A Formal Framework for P Systems with Dynamic Structure

Rudolf Freund[1], Ignacio Pérez-Hurtado[2],
Agustín Riscos-Núñez[2], Sergey Verlan[3]

[1] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
Email: rudi@emcc.at

[2] Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
University of Sevilla,
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
Email: {perezh,ariscosn}@us.es

[3] LACL, Département Informatique, Université Paris Est,
61, av. Général de Gaulle, 94010 Créteil, France
Email: verlan@univ-paris12.fr

Summary. This article introduces a formalism/framework able to describe different variants of P systems having a dynamic structure. This framework can be useful for the definition of new variants of P systems with dynamic structure, for the comparison of existing definitions as well as for their extension. We give a precise definition of the formalism and show how existing variants of P systems with dynamic structure can be translated to it.

1 Introduction

This article is an attempt to fulfill the goal of defining a formal framework that captures the essential properties of P systems with dynamic structure. 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 dynamic structure. There are numerous possible applications of the results of such an analysis, as, for example, the comparison and the extension of existing models and the creation of new models of P systems with a dynamic structure.

The article extends the approach used in [3] for P systems with static structure. We recall that the framework for the 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, transition mode and halting condition. The configuration is a list of multisets corresponding to the contents of membranes of a P system and the rules generalize most kind of rules used in the P systems area. Based on this general form of rules, the applicability and the 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 C ($Applicable(\Pi, C)$). Then this set is restricted according to the transition mode δ ($Applicable(\Pi, C, \delta)$). 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 dynamic structure the first three ingredients are to be changed in order to accommodate with the fact that the structure of the system can change. Informally, a *configuration* is 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, i.e. 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 like 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 transition modes and halting conditions can easily be applied to this set exactly as in the static case.

The article is organized as follows. Section 2 gives the definition of the framework and presents the related algorithms. Section 3 presents a taxonomy that permits to define shortcuts for the commonly used cases. Then in section 4 we give examples of the translation of different types of P systems with dynamical structure. Finally, we discuss the perspectives of the presented approach.

2 Definitions

We assume that the reader is familiar with standard definitions in formal language theory (for example, we refer to [8] for all details) and with standard notions of P systems, as described in the books [5] and [6] or at the web page [7].

2.1 Graph transformations

There exist several ways to define a graph transformation. We will define a the graph transducer using the formalism from [2]. 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 labeled by X;
- $D(X)$: deletion of a node labeled by X;
- $C(X, Y)$: change the label of the node labeled by X to Y;
- $I(l_1, \lambda, l_2; l'_1, a, l'_2)$: insert an edge labeled by a between two nodes labeled by l_1 and l_2; after the insertion nodes are relabeled to l'_1 and l'_2 respectively;
- $D(l_1, a, l_2; l'_1, \lambda, l'_2)$: delete the edge labeled by a between two nodes labeled by l_1 and l_2; after the deletion nodes are relabeled 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 labeled by a between two nodes labeled by l_1 and l_2, After this operation nodes are relabeled to l'_1 and l'_2 respectively.

It was proved in [2] that the above formalism is computationally complete.

In what follows we will use some particular graph transducers whose definition we give below:

- $DELETE(x)$: $C(x, x')$, $D(x', a, y; x', \lambda, y)$ (looping over a and y), $D(x')$
- $INSERT(x)$: $I(x)$
- $INSERT - EDGE(x, y)$: $I(x, \lambda, y; x, a, y)$
- $DELETE - EDGE(x, y)$: $D(x, a, y; x, \lambda, y)$

2.2 Definition of the framework

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

Definition 1 *A* basic configuration *C (of size n) is a list $(i_1, w_1) \dots (i_n, w_n)$, where each w_j is a multiset (over O) and each $i_j \in \mathbb{N}$, $i_j \neq i_k$, for $k \neq j$, $1 \leq j, k \leq n$.*

If not stated otherwise, we suppose that all multisets of a basic configuration are finite. If needed the definitions that follow can be adapted to infinite multisets by adding corresponding constraints to the rule definition like it was done in [3].

The set of basic configurations of any size $n > 0$ is denoted by $\mathbb{C}$. We remark that we will consider only basic configurations of finite size and we denote the size of C by size(C).

Each element (i_j, w_j), $1 \leq j \leq n$, of a configuration C is called a *cell*. We say that i_j is the *id* of the cell j and that w_j is the contents of the cell j. We define the function $id(x)$ which for a cell x returns its id. We require the function id to

be bijective, i.e., there should be a one-to-one correspondence between cells and their id's.

If not stated otherwise, we will consider that id is the identity function ($id(x) = x$), so we will not distinguish between an id and its position in the list of configurations.

Definition 2 *A* configuration $\mathcal{C}$ *is the couple* (L, ρ)*, where* $L \in Lab \times \mathbb{C}$ *is the list* $(i_1, l_1, w_1) \dots (i_n, l_n, w_n)$ *with* (i_j, w_j) *corresponding to an element of a basic configuration and* $l_j \in Lab$ *being the* label *of the cell,* $1 \leq j \leq n$ *(Lab is a set of labels). The second component* $\rho \subseteq \mathbb{N} \times \mathbb{N}$ *is the relation graph between cells.*

Hence in a configuration any cell has an id (equal to the position) which is unique and a label which is not necessarily unique. We define the function $lab(x) : \mathbb{N} \to L$ that returns the label of the cell having the id equal to x. We denote by $\mathcal{C}_m$ and $\mathcal{C}_\rho$ the first and the second components of the configuration $\mathcal{C}$. We also denote by $\bar{\mathcal{C}}_m \in \mathbb{C}$ the projection of $\mathcal{C}_m$ erasing the labels (yielding a basic configuration).

The relation ρ is defined on id's of cells being part of the configuration. In P systems this corresponds to the parent relation, while in tissue P systems this corresponds to the communication graph of the system.

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

Now we will give the definition of a rule. A rule r is defined by the following components. We remark that all of them are given in terms of virtual cell positions. We will also call them **relative** positions because they are introduced in the first component of the rule.

A. **Checking**
1. $Labels(r) \in Lab^*$ ($Labels(r) = (l_1, \dots, l_k)$) is a list of cell labels. This list identifies k (relative) positions labeled from 1 to k that we further call virtual cells. Let $\mathbb{N}_k = \{1, \dots, k\}$ and $\mathbb{K}$ be a subset of $\mathbb{C}$ where for any cell x it holds $1 \leq id(x) \leq k$.
2. $\rho(r) \subseteq \mathbb{N}_k \times \mathbb{N}_k$ is the constraint imposed by the (parent) relation on the virtual cells.
3. $Perm(r) \subseteq \mathbb{K}$ defines the permitting condition.
4. $For(r) \subseteq \mathbb{K}$ defines the forbidding condition.

B. **Modification of existing configuration/structure**
5. $Rewrite(r) \in (\mathbb{K} \times \mathbb{K})$ is a general rewriting rule permitting to rewrite a finite basic configuration to another one (e.g., $(j, u)(i, v) \to (m, w)$). By $Bound(r)$ we denote the first component (the left-hand-side) of this rewriting rule.
6. $Label\text{–}Rename(r) \in (\mathbb{N}_k \times Lab)^*$ renames the labels specified by the list.
7. $Delete(r) \in \mathbb{N}_k^*$ gives the indexes of virtual cells to be deleted.
8. $Delete\text{–}and\text{–}Move(r) \in (\mathbb{N}_k \times \mathbb{N}_k)^*$ is a list of couples of indices (e.g., (j, k)) indicates that the virtual cell j should be deleted and its contents should be moved to the virtual cell k).

C. **Creation of new structures**

9. $Generate(r) \in (\mathbb{N}' \times Lab \times O^\circ)^*$ 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. $Generate–and–Copy(r) \in (\mathbb{N}' \times Lab \times \mathbb{N} \times \bar{\mathcal{R}})$ -is a list of quadruplets consisting of a (primed) index, a label, an index, and a rewriting rule (e.g. $(j', h, i, u \to v)$). This component specifies new cells to be created by duplicating existing cells.

    We denote the smallest multiset containing any left-hand side of rewriting rules from *Generate–and–Copy* by $DPerm(r)$.

D. **Structure transformation**

11. *Change–Relation* 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).

Now we define what means the applicability of a rule. Before giving the algorithm, we define some additional notions related to relative positions.

An *instance* of size n is a vector of integers $I = (i_1, \ldots, i_n)$, $i_j \in \mathbb{N}$, $1 \leq j \leq n$. By size(I) we denote the size of an instance I, and by $I|_k$, $1 \leq k \leq n$, the k-th value of the vector I, i.e., i_k.

For a basic configuration $C \in \mathbb{C}$, $C = (j_1, w_1) \ldots (j_k, w_k)$, and for an instance I we define the *instantiation* of C by I, denoted $C\langle I \rangle$, as follows:

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

In the above formula we assume that the cells of configuration C do not necessarily have their id in the range $[1 \ldots \text{size}(C)]$. We also remark that size$(C) \leq$ size(I).

It is clear that if C is defined in terms of relative positions then $C\langle I \rangle$ permits to replace these relative positions by the corresponding values from I (a relative position k is replaced by $I|_k$ which is i_k).

For a rule r as defined above and for an instance I such that $|Labels(r)| \leq$ size(I) we obtain the instantiation of r by I, denoted by $r\langle I \rangle$, by replacing all relative positions k by $I|_k$ in $Perm(r)$, $For(r)$, $Rewrite(r)$, $Label–Rename(r)$, $Delete(r)$, $Delete–and–Move(r)$ and $Change–Relation(r)$.

Now we define what means the applicability of a group of rules. First we define the set of *valid instances* for a rule $r \in \mathbb{R}$ in a configuration $\mathcal{C}$. This set, denoted by $\mathcal{I}_\mathcal{C}(r)$, is obtained by the following algorithm.

1. Getting instances in conformity with $Labels(r)$:

$$\bar{\mathcal{I}}_\mathcal{C}(r) = \{(i_1, \ldots, i_k) \mid (l_1, \ldots, l_k) = Labels(r) \text{ and } lab(i_j) = l_j, \\ 1 \leq i_j \leq \text{size}(\mathcal{C}), 1 \leq j \leq k\}.$$

2. Checking the relation ρ:

$$\mathcal{I}_\mathcal{C}(r) = \{(i_1, \ldots, i_k) \in \bar{\mathcal{I}}_\mathcal{C}(r) \mid (j, m) \in \rho(r) \Rightarrow (i_j, i_m) \in \mathcal{C}_\rho\}.$$

For a multiset of rules $R \in \mathbb{R}^\circ$ and a configuration $\mathcal{C} \in \mathfrak{C}$ we define the set of multisets $\mathit{Applicable}(R, \mathcal{C}) \subseteq (\mathbb{R} \times \mathbb{N}^*)^\circ$ giving the set of multisets of instantiated rules that can be computed based on R and the configuration $\mathcal{C}$. This set is computed as follows.

Let $R = \{r_1, \ldots, r_n\}$ (the rules are not necessarily different) and let $\mathcal{I}_\mathcal{C}(r_i) = (v_{i,1}, \ldots, v_{ik_i})$, $1 \leq i \leq n$. Consider an arbitrary vector of rule instances $v = (v_{1,j_1}, \ldots, v_{n,j_n})$, $1 \leq j_i \leq k_i$, $1 \leq i \leq n$. The multiset $\{(r_1, v_{1,j_1}), \ldots, (r_n, v_{n,j_n})\}$ is added to $\mathit{Applicable}(R, \mathcal{C})$ if

- For all $p \in \mathit{Perm}(r_i) \cup \mathit{DPerm}(r_i)$, $p\langle v \rangle \subseteq \bar{\mathcal{C}}_m$, $1 \leq i \leq n$.
- For all $q \in \mathit{For}(r_i)$, $q\langle v \rangle \not\subseteq \bar{\mathcal{C}}_m$, $1 \leq i \leq n$.
- $\bigcup_{i=1}^n \mathit{Bound}(r_i)\langle v \rangle \subseteq \bar{\mathcal{C}}_m$.
- The consecutive application of graph transducers $\mathit{Change–Relation}(r_i)$ and $\mathit{Change–Relation}(r_j)$ yields the same result regardless of the order of the application, $1 \leq i, j \leq n$.

It is clear that there is a bound on the size of the multiset of rules R for which $\mathit{Applicable}(R, \mathcal{C})$ is not empty. We denote by $\mathit{Applicable}(\mathcal{C}) \subseteq (\mathbb{R} \times \mathbb{N}^*)^\circ$ the union of corresponding multisets:

$$\mathit{Applicable}(\mathcal{C}) = \bigcup_{\mathit{Applicable}(R,\mathcal{C}) \neq \emptyset} \mathit{Applicable}(R, \mathcal{C}).$$

For a P system Π having a set of rules R we define $\mathit{Applicable}(\Pi, \mathcal{C}) = \mathit{Applicable}(R, \mathcal{C})$. Following [3] it is possible to define now the transition modes as a restriction of this set. However, it should be noted that since the corresponding multisets contain instantiated rules, additional restrictions based on instances can be placed.

Now we are ready to define the application of a multiset of rules R.

Let $\mathcal{C} = (L, \rho)$ be the current configuration and let $RI \in \mathit{Applicable}(R, \mathcal{C})$, $RI = \{(r_1, v_1), \ldots, (r_n, v_n)\}$ be a multiset of instantiated rules. We now define the operation $\mathit{Apply}(RI, \mathcal{C}) \in \mathfrak{C}$ which is the result of the application of RI to C.

Before giving the algorithm we remark that a rule is composed from 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, e.g. 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, *i.e.* 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 a further research.

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

1. (rewriting application): $L_1 = \{(i_1, l_1, w_1') \ldots (i_n, l_n, w_n')\}$ where:

$$w'_j = w_j + \bigcup_{\substack{(r_k, v_k) \in RI \\ (s, u \to v) \in Rewrite(r_k) \\ v_k|_s = j}} (-u + v).$$

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

$$l'_j = \begin{cases} e_s, \text{ there is } (r_k, v_k) \in RI \text{ such that } (s, e_s) \in Label\text{–}Rename(r_k) \\ \quad \text{and } v_k|_s = j, \\ l_j, \quad \text{otherwise.} \end{cases}$$

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

$$\begin{aligned} L_c(r_k) =& (m_1, h_1, u_1) \dots (m_t, h_t, u_t), \quad (r_k, v_k) \in RI \text{ and} \\ & Generate(r_k) = \{(1', h_1, u_1) \dots (t', h_t, u_t)\}. \\ L_c =& \prod_{k=1}^{n} L_c(r_k). \end{aligned}$$

$L'_c(r_k) = (m_{t+1}, h_{t+1}, w'_{n_1} - u_1 + v_1) \dots (m_{t+s}, h_{t+s}, w'_{n_s} - u_s + v_s)$, where $(r_k, v_k) \in RI$ and

$$\begin{aligned} Generate\text{–}and\text{–}Copy(r) =& \{((t+1)', h_{t+1}, n_{t+1}, u_{t+1} \to v_{t+1}) \dots \\ & ((t+s)', h_{t+s}, n_{t+s}, u_{t+s} \to v_{t+s})\}, \\ & (i_j, l'_j, w'_j) \in L_2, 1 \le j \le n \end{aligned}$$

$$L'_c = \prod_{k=1}^{n} L'_c(r_k).$$

By definition, we put $v_k|_{q'} = m_q$, $1 \le q \le t + s$.
We also consider a graph transducer $CREATE\text{–}NODES$ 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 R).

4. (membrane deletion):
Consider a vector $P = (p_1, \dots, p_n)$ defined as follows:

$$p_j = \begin{cases} *, & \text{if there exists } (r_k, v_k) \in RI \text{ such that } s \in Delete(r_k) \\ & \text{and } v_k|_s = j, \\ v_k|_m, & \text{if there exists } (r_k, v_k) \in RI \text{ such that} \\ & (s, m) \in Delete\text{–}and\text{–}Move(r_k) \text{ and } v_k|_s = j, \\ j, & \text{otherwise.} \end{cases}$$

The first two cases correspond to those ids j for which the corresponding cells should 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 $x_1, \ldots, x_m$ with $x_1 = p_k$, $x_m = z$, and $x_j = p_{x_{j-1}}$, $2 \leq j \leq m$. We denote this by $z = last(x)$. The above affirmation follows from that fact that the *Delete–and–Move* relation (considered as a parent relation) induces a forest on the ids 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 (if they are different from $*$).
Next we describe how the contents is moved:
$L_4 = \{(i_1, l'_1, w''_1) \ldots (i_n, l'_n, w''_n)\}$ where $(i_k, l'_k, w'_k) \in L_3$, $1 \leq k \leq n$, and

$$w''_j = w'_j + \bigcup_{last(k)=j} w'_k.$$

The deletion of cells induces changes to the relation ρ. We collect these modifications as a graph transducer $DELETE\text{–}NODES$ that will be run after the *Change–Relation* transducer. This transducer deletes all vertices j such that $p_j \neq j$ 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 $CREATE\text{–}NODES$, $Change\text{–}Relation(r\langle v_k \rangle)$ and $DELETE\text{–}NODES$ for all $(r_k, v_k) \in R$ on $\mathcal{C}_\rho$.

3 Taxonomy

In order to simplify the notation we consider several variants of rule notation:

Simple rewriting rule (R-rule)

An R-rule is defined only by the following components:
$r = (Labels(r), \rho(r), Rewrite(r))$

Simple rewriting rule with label rename (LR-rule)

An LR-rule is defined only by the following components:
$r = (Labels(r), \rho(r), Rewrite(r), Label\text{–}Rename(r))$

Simple creation rule (C-rule)

A C-rule is defined by the following components:
$r = (Labels(r), \rho(r), Generate(r), Generate\text{–}and\text{–}Copy(r),$
$Change\text{–}Relation(r))$

Simple creation rule with label rename (CL-rule)

A CL-rule is defined by the following components:
$r = (Labels(r), \rho(r), Label–Rename(r), Generate(r),$
$Generate–and–Copy(r), Change–Relation(r))$

Simple dissolution rule (D-rule)

A D-rule is defined by the following components:
$r = (Labels(r), \rho(r), Delete(r), Delete–and–Move(r),$
$Change–Relation(r))$

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 parent relation will additionally contain the letter P (e.g., PC-rule).

In a more general way we can combine several components: L – label rename, R – rewriting, C – membrane creation, D – membrane deletion (and get RD rules for example).

Rules r having a non-empty $Delete–and–Move(r)$ component can be simplified by reducing their $Change–Relation(r)$ component in the case of the parent relation.

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

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

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

4 Some examples

4.1 Active membranes

Let us start with the example of traditional active membrane rules (e.g., see Section 11.2 from handbook).

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

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:

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

In the future 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 ($\forall p \in Lab$):

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

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

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

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

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

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

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

4.2 Rules without polarizations

(According to Section 11.4 from the handbook). Since in our case the label is a couple $\langle e, h\rangle$, there is no distinction with respect to the previous case.

4.3 Creation rules

Consider creation rules like on p. 326 in handbook.

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

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

4.4 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:

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

4.5 Division based on polarizations

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

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

5 Conclusions

In this paper we presented a framework for P systems with dynamic structure. The obtained meta-language has a precise semantics centered around 2 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 dynamical 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 better 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 [3] permitted to define new variants of P systems and to better express some existing ones [1, 4].

The introduced model works with an arbitrary (binary) relation between membranes, so it could be interesting 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. It could also be interesting to consider a 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.

Another 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, by consequence the set $Applicable(\Pi, C, \delta)$ will contain vectors (or lists) of rules. This interesting idea was not yet considered in the framework of P systems and we think that it can lead to interesting results.

References

1. A. Alhazov, M. Oswald, R. Freund, S. Verlan, Partial Halting and Minimal Parallelism Based on Arbitrary Rule Partitions, *Fundamenta Informaticae* **91**(1), 2009, 17–34.
2. R. Freund, B. Haberstroh, Attributed Elementary Programmed Graph Grammars, Proceedings 17th Intern. Workshop on Graph-Theoretic Concepts in Computer Science, *Lecture Notes in Computer Science* **570**, Springer, 1991, 75–84.
3. R. Freund, S. Verlan, A Formal Framework for Static (Tissue) P Systems, Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **4860** , 271–284, Springer, 2007.
4. R. Freund, S. Verlan, (Tissue) P systems working in the k-restricted minimally or maximally parallel transition mode, *Natural Computing* **10**(2), 2011, 821–833.
5. Gh. Păun, *Membrane Computing. An Introduction*. Springer–Verlag, 2002.
6. G. Păun, G. Rozenberg, A. Salomaa, *The Oxford Handbook Of Membrane Computing*. Oxford University Press, 2009.
7. *The Membrane Computing Web Page*: `http://ppage.psystems.eu`
8. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*. Springer–Verlag, Berlin, 1997.