Using the Formal Framework for P Systems

Sergey Verlan

[1] Laboratoire d'Algorithmique, Complexité et Logique,
Université Paris Est – Créteil Val de Marne,
61, av. gén. de Gaulle, 94010 Créteil, France
[2] Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova,
Academiei 5, Chisinau, MD-2028, Moldova
verlan@u-pec.fr

Abstract. In this article we focus on the model called the *formal framework for P systems*. This model provides a descriptional language powerful enough to represent in a simple way, via a strong bisimulation, most of the variants of P systems. The article presents a series of concrete examples of the application of the formal framework in order to understand, extend, compare and explain different models of P systems leading to new research ideas and open problems.

1 Introduction

The model called *the formal framework for P systems* (FF) was introduced in [4] and later developed in [3]. It aims to provide a concrete variant of P systems that can act as descriptional language powerful enough to represent in a simple way most of the variants of P systems with the goal of better understanding and comparison of different models of P systems.

The formal framework permits to simulate most of variants of P systems. Moreover, in most cases it is a strong bisimulation, i.e. one step in the original system is done by one step in the formal framework. This becomes possible because the form of configurations and rules is close to multiset rewriting and generalizes most common configuration changes in P systems. Hence, most of existing models of P systems could be obtained by a restriction (eventually using a simple encoding) of FF with respect to different parameters. The strong bisimulation property also permits a discussion about the semantics of the target P system, although this is not the primary goal of FF.

Using FF mainly benefits for the following cases (a) understand the functioning of some variant of P systems; (b) compare variants of P systems; (c) explain points of the definition and semantics that can have different interpretations; (d) extend variants of P systems with new features.

The aim of this paper is not to present the framework itself, but rather several examples of its application for the description and the comparison of different variants of P systems with static structure, with probabilities and with dynamic structure. We also show how these investigations give a uniform view on P systems and lead to new research ideas and open problems.

A. Alhazov et al. (Eds.): CMC 2013, LNCS 8340, pp. 56–79, 2014.

2 A Short Presentation of the Formal Framework

We assume that the reader is familiar with basic notions on formal languages and on P systems and we refer to [9] and [8] for missing details. We will use a string notation to denote multisets and we denote the set of finite multisets over an alphabet V as V°. For a multiset M we denote by $|M|$ its size and by $card(M)$ its cardinal (i.e. the number of different occurring symbols in M). By $|M|_x$ we denote the number of occurrences of symbol x in M.

Before giving the definition of the formal framework we would like to make some remarks about the definition of different variants of P systems. Informally speaking, 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 like labels, charges, etc),
- a list of the initial multisets of objects present in each compartment at the beginning of the computation,
- a set of rules, acting over objects and / or over the structure.

The configuration of a P system is generally representing the contents of each compartment and the current structure (if it can be modified).

A computation of a P system can be defined as a sequence of transitions between configurations ending in some halting configuration. To give a more precise description of the semantics we must define the following 4 notions (functions):

- $Applicable(\Pi, \mathcal{C}, \delta)$ – the set of multisets of rules of Π applicable to 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.
- $Result(\Pi, \mathcal{C})$ – 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 (generally this is the maximally parallel mode) is defined as follows: we pass from a configuration $\mathcal{C}$ to $\mathcal{C}'$ (written as $\mathcal{C} \Rightarrow \mathcal{C}'$) iff

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

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, gathering all possible derivations). Note that this is a theoretical (non-constructive) definition, since there may exist an infinite number of halting configurations reachable from a single initial configuration $\mathcal{C}_0$.

The precise definition of the four functions above depends on the selected model of P systems. The goal of works [3,4,12] is to provide a concrete class of P systems (hence with concrete definitions of these functions), called the *formal framework*, such that most of existing models of P systems could be obtained by a strong bisimulation of a restriction (eventually using a simple encoding) of this formal framework with respect to different parameters.

In the remainder of this section we give a summarized version of the definition of a *network of cells*, the class containing all networks of cells being the formal framework. We base the definitions on those given in [4] and we will call the obtained model *FF1*. This version takes into account only P systems where the membrane structure does not evolve in time (is static). In paper [3], an extension of the formal framework to the case of P systems with dynamically evolving structure is proposed (we will call this version of the definition *FF2*). However, in order to have a more simple presentation, in this paper we will only consider the FF1 variant, except for Section 6 which is devoted to the dynamical extension of FF and therefore uses the FF2. We remark that in the case of static structures FF1 and FF2 variants coincide, although the notation is slightly different.

Definition 1 ([4]). *A* network of cells of degree $n \geq 1$ *is a construct*

$$\Pi = (n, V, w, Inf, R)$$

where

1. n *is the number of cells;*
2. V *a* finite alphabet*;*
3. $w = (w_1, \ldots, w_n)$ *where* $w_i \in V^\circ$*, for all* $1 \leq i \leq n$*, is the* finite multiset initially associated to cell i*;*
4. $Inf = (Inf_1, \ldots, Inf_n)$ *where* $Inf_i \subseteq V$*, for all* $1 \leq i \leq n$*, is the* set of symbols occurring infinitely often in cell i *(in most of the cases, only one cell, called the* environment*, will contain symbols occurring with infinite multiplicity);*
5. R *is a finite set of rules of the form*

$$(X \to Y; P, Q)$$

where $X = (x_1, \ldots, x_n)$*,* $Y = (y_1, \ldots, y_n)$*,* $x_i, y_i \in V^\circ$*,* $1 \leq i \leq n$*, are vectors of multisets over* V *and* $P = (p_1, \ldots, p_n)$*,* $Q = (q_1, \ldots, q_n)$*,* p_i, q_i*,* $1 \leq i \leq n$ *are finite sets of multisets over* V*. We will also use the notation*

$$(x_1, 1) \ldots (x_n, n) \to (y_1, 1) \ldots (y_n, n) \, ; [(p_1, 1) \ldots (p_n, n)]; [(q_1, 1) \ldots (q_n, n)]$$

for a rule $(X \to Y; P, Q)$*; moreover, if some* p_i *or* q_i *is an empty set or some* x_i *or* y_i *is equal to the empty multiset,* $1 \leq i \leq n$*, then we may omit it from the specification of the rule.*

The semantics of the above rule is to rewrite objects x_i from cells i into objects y_j in cells j, $1 \leq i, j \leq n$, if every cell k, $1 \leq k \leq n$, contains all multisets from

p_k and does not contain any multiset from q_k. In other words, the first part of the rule specifies the rewriting of symbols, the second part of the rule specifies permitting conditions and the third part of the rule specifies the forbidding conditions.

For a rule r of the form above, the set

$$\{i \mid x_i \neq \lambda \text{ or } y_i \neq \lambda \text{ or } p_i \neq \emptyset \text{ or } q_i \neq \emptyset\}$$

induces a (hypergraph) relation between the interacting cells. However, this relation need not give rise to a *structure* relation like a tree as in P systems or a graph as in tissue P systems.

A *configuration* C of Π is an n-tuple of multisets over V $(u_1, \ldots, u_n)$ satisfying $u_i \cap Inf_i = \emptyset$, $1 \leq i \leq n$.

Example 1. Consider the network of cells $\mathfrak{C}$ having 5 cells and the configuration $C = (ba, c, a, \lambda, \lambda)$. Suppose that $\mathfrak{C}$ has the following rule:
$r = (1, a)(2, c) \rightarrow (1, c)(4, a)(5, b); [(1, b)]; [(3, d)]$.
Then $C \Longrightarrow^r C'$, where $C' = (bc, \lambda, a, a, b)$.

The semantics of network of cells is defined as follows (see [4] for more details):

Applicable(Π, C, δ): An algorithm is used to compute *Applicable*$(\Pi, C, asyn)$, the set of multisets of all possible (parallel) applications of rules, which correspond to the set of multisets applicable in the asynchronous mode $(asyn)$. Then this set is (set-)restricted according to δ. As well known examples of δ we can cite max, seq, min, min_k.

Apply(Π, C, R): The application is performed using an algorithm. In the dynamical case (in FF2 definition) there are several variants.

$Halt(\Pi, C, \delta)$: This function is not specified in the definition and is defined separately. Several examples include total halting (no rule is applicable), signal halting (the configuration has some properties) and adult halting (no changes in the configuration occur).

Result(Π, C): This function is not specified in the definition and is defined separately. Generally it is the contents of some cell.

2.1 Comparison with Multiset Rewriting

It is known that any variant of static P systems can be seen as multiset rewriting: an object x in membrane i corresponds to a symbol x_i and each rule moving or rewriting x in membrane i, can be rewritten as corresponding multiset rewriting involving x_i. For example an antiport P system with 3 membranes arranged in the structure $[_1[_2]_2[_3]_3]_1$, the initial configuration (bc, λ, a) and a rule $(a, out; b, in)$ in membrane 3 can be rewritten as the following multiset rewriting: starting multiset $b_1c_1a_3$ and a rule $a_3b_1 \rightarrow a_1b_3$.

However, considering a P system like a multiset rewriting loses the important structural information. For example, try to figure out what happens in the system defined as follows.

Example 2. Consider the multiset rewriting system with the starting multiset $a_1b_2c_3$ and the rules $a_1b_2 \to a_2b_1$, $a_1c_2 \to a_2c_1$, $a_2c_3 \to a_3c_2$, $a_2b_3 \to a_3b_2$, $a_3c_1 \to a_1c_3$, $a_3b_1 \to a_1b_3$.

The formal framework groups the information in cells/membranes, does a group rewriting and represents the structure of the P system separately. So it is extremely close to the multiset rewriting, it just reorders objects and rules. This permits to keep the information about the static structure: rules induce a hypergraph. The communication graph can be deduced from this hypergraph. A similar approach is used in Petri nets, for example a multiset rewriting rule $aabc \to cde$ is represented as shown in Figure 1.

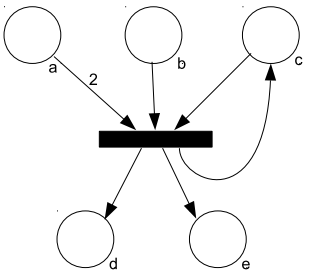

Fig. 1. A Petri net representation of the rule $aabc \to cde$

Example 3. Consider the system from Example 2. By rewriting the rules in terms of network of cells we obtain the following rules:

$$(1,a)(2,b) \to (2,a)(1,b) \qquad (1,a)(2,c) \to (2,a)(1,c)$$
$$(2,a)(3,c) \to (3,a)(2,c) \qquad (2,a)(3,b) \to (3,a)(2,b)$$
$$(3,a)(1,c) \to (1,a)(3,c) \qquad (3,a)(1,b) \to (1,a)(3,b)$$

Consider the hyperedge induced by the first rule: it goes from the cells 1 and 2 to cells 1 and 2. So we can make a supposition that we could have a communication graph that would contain an edge $1-2$. By looking at what the rule is doing we remark that it exchanges symbols a and b located in cells 1 and 2 respectively. Hence it corresponds to an antiport rule a/b on the edge $1-2$. By repeating this process for all above rules we obtain the antiport tissue P system shown in Figure 2. Hence it is clear that the system is moving symbol a in clockwise direction and symbols b and c in anticlockwise direction.

3 Implementing Different Features of P Systems

In this section we discuss the implementation of some features of P systems that are not present by default in the framework.

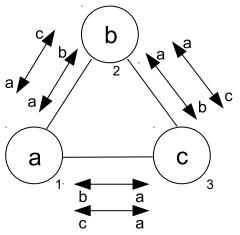

Fig. 2. The antiport system obtained in Example 3. The arc $a \leftrightarrow b$ corresponds to the antiport rule a/b.

3.1 New Derivation Modes

In order to define a new derivation mode for a P system it is sufficient to consider the network of cells equivalent to that system and to provide a set restriction for asynchronous (asyn) mode. Then, because of the bisimulation, the definition can immediately be interpreted in the corresponding P system.

Example 4. In this example we define two derivation modes: *minimally parallel mode restricted to partitions of size 1* (min_1) and *mixed set minimally parallel mode restricted to partitions of size 1* ($msmin_1$). In order to do this we assume that the ruleset R is divided into several sets $R_1, \ldots, R_m$, $m > 0$, such that $R = \bigcup_{1 \leq i \leq m} R_i$. Due to historical reasons we will call these sets *partitions* although this term is not accurate because the sets $R_1, \ldots, R_m$ are not necessarily disjoint.

The min_1 mode is defined as follows (see also [5]):

$$\begin{aligned} Applicable(\Pi, C, min_1) = \{S \in Applicable(\Pi, C, asyn) : |S \cap R_i| \leq 1, 1 \leq i \leq m \\ \text{and } \nexists S' \in Applicable(\Pi, C, asyn), \\ \text{with } |S' \cap R_i| \leq 1 \text{ such that } S' \supsetneq S\} \end{aligned}$$

Hence, the min_1 mode is in some sense requiring to take at most one rule from each partition, when possible. It coincides with the definition of the minimally applicable multiset of rules from Section 1.9 of [9].

For the $msmin_1$ mode we additionally classify the partitions into two categories: *-partitions and 1-partitions. In order to simplify the definition we suppose that R is divided into m partitions $R_1, \ldots, R_m$ and that the first k partitions are 1-partitions and the partitions from $k+1$ until m are *-partitions. Then the mode is defined as follows:

$$\begin{aligned} Applicable(\Pi, C, msmin_1) = \{S \in Applicable(\Pi, C, asyn) : \\ \text{such that for all } i, j \text{ where } 1 \leq i \leq k < j \leq m, \\ |S \cap R_i| \leq 1 \text{ and } card(S \cap R_j) \leq 1, \\ \text{and } \nexists S' \in Applicable(\Pi, C, asyn), \\ \text{with } |S \cap R_i| \leq |S' \cap R_i| \leq 1, \\ card(S \cap R_j) \leq card(S' \cap R_j) \leq 1 \text{ and } S' \supsetneq S\} \end{aligned}$$

The difference between the two definitions is that in $msmin_1$ mode one rule is chosen and applied from each 1-partition, if possible, and one rule is chosen and applied a maximal number of times from each *-partition, if possible.

Example 5. Consider a symport/antiport P system with a mode that ensures that a cell is used sequentially, only in a single operation. This can be done by using a partition of rules such that a rule involving cell i, will be a part of partition i. Hence, each rule will be in two partitions. The desired result is then obtained by applying the min_1 mode with the obtained partitions.

3.2 Membrane Thinkness/Polarization/Labels

We remark that the notions of membrane thickness, polarization and label are related to each other and designate the property of a membrane to be in some finite state. In order to be able to simulate efficiently these concepts we introduce into each membrane a special object coding the state of the membrane. All rules involving a membrane will additionally have a permitting context (promoter) checking this state object.

Example 6. Consider following active membranes rules (1) $[a \to bc]_h$ and (2) $a[]_h \to [b]_{h'}$ in membrane k. They are simulated in the formal framework by the following rules: $(k, a) \to (k, bc); [(k, h)]$ and $(k', a)(k, h) \to (k, bh')$, where k' is the parent of k.

As we can see the change of the state is done directly by the rule (case (2)). However, it should be noted that in the above implementation only one state change per membrane can occur in one step which is consistent with actual definitions used in P systems.

We remark that in the example above the rules (1) and (2) become cooperative after translation. This translates the intuitive idea that the object that is communicated/rewritten is cooperating with membrane state at the level of the rule.

We would like to remark that in FF2 the membrane state is an explicit part of the configuration, so no special object is needed for its representation. This is done because in P systems with dynamic structure the membrane labels are always used, so considering them as a part of configuration permits to save space in the description of the rule. However, it shall be noted that like in the case above there is an implicit cooperation between the membrane state and the objects used in the rule.

3.3 Priorities

Already in the first models of P systems a priority relation on the rules of the system was considered. The underlying relation is a strict partial order (i.e. an irreflexive, asymmetric, and transitive). We consider here two notions of priority following [6], the *strong* priority and the *weak* priority. Under the semantics of

strong priority, if a rule with higher priority is used, then no rule of a lower priority can be used even if the two rules do not compete for objects. For weakly prioritized systems, a rule is applicable if it cannot be replaced by a higher priority one. In the original definition of transitional P systems from [7] the strong priority is used.

Example 7. Consider a transitional P system which has following three rules: $r_1 : ab \to cd$, $r_2 : ac \to bd$ and $r_3 : bc \to aa$. Let the priority relation be $r_3 > r_1$. Suppose that the current configuration contains the multiset $aabbc$ in the corresponding membrane. Then in the case of the strong priority only rules r_2 or r_3 are applicable. In the case of the weak priority it is possible to apply additionally rule r_1, yielding the following applicable sets: $\{r_1, r_2\}$ and $\{r_1, r_2\}$.

It is not difficult to see that the strong priority corresponds directly to forbidding conditions: indeed $r_1 > r_2$ corresponds to two rules (1) r_1 and (2) r_2 enriched with forbidding sets containing the left-hand sides of all rules $r > r_2$.

Example 8. Consider the system from Example 7. We can translate the first rule to the formal framework as follows: $r'_1 : (k, ab) \to (k, cd); []; [(k, bc)]$.

The case of weak priorities can be handled using a special derivation mode that keeps track of the relation between rules. It will choose those multisets where a rule of higher priority cannot be applied anymore even if all rules of a lower priority are not taken:

$$\begin{aligned} Applicable(\Pi, C, Pri_w\delta) = \{R \in Applicable(\Pi, C, \delta) \mid \nexists R' \in Applicable(\Pi, C, \delta), \\ \text{such that } r \in R' \setminus R \text{ and } R'' \notin Applicable(\Pi, C, \delta), \\ \text{where } R'' \supseteq R \cup \{r\} \setminus \{r' \in R \mid r > r'\}\} \end{aligned}$$

3.4 Dissolution

We recall that the dissolution operation (denoted by δ) removes the membrane in which it occurred as well as all rules involving the dissolved membrane. All objects present in that membrane are transferred to its parent. In a general case this operation is handled in FF2 as a special operation acting on the structure like the creation or the division of membranes. However, if we consider the class of P systems where only dissolution is used (no creation/division of membranes) then it is obvious that such systems have a finite number of possible membrane structures (the dissolution operation can only decrease the number of membranes already present at the beginning of the computation). Hence, it is possible to mimic the effect of the dissolution by assigning a marker to each membrane in order to indicate if the membrane is dissolved or not, by using permitting and/or forbidding context in order to check this marker and by using a subset construction at the level of rules in order to capture all possible structure changes.

Example 9. Consider the following transitional P system (see Figure 3) $\Pi = (\{a, b, c, d, \#\}, [_0[_1]_1[_2]_2]_0, \{ac\}, \{c\}, \{a\}, R_0, R_1, R_2)$, where

$$R_0 = \{r_{01} : ac \to \lambda, r_{02} : da \to \#, r_{03} : bc \to \#, r_{04} : \# \to \#\},$$
$$R_1 = \{r_{11} : c \to cc, r_{12} : c \to d\delta\},$$
$$R_2 = \{r_{21} : a \to aa, r_{22} : a \to b\delta\}.$$

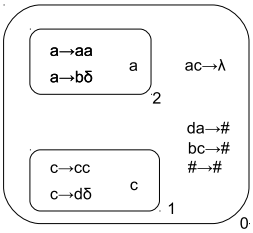

Fig. 3. The P system from Example 9

In order to translate it to the formal framework we shall use 3 objects s_0, s_1 and s_2 indicating that corresponding membranes are not yet dissolved. We place these objects in corresponding cells, although they all can be placed in a particular cell, e.g. cell 0. There are 4 possible membrane structures and they are encoded by the following combinations of objects s_i: $\{(0, s_0)(1, s_1)(2, s_2)\}$, $\{(0, s_0)(1, s_1)\}$, $\{(0, s_0)(2, s_2)\}$ and $\{(0, s_0)\}$. Now in order to finalize the translation we shall do a subset construction for all rules in membrane 0 in order to take into account that corresponding objects can originate from a dissolved membrane:

Rules from R_1 are translated as follows:

$(1, c) \to (1, cc); [(1, s_1)]$ $\qquad$ $(1, s_1)(1, c) \to (0, d)$

Rules from R_2 are translated as follows:

$(2, a) \to (2, aa); [(2, s_2)]$ $\qquad$ $(2, s_2)(2, a) \to (0, b)$

Rules from R_0 are translated as follows:

$(0, ac) \to \lambda; [(2, s_0)]$ $\qquad$ $(0, a)(1, c) \to \lambda; [(0, s_0)]; [(2, s_2)]$

$(0, c)(2, a) \to \lambda; [(0, s_0)]; [(1, s_1)]$ $\qquad$ $(1, c)(2, a) \to \lambda; [(0, s_0)]; [(1, s_1)(2, s_2)]$

$(0, da) \to (0, \#); [(0, s_0)]$ $\qquad$ $(0, d)(2, a) \to (0, \#); [(0, s_0)]; [(2, s_2)]$

$(0, bc) \to (0, \#); [(0, s_0)]$ $\qquad$ $(0, b)(1, c) \to (0, \#); [(0, s_0)]; [(1, s_1)]$

$(0, \#) \to (0, \#); [(0, s_0)]$

We observe that for the translation of rule r_{01} we had to consider 4 cases depending on the possible origin of symbols a and c. The difference between cases is done by corresponding permitting and forbidding conditions.

We also remark that is is possible to avoid forbidding conditions by considering that a state of the membrane i is defined by one of two (dual) objects s_i or $\bar{s}_i$, where the first one indicates that the membrane exists and the second one indicates that the membrane is dissolved. In this case the rule dissolving the membrane will rewrite s_i to $\bar{s}_i$ and the forbidding contexts for s_i are replaced by permitting for $\bar{s}_i$. For example, in the case above the second rule from R_1 becomes $(0, s_1)(1, c) \to (0, d)(1, \bar{s}_1)$ and the second rule from R_0 becomes $(0, a)(1, c) \to \lambda; [(0, s_0)(2, \bar{s}_2)]$.

In some particular cases it is possible to simplify the above construction by assigning a number (hence a special object) to each of possible membrane structures and checking by permitting context the current structure. The drawback of this method is the difficulty to perform several dissolutions in parallel, as several rules should modify the same object at the same step. In some cases (e.g. if the maximally parallel or sequential derivation mode is used) it is possible to overcome this difficulty by using additional rules that perform all required dissolutions in one step.

Example 10. Consider system Π from Example 9. We encode by objects s_i, $1 \leq i \leq 4$, placed in cell 0, the four possible variants of the membrane structure (initial, membrane 2 dissolved, membrane 1 dissolved, membrane 1 and 2 are dissolved). In order to handle the parallel dissolution of membranes 1 and 2 a special rule is introduced.

Rules from R_1 are translated as follows:

$$(1, c) \to (1, cc); [(0, s_1)] \qquad (1, c) \to (1, cc); [(0, s_2)]$$
$$(0, s_1)(1, c) \to (0, ds_3) \qquad (0, s_2)(1, c) \to (0, ds_4)$$

Rules from R_2 are translated as follows:

$$(2, a) \to (2, aa); [(0, s_1)] \qquad (2, a) \to (2, aa); [(0, s_3)]$$
$$(0, s_1)(2, a) \to (0, bs_2) \qquad (0, s_3)(2, a) \to (0, bs_4)$$

Rules from R_0 are translated as follows:

$$(0, ac) \to \lambda \qquad (0, a)(1, c) \to \lambda; [(0, s_3)]$$
$$(0, c)(2, a) \to \lambda; [(0, s_2)] \qquad (1, c)(2, a) \to \lambda; [(0, s_4)]$$

$$(0, da) \to (0, \#) \qquad (0, d)(2, a) \to (0, \#); [(0, s_1)]$$

$$(0, bc) \to (0, \#) \qquad (0, b)(1, c) \to (0, \#); [(0, s_3)]$$

$$(0, \#) \to (0, \#)$$

Additional rule for parallel dissolution:

$$(0, s1)(1, c)(2, a) \to (0, bds_4)$$

We remark that the above construction cannot be generalized to any case. For example, consider a P system evolving in a special derivation mode that requires to use exactly one rule from every membrane at each step. In this case the above construction would fail as the two dissolutions were replaced by a single rule performing both of them.

3.5 Flattening

We call a *flattening* of a P system Π the process of construction of a new P system Π' having only one cell such that $N(\Pi) = N(\Pi')$. We remark that system Π' need not belong to the same class of P systems as Π. A *strong* flattening requires that Π and Π' belong to the same class.

In the case of P systems with static structure (that does not change in time) it can easily be seen that the flattening is very simple, because of the one-to-one relation with multiset rewriting grammars. If the dissolution is present, then it is possible to simulate it as described in previous subsection, hence the flattening procedure will require the use of permitting and eventually forbidding contexts. In the case of systems with dynamically evolving structure (with creation and/or dissolution of membranes) the flattening is not trivial as one should deal with an unbounded number of membranes. A trivial translation yields an unbounded alphabet and an unbounded number of rules, so encodings are necessary to represent the flattening correctly and it is a challenge to provide an algorithm that performs this task.

For the strong flattening it can easily be seen that it is not always possible. For example, any P system that does not allow rules corresponding directly to the multiset rewriting (e.g. symport P systems or minimal symport/antiport P systems) cannot be strongly flattened.

Another example of systems that do not admit strong flattening are systems that have dissolution and do not allow permitting and forbidding contexts, e.g. transitional P systems with dissolution. Because the algorithms eliminating the dissolution require at least a permitting context, it is clear that the flattening cannot be done if remaining inside the same model.

A longer discussion on flattening can be found in [2].

4 Examples of Application of FF

In this section we consider three applications of the formal framework. The first one is the comparison of (purely) catalytic P systems with context-free transitional P systems. The second application consists in extending symport/antiport P systems with variable membrane thickness. The third application studies the model of P colonies and helps in understanding this model and allows to easily obtain some new results.

4.1 Catalytic P Systems

The translation of (purely) catalytic P systems to FF can be done in a quite straightforward manner as follows: every rule $ca \to c(a_1, tar_1) \dots (a_k, tar_k)$ of cell i becomes $(i, ca) \to (F(i, tar_1), a_1) \dots (F(i, tar_k), a_k)$, where $F(i, here) = i$, $F(i, in_j) = j$ and $F(i, out) = m$, where m is the parent of i.

Now we remark that the inherent property of catalytic P systems is that at each step only one rule can be chosen among all rules involving the same catalyst. This property can be deduced from the form of rules of such systems. At the same time it is clear that this property relates more to the way the rules are used together, i.e. to the derivation mode, than to rules' action itself. In FF it is possible to consider a derivation mode that obeys the above restriction. It is not difficult to see that in the case of catalytic (resp. purely catalytic) P systems the above requirements are satisfied by the $msmin_1$ (resp. min_1) derivation mode. The (1-)partitions used for the definition of corresponding modes correspond to each catalyst. In the case of non-purely systems the *-partitions consist of single rules, those that are used in a context-free manner. From the above analysis we can also deduce that catalytic P systems having several membranes and one catalyst are a restricted variant of catalytic P systems with one membrane and using several catalysts.

Hence, we obtain that catalytic P systems evolving in the maximally parallel mode are equal to context-free transitional P systems working in the $msmin_1$ mode, where the 1-partitions correspond to catalysts and *-partitions to each context-free rule. The systems corresponding to purely catalytic P systems have the number of 1-partitions equal to the number of catalysts and no *-partitions, so they evolve in the min_1 mode. It immediately follows that the model of purely catalytic P systems is weaker than the general variant, as context-free rules add more complexity, that can be quantified by the increase of the number of partitions and by the increase of the degree of parallelism.

The usage of the $msmin_1$ mode may look a little bit artificial, so we remark that it is also possible to consider a special mixed mode derived in a straightforward way from the real semantics of catalytic P systems with k catalysts: using $k + 1$ partitions with k partitions working in the min_1 mode (corresponding to catalysts) and one special partition working in the maximally parallel mode.

Example 11. Consider the following catalytic P system
$\Pi = (O, \{c\}, [_0[_1[_3]_3[_4]_4]_1[_2]_2]_0, \{abc\}, \{aac\}, \{c\}, \{a\}, \emptyset, R_0, R_1, R_2, R_3, R_4)$,
where $O = \{a, b, c, d, e\}$, $R_0 = \{cb \to ca_{in_2}\}$, $R_1 = \{ca \to cb_{in} d_{in_4} b_{out} e_{out} e_{here}\}$, $R_3 = \{b \to aa, a \to bc\}$, $R_2 = R_4 = \emptyset$.

The straightforward translation of this system gives the following rules:

$$(0, cb) \to (0, c)(2, a)$$
$$(1, ca) \to (0, be)(1, ce)(4, bd) \qquad (1, ca) \to (0, be)(1, ce)(3, b)(4, d)$$
$$(3, a) \to (3, bc) \qquad (3, b) \to (3, aa)$$

We remark that the translation of the in target is done as in_k, for all k.

Now we construct the context-free network of cells Π' equivalent to Π. Consider following sets of rules:

$$P_1 = \{(1,a) \to (0,be)(1,e)(4,bd), (1,a) \to (0,be)(1,e)(3,b)(4,d)\},$$
$$P_2 = \{(0,b) \to (0,c)(2,a)\},$$
$$P_3 = \{(3,a) \to (3,bc)\},$$
$$P_4 = \{(3,b) \to (3,aa)\}.$$

Let P_1 and P_2 form 1-partitions and let P_3 and P_4 form *-partitions. Then from the discussion above it is clear that Π' working in the $msmin_1$ mode is equivalent to Π. We remark that the obtained network of cells can easily be translated to a context-free transitional P system working in the $msmin_1$ mode.

4.2 Symport/Antiport

In this subsection we discuss how it is possible to use the formal framework in order to extend an existing class of P systems.

Consider the class of symport/antiport P systems. We will extend it with two features: (1) membrane permeability – a symport rule can modify the membrane permeability (with δ or τ). If the membrane is in state 2, then no antiport rule involving this membrane can be used. At each step only one permeability changing rule per membrane can be applied; (2) maximal objects: at each step if there are several maximally parallel evolutions choose the one having the maximal number of objects involved.

In order to define the semantics for both features we will translate symport/antiport P systems to the formal framework, then we will do the necessary transformations in order to accommodate the desired behavior. Then due to the strong bisimulation we can recover the desired semantics in symport/antiport P systems.

The translation of symport and antiport rules can be done in a quite simple way: each antiport rule $(a, in; b, out)$ (resp. symport rule (a, out)) of membrane i is translated as $(i,b)(j,a) \to (i,a)(j,b)$ (resp. $(i,a) \to (j,a)$), where j is the parent membrane of i; each symport rule (a, in) is translated as $(i,a) \to (j,a)$, for all child membranes j of i.

Now in order to implement the first proposed extension, for each membrane i we consider an object in cell i taken from the triple (D, N, C) indicating the state of the membrane (dissolved, normal, closed). Each symport rule having δ or τ will modify this object (going to the left or right in the above sequence) and each antiport rule will check the permitting context if it is in state N. For example: the rule $(ab, out); \tau$ in membrane i will correspond to rules $(i, abN) \to (i, C)(j, ab)$ and $(i, abC) \to (i, C)(j, ab)$.

Now in order to satisfy the second requirement we consider the following derivation mode $max_{obj}max$:

$$Applicable(\Pi, C, max_{obj}max) = \Big\{ R \in Applicable(\Pi, C, max) \mid \nexists R' \in Applicable(\Pi, C, max) : \sum_{r' \in R'} |lhs(r')| > \sum_{r \in R} |lhs(r)| \Big\}$$

It should be clear that the network of cells obtained using the transformations above will be a strong bisimulation of the initial P system.

4.3 P Colonies

We recall the definition of P colonies taken from [9].

A P colony consists of n cells (agents) C_i, $1 \leq i \leq n$, each of them containing a multiset of exactly k symbols and an environment containing initially a distinguished symbol e in an unbounded number of copies. Each cell C_i has a set of programs $\{p_{i,1}, \ldots, p_{i,k_i}\}$, where each program $p_{i,j}$ consists of exactly k rules of the forms $a \to b$ (*internal point mutation*), $c \leftrightarrow d$ (*one object exchange* with the environment), or r_1/r_2 (*priority rule*, where r_1 and r_2 are arbitrary combinations of point mutation and/or exchange rules). The computation can be performed in the maximally parallel or in the sequential mode with respect to the programs of cells. If no more programs are applicable, the system halts and the result is collected as the number of distinguished symbols f in the environment. The number of cells, the maximal number of programs in a cell, and the maximal number of rules in each program in a given P colony Π are called the degree, the height, and the capacity of Π, respectively. The family of sets of numbers computed in the derivation mode x for $x \in \{par, seq\}$ by P colonies of capacity k, degree at most $n \geq 1$ and height at most $h \geq 1$, without (resp. with) using priority rules in their programs, is denoted by $NPCol_x(k, n, h)$ (resp. $NPCol_xK(k, n, h)$).

We will construct a strong bisimulation of the P colony model in the formal framework:

- each rule $a \to b$ in p_{ij} becomes $r_{ij} : (i, a) \to (i, b)$;
- each rule $a \leftrightarrow b$ in p_{ij} becomes $r_{ij} : (i, a)(0, b) \to (i, b)(0, a)$;
- each rule r_1/r_2 in p_{ij} is replaced by two rules: r_1, and $r2; [\emptyset]; [\{(i, a)\}]$ if r_1 is $a \to b$ and $r2; [\emptyset]; [\{(i, a)(0, b)\}]$ if r_1 is $a \leftrightarrow b$.

For the derivation mode each program becomes a rule partition and then the derivation mode requires to be maximal, but using exactly k rules from each partition (or using all rules from a partition). In the sequential case, the derivation mode implies to use only one partition (but all rules from that partition).

Example 12. Consider the following P colony Π having 3 cells.

- C_1 contains the initial multiset aa and the following programs: $p_{11} : a \to b, a \leftrightarrow e$, $p_{12} : a \to c, a \leftrightarrow e$, $p_{13} : b \to a, e \to a$.

- C_2 contains the initial multiset be and the following program: $p_{21} : b \leftrightarrow e, e \rightarrow b$.
- C_3 contains the initial multiset ee and the following programs: $p_{31} : e \leftrightarrow a, e \leftrightarrow b$, $p_{32} : b \rightarrow f, a \rightarrow b$, $p_{33} : f \leftrightarrow a, b \rightarrow b$.

Figure 4 shows a graphical representation of this system.

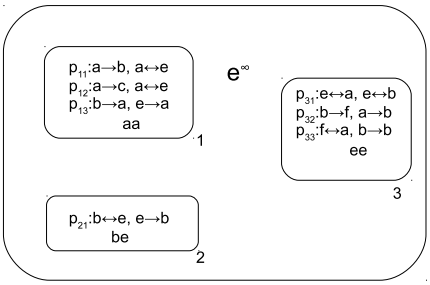

Fig. 4. The P colony from Example 12

We translate this system to a network of cells Π' having 4 cells (numbered from 0 to 3), corresponding to the cells of Π, and having same initial contents as corresponding agents and $Inf_0 = \{e\}$. System Π' contains the following rules:

Rules simulating programs from the first cell:

$$r_{111} : (1, a) \rightarrow (1, b) \qquad r_{112} : (1, a)(0, e) \rightarrow (1, e)(0, a)$$
$$r_{121} : (1, a) \rightarrow (1, c) \qquad r_{122} : (1, a)(0, e) \rightarrow (1, e)(0, a)$$
$$r_{131} : (1, b) \rightarrow (1, a) \qquad r_{132} : (1, e) \rightarrow (1, a)$$

Rules simulating programs from the second cell:

$$r_{211} : (2, b)(0, e) \rightarrow (2, e)(0, b) \qquad r_{212} : (2, e) \rightarrow (2, b)$$

Rules simulating programs from the third cell:

$$r_{311} : (3, e)(0, a) \rightarrow (3, a)(0, e) \qquad r_{312} : (3, e)(0, b) \rightarrow (3, b)(0, e)$$
$$r_{321} : (3, b) \rightarrow (3, f) \qquad r_{322} : (3, a) \rightarrow (3, b)$$
$$r_{331} : (3, f)(0, a) \rightarrow (3, a)(0, f) \qquad r_{332} : (3, b) \rightarrow (3, b)$$

We remark that the derivation mode of P colonies groups rules corresponding to programs, uses maximal parallelism or sequentiality, and requires that all rules from a group shall be used. Since working with one symbol, the group r_{111} and r_{112} from example above is equivalent to the application of a single rule

$r_{11} : (1, aa)(0, e) \to (1, be)(0, a)$. Hence we obtain that a program corresponds to a more complicated rule, and k is the size of the LHS of this rule (and equal to RHS). By considering such rules, the evolution of a P colony becomes just maximally parallel or sequential.

Example 13. Consider the system Π from Example 12. Using the above remark it can be translated to the following network of cells Π'':

$$r_{11} : (1, aa)(0, e) \to (1, be)(0, a) \qquad r_{12} : (1, aa)(0, e) \to (1, ce)(0, a)$$
$$r_{13} : (1, be) \to (1, aa)$$
$$r_{21} : (2, be)(0, e) \to (2, be)(0, b)$$
$$r_{31} : (3, ee)(0, ab) \to (3, ab)(0, ee) \qquad r_{32} : (3, ab) \to (3, fb)$$
$$r_{33} : (3, bf)(0, a) \to (3, ab)(0, f)$$

We can go further by remarking that the number of combinations of objects in an agent is finite, so it can be represented by a single symbol, the state. Symbol e from cell 0 can be ignored as it carries no information. This permits to deduce that a P colony corresponds to a cooperative rewriting with the size of LHS or RHS at most $k + 1$ and forbidding conditions (if checking rules are present). It can be also possible to consider it as a catalytic P system with catalysts having n-states.

Example 14. Consider system Π from Example 12. Consider the array of multisets $A = (aa, be, ce, ee, ab, bf)$ and the following encoding $f(A[i]) = s_i$. Then the rules from Example 13 can be rewritten as follows:

$$r_{11} : (1, s_1) \to (1, s_2)(0, a) \qquad r_{12} : (1, s_1) \to (1, s_3)(0, a) \quad r_{13} : (1, s_2) \to (1, s_1)$$
$$r_{21} : (2, s_2) \to (2, s_2)(0, b)$$
$$r_{31} : (3, s_4)(0, ab) \to (3, s_5) \qquad r_{32} : (3, s_5) \to (3, s_6)$$
$$r_{33} : (3, s_6)(0, a) \to (3, s_5)(0, f)$$

In order to highlight the original semantics (that only one program per cell can be executed), we can use catalysts: the rule $r_{11} : (1, s_1) \to (1, s_2)(0, a)$ becomes $c_1 s_1 \to c_1 s_2 a$. Although using catalysts is not necessary as the state is unique, this permits to consider a restricted variant of the model of P colonies that uses only rules of the above type and therefore corresponds to specific purely catalytic P systems. This remark permits us to transpose results from P colonies to purely catalytic P systems and conversely. For example, from the results for P colonies (Remark after Theorem 23.1.1 and Theorem 23.1.2 from [9]) we immediately obtain that:

Proposition 1. *The following results hold:*

- *Purely catalytic P systems with one catalyst and the size of the rule equal to 3 are not computationally complete.*
- *Purely catalytic P systems of with rules of size 2, an unbounded number of catalysts and using at most 6 rules for each catalyst are computationally complete.*

In the other direction we can also immeadiately obtain (Theorem 4.1 from [9]) that:

Proposition 2. $NPCol_{par}(3,3,*) = RE$.

Our representation of P colonies permits to answer the open question raised in Section 23.1.3 from [9]:

Proposition 3. $NPCol_{seq}(*,1,*) \subsetneq RE$.

This follows from the fact that the corresponding model is identical to purely sequential multiset rewriting which is known to not be computationally complete.

5 Probabilistic P Systems

In this section we extend the formal framework in order to take into account probabilities and thus become able to represent via bisimulation different variants of probabilistic P systems. This section closely follows [11]. To achieve the proposed goal we recall that in order to perform a computational step in a P system a set of multisets of applicable rules, denoted by $Applicable(\Pi, C, \delta)$, is computed according to the type of the system and the derivation mode δ, for any configuration of the system C. After that, one of the elements from this set is chosen, non-deterministically, for the further evolution of the system.

We remark that from the point of view of the computer simulation of P systems the non-deterministic choice can be considered equivalent to a probabilistic choice where each multiset of rules has an equal probability to be chosen. Permitting these multisets to have a *different probability* is the main idea of the extension that we discuss in this section. More precisely, for each multiset of rules $R \in Applicable(\Pi, C, \delta)$ we compute the probability $p(R, C)$ based on the propensity function $f : \mathcal{R}^\circ \times (\mathbb{N} \times O^\circ)^* \to \mathbb{R}$, where $\mathcal{R}$ and O are the set of rules and objects of Π respectively. This function associates a real value for a multiset of rules with respect to a configuration. Hence the value $f(R, C)$ depends not only on the multiset of rules R, but also on the configuration C.

The probability to choose a multiset $R \in Applicable(\Pi, C, \delta)$ is defined as the normalization of corresponding probabilities:

$$p(R, C) = \frac{f(R, C)}{\sum_{R' \in Applicable(\Pi, C, \delta)} f(R', C)} \tag{1}$$

5.1 Discussion

So far we did not indicate the propensity function f, which is the main ingredient of the model. Below we will give examples of simple propensity functions each leading to different execution strategies.

Constant strategy: each rule r from $\mathcal{R}$ has a constant contribution to f and equal to c_r:

$$f(R, C) = \prod_{r \in R} c_r \tag{2}$$

Multiplicity-dependent strategy: each rule r from $\mathcal{R}$ has a contribution to f proportional to the number of times this rule can be applied and to a stochastic constant c_r that only depends on r:

$$N_r(C) = \min_{x \in lhs(r)} \left\lfloor \frac{|C|_x}{|lhs(r)|_x} \right\rfloor \tag{3}$$

$$f(R, C) = \prod_{r \in R} c_r N_r(C) \tag{4}$$

Concentration-dependent strategy: each rule r from $\mathcal{R}$ has a contribution to f proportional to $h_r(C)$, the number of distinct combinations of objects from C that activate r, and to a stochastic constant c_r that only depends on r (by $\binom{a}{b}$ we denote the binomial function):

$$h_r(C) = \prod_{x \in lhs(r)} \binom{|C|_x}{|lhs(r)|_x} \tag{5}$$

$$h_R(C) = \prod_{r \in R} c_r h_r(C) \tag{6}$$

$$f(R, C) = h_R(C) \tag{7}$$

Gillespie strategy: each rule r from $\mathcal{R}$ has a contribution to f that depends on the order in which it was chosen and it is equal to $c_r \cdot h_r(C')$, where C' is the configuration obtained by applying all rules that were chosen before r.

We remark that the concentration-dependent strategy is not equal to Gillespie strategy. More precisely, in a Gillespie run the probability to choose a new rule depends on the objects consumed and produced by previously chosen rules. We can consider a Gillespie run as a sequence of sequential (single-rule) applications using the concentration-dependent strategy.

We also remark that the Gillespie algorithm uses the notion of time that we do not consider in this paper. However, the definitions can easily be adapted for to handle this case.

5.2 Examples

Dynamical Probabilistic P Systems Dynamical probabilistic P (DPP) systems were introduced in [10]. Below, we present the definition of the evolution step. For the sake of the simplicity we will consider only one compartment, however the discussion below can easily be generalized to several compartments.

Let C be the current configuration and $\mathcal{R}$ be the set of all rules. Then the system evolves from C to C' as follows.

1. For each rule $r \in \mathcal{R}$, the propensity of $a_r(C) = c_r * h_r(C)$ (h_r being defined as in Equation (5)) is computed.

2. The propensities are normalized giving a probability for a rule r to be chosen: $p_r(C) = \frac{a_r(c)}{\sum_{r' \in \mathcal{R}} a_{r'}(C)}$.
3. The rules to be applied are chosen according to their probabilities. If a non-applicable rule is chosen, the choice is repeated.
4. The process stops when a maximal (parallel) multiset of rules R is obtained.
5. The multiset of rules obtained at the previous step is applied and yields a new configuration C'.

It can be easily seen that, since the probabilities to apply a rule (p_r) are computed only at the beginning of each step, the maximal multiset of rules R then is composed from independent rules (the order in which the rules were chosen has no influence). Hence the probability to choose a multiset of rules R is equal to the product of the probabilities of each rule: $p_R(C) = \prod_{r \in R} p_r$. Now if we normalize this value with respect to all possible maximally parallel multisets of rules we obtain:

$$\frac{\prod_{r \in R} p_r(C)}{\sum_{R' \in Appl(\Pi, C, max)} \prod_{z \in R'} p_z(C)} = \frac{\prod_{r \in R} \frac{a_r(C)}{\sum_{r' \in \mathcal{R}} a_{r'}(C)}}{\sum_{R' \in Appl(\Pi, C, max)} \prod_{z \in R'} \frac{p_z(C)}{\sum_{r' \in \mathcal{R}} a_{r'}(C)}} = \frac{\prod_{r \in R} a_r(C)}{\sum_{R' \in Appl(\Pi, C, max)} \prod_{z \in R'} a_z(C)} \tag{8}$$

Since in the case of the concentration-dependent strategy we have that $f(R, C) = \prod_{r \in R} a_r(C)$, it follows that (8) equals (1). Hence we just showed that DPP systems can be translated to probabilistic P systems with a concentration-dependent strategy.

Probabilistic Functional Extended P Systems Probabilistic functional extended P (PFEP) systems where introduced in [1] as a part of a framework used to model eco-systems. In order to simplify the presentation we consider a flattening of the structure of the P system, thus using only multiset rewriting rules. We also consider that the rules having the same left-hand side form a partition of the set of rules $\mathcal{R}$ into n subsets $\mathcal{R} = \mathcal{R}_1 \dots \mathcal{R}_n$, where $r_1, r_2 \in \mathcal{R}_i \Rightarrow lhs(r_1) = lhs(r_2)$, $1 \leq i \leq n$.

The evolution of a PFEP system is done as follows:

1. A maximally parallel multiset of rules R is chosen.
2. R is partitioned into submultisets based on the left-hand side of rules: $R_i = \{r \in R \mid r \in \mathcal{R}_i\}$.
3. For each non-empty partition R_i, $|R_i|$ rules from $\mathcal{R}_i$ are chosen according to the given probability $f_r(a)$, where $r \in R_i$ and a is a moment of time.
4. The resulting multiset of rules is applied yielding a new configuration.

From the description of the strategy it is clear that it corresponds to the multiplicity-dependent strategy for the maximally parallel derivation mode (and where the constant c_r is replaced by $f_r(a)$).

6 Active Membranes

In this section we consider the FF2 model described in [3]. The first change with respect to FF1 is the definition of the configuration, which now shall take into account the labels and the membrane structure. Hence, a configuration becomes a couple (L, ρ), where L is a list of "labeled cells" $(i_1, l_1, w_1) \dots (i_n, l_n, w_n)$, with the id $i_j \in \mathbb{N}$, the label $l_j \in Lab$ (the set of labels) and the contents $w_j \in O^*$ (O being the alphabet of the system), for $1 \leq j \leq n$, such that all the cells' id's (the first element of each triple) 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 cells id's).

Note that in a configuration each cell has an id which is unique and a label which is not necessarily unique.

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.

The simulation of most existing variants of active membranes can be done by rules that use following actions:

1. Rewriting of objects (in several cells simultaneously as it is done for the static case).
2. Label change.
3. Creation of a new cell.
4. Creation of a new cell having a contents of some existing cell (and also some additional object rewriting).
5. Deletion of a cell (loosing its contents).
6. Deletion of a cell and moving its contents to some other cell.
7. Arbitrary rewriting of the structure ρ using a graph transducer.

A rule of the network of cells is defined in terms of pattern-matching. First a pattern subgraph structure is given and all actions like rewriting, membrane deletion etc. are given in terms of the pattern structure. During the applicability check the pattern is matched to the actual structure given by the relation ρ. This procedure can yield several matches (instances), so all of them are considered. For each match the preconditions given by the rule (presence or absence of some objects in some cells) are checked and if all of them are satisfied then the rule is applicable. The applicability check is extended to a multiset of rules in a way that is consistent with individual instances of rules. The resulting set of all applicable rules ($Applicable(\Pi, C, asyn)$) is computed as the multisets of couples rule/instance that are applicable to C. Based on this set it is possible to define derivation modes as in the static case. However, additional possibilities related to instances of rules may be investigated, e.g. accepting only particular instances during the derivation (e.g. mandatory including cell number 1).

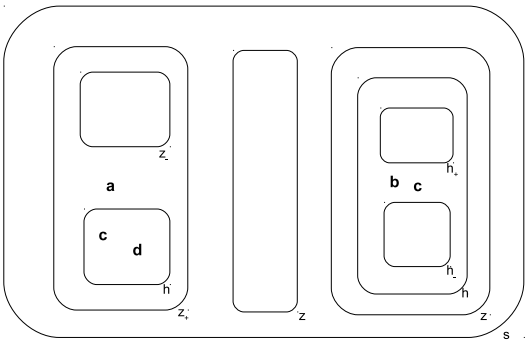

Fig. 5. The P system with active membranes from Example 15. The bold symbols represent the objects.

Example 15. Consider a P system with active membranes having the configuration shown on Figure 5. Let r be the following rule (according to FF2):

$$r : Labels(r) = (z_+, h); \quad \rho(r) = \{(1, 2)\}; \quad Rewrite(r) = (1, a)(2, c) \to (1, c)(2, a).$$

In order to apply r we first should find a combination of two membranes having the labels z_+ and h. There are two such combinations. After that we check the relation $\rho(r)$ which states that membrane 1 (the one identified by the label z_+) is the parent of membrane 2 (the one identified by h). Hence, only the couple at the left remains. We now can identify the numbers 1 and 2 with corresponding cells. Next, the rewriting part of the rule can be applied. It will exchange the symbols a and c, which are located in cells 1 and 2 respectively.

It can easily be seen that the "maximally parallel" derivation mode for active membranes is not really maximally parallel. More precisely it is min_1 for rules involving membranes and max otherwise, so it can be seen as $msmin_1$. This fact makes the active membrane model similar to catalytic P systems, so interesting links can be done. We also remark that even if it is not mentioned explicitly, membrane labels induce cooperation to all rules, thus they have a hidden promoter/permitting context. In the case of the minimally parallel derivation mode for active membranes (min), see Section 11.5 from [9], there could be several interpretations of this concept depending on whether rewriting and membrane rules for a cell are considered to be in the same partition or not.

We would like to emphasize that the current definition of the derivation mode for active membranes allows only one rule per membrane (except rewriting). Using the formal framework it is possible to define in a consistent non-ambiguous way the application of several creation, communication and deletion rules involving the same membrane/cell.

The application $Apply(\Pi, C, R)$ is defined using an algorithm that first applies the rewriting, then creation and then deletion parts of R. At the end, the structure can be modified in an arbitrary way by a graph transducer. We remark that

we used the order rewrite, create, delete (RCD), which is consistent with actual definitions of active membranes. However, it is possible to define the application using other orders like RDC or DCR, yielding slightly different semantics. For example, in RDC order deleted membranes cannot be copied to newly created ones and in CDR order the newly duplicated membranes get the "old" contents, before rewriting. The application of rules in some sense is "global", because the applicability imposes the order of their application to be irrelevant. It is possible to relax this condition and to obtain new application strategies that will differ depending on whether the rules are applied from inside-out or not.

Example 16. Consider a P system with active membranes Π having the configuration shown in Figure 5. Suppose that Π has the following rules:

$$r_1 : a[]_h \to [b]_{h_+} \quad r_2 : [b]_h \to c[]_{h_-} \quad r_3 : [c \to da]_h \quad r_4 : [d]_h \to [e]_{h_+}[f]_h$$

These rules are translated according to FF2 as follows (see [3]). We suppose that h' is an arbitrary membrane label from the set Lab. We also use the following shorthand notation Ls for $Labels$, RW for $Rewrite$, LR for $Label - Rename$, GC for $Generate - and - Copy$ and CR for $Change - Relation$.

$$\begin{array}{llll}
Ls(r_1) = (h', h); & \rho(r_1) = \{(1,2)\}; & RW(r_1) = (1,a) \to (2,b); & LR(r_1) = \{(2,h_+)\}; \\
Ls(r_2) = (h', h); & \rho(r_2) = \{(1,2)\}; & RW(r_2) = (2,b) \to (1,c); & LR(r_2) = \{(2,h_-)\}; \\
Ls(r_3) = (h); & \rho(r_3) = \emptyset; & RW(r_3) = (1,c) \to (1,da); & LR(r_3) = \emptyset; \\
Ls(r_4) = (h', h); & \rho(r_4) = \{(1,2)\}; & RW(r_4) = (2,d) \to (2,e); & LR(r_4) = \{(2,h_+)\}; \\
GC(r_4) = \{(1', h, 2, d \to f)\}; & CR(r_4) = \{INSERT - EDGE(1,1')\}. & &
\end{array}$$

It can easily be seen that to the leftmost membrane labeled by h rules r_1, r_3, and r_4 are applicable, while to the rightmost membrane labeled by h only rules r_2 and r_3 are applicable. The result of the maximally parallel evolution can be seen in Figure 6.

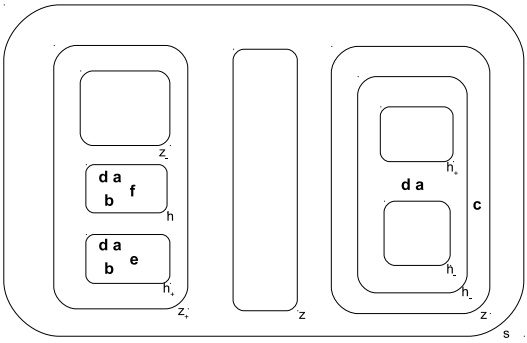

Fig. 6. The result of the application of rules given in Example 16 on the configuration from Figure 5

We remark that this evolution does not correspond to a standard active membranes derivation, because, as it was mentioned above, P systems with active membranes evolve in the $msmin_1$ mode and therefore it is not possible to apply in parallel rules r_1 and r_4 to the leftmost membrane h.

7 Conclusion

In this article we described the model of P systems called the formal framework and we showed how it can be useful when dealing with the following questions: (1) understanding an existing model of P systems; (2) extending a model of P systems with new features or using a different derivation mode; (3) compare two different models of P systems; and (4) explaining details of the semantics that can have several interpretations and raising questions related to these interpretations.

The presented formalism permits to have a powerful language for the description of the features of P systems and is especially useful for making links and transposing results between different models of P systems like it is exemplified in Sections 4.3. Another advantage of the formalism is the ability to treat in a uniform way P systems with static structure, with dynamically evolving structure, and with priorities. This permits to share some basic concepts like derivation modes and may be useful in order to create new formalisms like P systems with active membranes and probabilities.

Acknowledgements. The author would like to thank Rudi Freund for many interesting remarks and suggestions that permitted to improve this paper as well as the support of ANR project SynBioTIC.

References

1. Cardona, M., Colomer, M.A., Margalida, A., Palau, A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Sanuy, D.: A computational modeling for real ecosystems based on P systems. Natural Computing 10(1), 39–53 (2011)
2. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C.: Flattening in (Tissue) P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) CMC 2013, vol. 8340, pp. 173–188. Springer, Heidelberg (2014)
3. Freund, R., Pérez-Hurtado, I., Riscos-Núñez, A., Verlan, S.: A formalization of membrane systems with dynamically evolving structures. International Journal of Computer Mathematics 90(4), 801–815 (2013)
4. Freund, R., Verlan, S.: A formal framework for static (Tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 271–284. Springer, Heidelberg (2007)
5. Freund, R., Verlan, S.: P systems working in the k-restricted minimally parallel mode. In: International Workshop on Computing with Biomolecules, vol. 244, Wien, Austria. Oesterreichische Computer Gesellschaft, pp. 43–52 (2008)

6. Ibarra, O., Yen, H.: Deterministic catalytic systems are not universal. Theor. Comput. Sci. 363(2), 149–161 (2006)
7. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
8. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer (1997)
9. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook Of Membrane Computing. Oxford University Press (2010)
10. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic P systems. International Journal of Foundations of Computer Science 17, 183–204 (2006)
11. Verlan, S.: A note on the probabilistic evolution for P systems. In: Proc. of Tenth Brainstorming Week on Membrane Computing Sevilla, vol. II, pp. 229–234 (2012)
12. Verlan, S.: Study of language-theoretic computational paradigms inspired by biology. Habilitation thesis, University of Paris Est (2010)