by the following components:

$$r = (\text{Labels}(r), \rho(r), \text{Label-Rename}(r), \text{Generate}(r), \text{Generate-and-Copy}(r), \text{Change-Relation}(r))$$

Simple dissolution rule (D-*rule*). A D-rule is defined by the following components:

$$r = (\text{Labels}(r), \rho(r), \text{Delete}(r), \text{Delete-and-Move}(r), \text{Change-Relation}(r))$$

In the case of the parent relation (tree case), we can simplify the rules and omit $\rho(r)$ by supposing that Labels(r) is of size 2. In this case, we implicitly assume that $(1, 2) \in \rho(r)$. The type of corresponding rules with the parent relation will additionally contain the letter P (e.g. PC-rule).

In a more general way, we can also consider combined types of rules, merging the corresponding components: L – label rename, R – rewriting, O – permitting and forbidding condition check, C – membrane creation, D – membrane deletion and E – change-relation (and, e.g. get RD rules).

In the case of the parent relation (for cell-like P systems), rules r having a non-empty Delete-and-Move(r) component can be simplified by reducing their Change-Relation(r) component.

In the above case, we assume that Change-Relation(r) contains the transducer MOVE-CONNECTIONS described below. This transducer adds the following edges to $\rho : \{(a_x, b_y) \mid (x, y) \in \mathcal{C}_\rho \text{ and } a_x, b_y \neq *\}$, where (a_x, b_y) is defined as follows (i_m is the id of the mth membrane):

$$(a_x, b_y) = \begin{cases} (\text{last}(x), y), & (x, y) \in \rho \text{ and } p_x \neq i_x, \\ (y, \text{last}(x)), & (y, x) \in \rho \text{ and } p_x \neq i_x. \end{cases}$$

The above transformations correspond to the deletion of cells and to the movement of their contents according to the Delete-and-Move relation.

6. Some examples

6.1 *Active membranes*

Let us start with the example of traditional active membrane rules (e.g. see Section 11.2 of the handbook of Păun *et al.* [11]).

Polarization can be treated in two ways – as a special object inside a membrane or like a special label; we consider here the latter case; that is, the couple (label, polarization) will be a new type of label.

Note that there is no distinction for polarizationless active membranes (according to Section 11.4 of the handbook of Păun *et al.* [11]), since in our case the label is a couple $\langle e, h\rangle$.

Thus, a rule $r : [a \to v]_h^e$ will be treated as $r : [a \to v]_{\langle e,h\rangle}$ and it can be translated as the following PR-rule:

$$\begin{aligned} r :\ \text{Labels}(r) &= (\langle e, h\rangle),\\ \text{Rewrite}(r) &= \{(1, a \to v)\}. \end{aligned}$$

In the following, we indicate e instead of $\langle e, h\rangle$.

A rule $a[]_{e_1} \to [b]_{e_2}$ can be translated as the following group of PLR-rules ($\forall p \in \text{Lab}$):

$$\begin{aligned} r :\ \text{Labels}(r) &= (e_1, p),\\ \text{Rewrite}(r) &= (2, a) \to (1, b),\\ \text{Label-Rename}(r) &= \{(1, e_2)\}. \end{aligned}$$

A rule $[a]_{e_1} \to []_{e_2} b$ can be translated as the following group of PLR-rules ($\forall p \in \text{Lab}$):

$$\begin{aligned} r :\ \text{Labels}(r) &= (e_1, p),\\ \text{Rewrite}(r) &= (1, a) \to (2, b),\\ \text{Label-Rename}(r) &= \{(1, e_2)\}. \end{aligned}$$

A rule $[a]_e \to b$ can be translated as the following group of PD-rules ($\forall p \in \text{Lab}$):

$$\begin{aligned} r :\ \text{Labels}(r) &= (e, p),\\ \text{Rewrite}(r) &= (1, a) \to (1, b),\\ \text{Delete-and-Move}(r) &= \{(1, 2)\}. \end{aligned}$$

A rule $[a]_{e_1} \to [b]_{e_2}[c]_{e_3}$ can be translated as the following group of PCLR-rules ($\forall p \in \text{Lab}$):

$$\begin{aligned} r :\ \text{Labels}(r) &= (e_1, p),\\ \text{Rewrite}(r) &= (1, a) \to (1, b),\\ \text{Label-Rename}(r) &= \{(1, e_2)\},\\ \text{Generate-and-Copy}(r) &= \{(1', e_3, 1, b \to c)\},\\ \text{Change-Relation}(r) &= \text{INSERT-EDGE}(1', 2). \end{aligned}$$

6.2 *Creation rules*

Creation rules (e.g. see Section 12.6 of the handbook of Păun *et al.* [11]) yield the creation of a new membrane inside the region where the rule is applied. The object triggering the rule is consumed,

and there is no replication of objects. The new membrane will contain only the multiset specified in the rule.

A rule $[a \to [u]_{h_1}]_{h_2}$ can be translated as the following PCR-rule:

$$\begin{aligned} r:\ \text{Labels}(r) &= (h_2),\\ \text{Rewrite}(r) &= (1,a) \to (1,\lambda),\\ \text{Generate}(r) &= \{(1',h_1,u)\},\\ \text{Change-Relation}(r) &= \text{INSERT-EDGE}(1',1). \end{aligned}$$

6.3 *Strong division*

A rule $[[]_{h_1} \dots []_{h_k}[]_{h_{k+1}} \dots []_{h_n}]_h \to [[]_{h_1} \dots []_{h_k}]_h[[]_{h_{k+1}} \dots []_{h_n}]_h$ can be defined as the following C-rule:

$$\begin{aligned} r:\ \text{Labels}(r) &= (h_1,\dots,h_n,h),\\ \rho(r) &= \{(i,n+1) \mid 1 \le i \le n\},\\ \text{Rewrite}(r) &= \emptyset\\ \text{Generate}(r) &= \{(1',h,\lambda)\},\\ \text{Generate-and-Copy}(r) &= \emptyset,\\ \text{Change-Relation}(r) &= \text{DELETE-EDGE}(i,n+1),\ \text{and}\\ &= \text{INSERT-EDGE}(i,1'),\ \text{for } k+1 \le i \le n. \end{aligned}$$

6.4 *Division based on polarizations*

Consider a rule of type $[]_h \to [+]_{h1}[-]_{h2}[0]_{h3}$ that regroups all membranes with the same polarization in three new membranes. This can be simulated with the following CL-rule:

$$\begin{aligned} r:\ \text{Labels}(r) &= (h),\\ \rho(r) &= \emptyset,\\ \text{Label-Rename}(r) &= (1,h_1),\\ \text{Generate}(r) &= \{(1',h_2,\lambda),(2',h_3,\lambda)\},\\ \text{Generate-and-Copy}(r) &= \emptyset,\\ \text{Change-Relation}(r) &= \text{DELETE-EDGE}(k,1),\ \text{and INSERT-EDGE}(k,1'),\\ &\quad \text{for all } k \text{ such that } \text{lab}(k) = \langle l,-\rangle\\ &\quad \text{DELETE-EDGE}(k,1),\ \text{and INSERT-EDGE}(k,2'),\\ &\quad \text{for all } k \text{ such that } \text{lab}(k) = \langle l,0\rangle. \end{aligned}$$

6.5 *Kernel P systems*

In this section, we would like to consider kernel P systems defined in [8]. This model corresponds to P systems with a dynamic structure (with a configuration from $\mathfrak{C}$) having the following types of rules:

(1) Rewriting and communication rule: $x \to y$, where x is a multiset, while the right-hand side, y, has the form $y = (a_1, t_1) \cdots (a_h, t_h)$, with t_j being the indexes of the target cells. This rule rewrites x to y and moves the objects to the corresponding cells.
(2) Input–output rule: this is a form of symport/antiport rule: (x/y). This rule exchanges multisets x and y being in two different cells.
(3) Structure change rules:
   (a) Membrane division rule: $[]_{l_i} \to []_{l_{i1}} \ldots []_{l_{ih}}$, the cell l_i will be replaced by h cells obtained from l_i, that is, the contents of the cells will coincide with that of l_i; their labels are $l_{i_1}, \ldots, l_{i_h}$, respectively; all the links of l_i are inherited by each of the newly created cells.
   (b) Membrane dissolution rule: $[]_{l_i} \to \lambda$; the cell l_i will be destroyed together with its links.
   (c) Link creation rule: $[]_{l_i}; []_{l_j} \to []_{l_i} - []_{l_j}$; the current compartment, l_i, is linked to l_j, and if more than one l_j exists, then one of them will be non-deterministically picked up.
   (d) Link destruction rule: $[]_{l_i} - []_{l_j} \to []_{l_i}; []_{l_j}$; this is the opposite of link creation and means that compartments l_i, l_j are disconnected.

In [8], rules also contain a guard g that corresponds to a union of the permitting and forbidding conditions.

We can represent the above rules in the formal framework as follows.

The rewriting rule corresponds to the RO-rule:

$$\begin{aligned} r:\ \text{Labels}(r) &= (l, l_1, \ldots, l_h), \\ \rho(r) &= \{(1,j) \mid 1 \le j \le h\}, \\ \text{Perm}(r) \cup \text{For}(r) &= g, \\ \text{Rewrite}(r) &= \{(1,x) \to (1,a_1) \cdots (h,a_h)\}. \end{aligned}$$

The input–output rule can be translated in a similar way.

The membrane division rule corresponds to the CDO-rule:

$$\begin{aligned} r:\ \text{Labels}(r) &= (l_i), \\ \rho(r) &= \emptyset, \\ \text{Perm}(r) \cup \text{For}(r) &= g, \\ \text{Generate-and-Copy}(r) &= \{(j', l_{i_j}, 1, \emptyset)\}, 1 \le j \le h, \\ \text{Delete}(r) &= \{1\}, \\ \text{Change-Relation}(r) &= \text{INSERT-EDGE}(j', x), \\ &\quad \text{for all } x \text{ such that } (1,x) \in C_\rho, 1 \le j \le h. \end{aligned}$$

The membrane dissolution rule corresponds to the DO-rule:

$$\begin{aligned} r:\ \text{Labels}(r) &= (l_i), \\ \rho(r) &= \emptyset, \\ \text{Perm}(r) \cup \text{For}(r) &= g, \\ \text{Delete}(r) &= \{1\}, \\ \text{Change-Relation}(r) &= \emptyset. \end{aligned}$$

The link creation rule corresponds to the OE-rule:

$$\begin{aligned} r:\ \text{Labels}(r) &= (l_i, l_j), \\ \rho(r) &= \emptyset, \\ \text{Perm}(r) \cup \text{For}(r) &= cg, \\ \text{Change-Relation}(r) &= \text{INSERT-EDGE}(1,2). \end{aligned}$$

The link destruction rule corresponds to the following *OE*-rule:

$$\begin{aligned} r:\ \text{Labels}(r) &= (l_i, l_j), \\ \rho(r) &= \{(1,2)\}, \\ \text{Perm}(r) \cup \text{For}(r) &= cg, \\ \text{Change-Relation}(r) &= \text{DELETE-EDGE}(1,2). \end{aligned}$$

7. Conclusions

In this article, we have presented a framework for P systems with a dynamic structure. The obtained meta-language has a precise semantics centred on two notions: (1) the evolution of the objects and membrane labels and (2) the evolution of the membrane structure (creation and deletion of nodes and edges). As a consequence, it permits to easily describe different features of existing P systems with a dynamic structure, which permits to provide an interesting tool for the comparison of different variants of P systems. Moreover, the translation to the framework allows for a deeper understanding of the corresponding P system and provides ways to extend its definition by new features. We remark that in the case of the systems with a static structure, a similar approach using the framework from [5] permitted to define new variants of P systems and to better express some existing ones [1,6].

The introduced model works with an arbitrary (binary) relation between membranes, so it remains open to consider relations different from the parent relation widely used in P systems. As an interesting candidate, we suggest the brother/sister relation on a tree. Another possible extension could be to address the generalization of the framework to an arbitrary n-ary relation. In this case, the relation ρ induces a hypergraph, so the components changing the structure of ρ have to be adapted to work on hypergraphs.

Finally, a challenging direction for the development of the framework is to consider that for a multiset of rules the order of the application of Change-Relation produces different results. This implies that the order of rules is important, and as a consequence, the set $\text{Applicable}(\Pi, C, \delta)$ will contain vectors (or lists) of rules. This idea has not yet been considered in the framework of P systems and we think that it can lead to interesting results.

Acknowledgements

The work of the second and the third authors was supported by Project TIN2009-13192 of the Ministerio de Ciencia e Innovación of Spain, cofinanced by FEDER funds and Project of Excellence with *Investigador de Reconocida Valía*, from Junta de Andalucía, grant P08 – TIC 04200.

References

[1] A. Alhazov, M. Oswald, R. Freund, and S. Verlan, *Partial halting and minimal parallelism based on arbitrary rule partitions*, Fund. Inform. 91(1) (2009), pp. 17–34.

[2] M. Cardona, M.A. Colomer, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, and D. Sanuy, *A P system based model of an ecosystem of some scavenger birds*, Lect. Notes Comput. Sci. 5957 (2010), pp. 182–195.
[3] R. Freund and B. Haberstroh, *Attributed elementary programmed graph grammars*, Lect. Notes Comput. Sci. 570 (1991), pp. 75–84.
[4] R. Freund, I. Pérez-Hurtado, A. Riscos-Núñez, and S. Verlan, *A formal framework for P systems with dynamic structure*, in *Tenth Brainstorming Week on Membrane Computing Sevilla, January 30–February 3, 2012*, Vol. I, M.A. Martínez-del-Amor, Gh. Păun, I. Pérez-Hurtado, and F.J. Romero-Campero, eds., Fénix Editora, Sevilla, 2012, pp. 111–122.
[5] R. Freund and S. Verlan, *A formal framework for static (Tissue) P systems*, Lect. Notes Comput. Sci. 4860 (2007), pp. 271–284.
[6] R. Freund and S. Verlan, *(Tissue) P systems working in the k-restricted minimally or maximally parallel transition mode*, Nat. Comput. 10(2) (2011), pp. 821–833.
[7] M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, and A. Riscos-Núñez, *An overview of P-Lingua 2.0*, Lect. Notes Comput. Sci. 5957 (2010), pp. 264–288.
[8] M. Gheorghe, F. Ipate, and C. Dragomir, *A kernel P system*, in *Tenth Brainstorming Week on Membrane Computing Sevilla, January 30–February 3, 2012*, Vol. I, M.A. Martínez-del-Amor, Gh. Păun, I. Pérez-Hurtado, and F.J. Romero-Campero, eds., Fénix Editora, Sevilla, 2012, pp. 153–170.
[9] M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, and A. Riscos-Núñez. *A P-lingua based simulator for tissue P systems*, J. Log. Algebr. Program. 79 (2010), pp. 374–382.
[10] Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
[11] Gh. Păun, G. Rozenberg, and A. Salomaa (eds.), *The Oxford Handbook Of Membrane Computing*, Oxford University Press, New York, 2010.
[12] G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Springer-Verlag, New York, 1997.
[13] *The Membrane Computing Web Page*: http://ppage.psystems.eu.
[14] *The P-Lingua web page*: http://www.p-lingua.org.