Kernel P Systems: From Modelling to Verification and Testing

Marian Gheorghe[a,*], Rodica Ceterchi[b], Florentin Ipate[b], Savas Konur[a], Raluca Lefticaru[a,b]

[a]*School of Electrical Engineering and Computer Science, University of Bradford*
Bradford BD7 1DP, United Kingdom
[b]*Department of Computer Science, Faculty of Mathematics and Computer Science*
University of Bucharest
Str Academiei 14, 010014, Bucharest, Romania

Abstract

A kernel P system integrates in a coherent and elegant manner some of the most successfully used features of the P systems employed in modelling various applications. It also provides a theoretical framework for analysing these applications and a software environment for simulating and verifying them. In this paper, we illustrate the modelling capabilities of kernel P systems by showing how other classes of P systems can be represented with this formalism and providing a number of kernel P system models for a sorting algorithm and a broadcasting problem. We also show how formal verification can be used to validate that the given models work as desired. Finally, a test generation method based on automata is extended to non-deterministic kernel P systems.

1. Introduction

Membrane systems were introduced in [35] as a new natural computing paradigm inspired by the compartmentalised structure of the living cells and the main bio-chemical interactions occurring within compartments and at the inter-cellular level. Later on, they were also called *P systems*. An account of the basic fundamental results can be found in [36]. A comprehensive description of the main research developments in this area is provided in [37] and applications are presented in [15]. The key challenges of the membrane systems area and an overview of some future research directions are available in the survey paper [25].

More recently P systems have been used to model and simulate systems and problems from various areas. However, in order to facilitate the modelling process, in many cases various features and/or constraints have been added in an ad-hoc manner to these classes of P systems. This has led to a multitude of P system variants. The newly introduced concept

*Corresponding author

Email addresses: m.gheorghe@bradford.ac.uk (Marian Gheorghe), rceterchi@gmail.com (Rodica Ceterchi), florentin.ipate@ifsoft.ro (Florentin Ipate), s.konur@bradford.ac.uk (Savas Konur), r.lefticaru@bradford.ac.uk (Raluca Lefticaru)

of *kernel P systems (kP systems, for short)* [19, 20] provides a response to this issue. Indeed, a kP system integrates in a coherent and elegant manner some of the most successfully used features of P systems for modelling various applications and analysing these models.

The kP system model is supported by a modelling language, called kP-Lingua, capable of mapping a kP system specification into a machine readable representation. Furthermore, kP systems are supported by a software framework, kPWorkbench [24], which integrates a set of related simulation and verification methods and tools.

Testing is a major activity in the lifecycle of software systems. In practice, software products are almost always validated through testing. Testing has been discussed for cell-like P systems and various strategies, such as those based on rule coverage and automata techniques, have been proposed [18, 28].

After being introduced [19, 20], kP systems have been investigated from various research angles. Their relationships with other classes of membrane systems have been investigated - firstly, membrane systems with active membrane and neural-like membrane systems have been mapped into kP systems [20, 21], then generalised communicating P systems have been connected with kP systems [30]. Tools, such as kPWorkbench [24], have linked modelling aspects with formal verification and, in particular, model checking [16, 24, 17]. These tools have also included various simulation capabilities, supported by high performance computing techniques and software platforms [4, 31]. Testing methods earlier developed for basic classes of P systems have been extended to kP systems [17]. Various applications have been considered, such as 3-colouring problem [22], sorting algorithms [20, 17], simple broadcasting [24], and synthetic biology paradigms - genetic logic gates [23].

In this paper we continue the investigation of the kP systems class by reconsidering its relationship with other classes of P systems, showing its modelling capabilities by specifying and analysing two problems and providing a testing approach for one of them. The key contributions of the paper can be summarised as follows: (i) an extension of the results presented in [19, 20, 21] regarding the relationships between kP systems and membrane systems with active membranes and electrical charges – these models have been intensively studied, especially with respect to complexity aspects [34, 32, 40]; the connection between kP systems and the class of symport/antiport P systems - another highly investigated model [33, 39]; (ii) new approaches for modelling a sorting algorithm and a broadcasting problem and their formal verification using the model-checking capabilities of the kPWorkbench tool; and finally (iii) an extension of the testing approach developed in [17] for a class of non-deterministic kP systems.

The paper is organised as follows. In Section 2 we introduce the definition of a kernel P system and related concepts. In Section 3 we study the versatility of kP systems, namely, to what extent can we express with kP systems other particular classes of P systems. Section 3.1 investigates the relationship between kP systems and P systems with active membranes, continuing the work in [19, 20, 21]. Section 3.2 is devoted to the relationship between kP systems and P systems with symport/antiport rules. In both cases we show that the latter P system classes can be simulated with suitably designed kP systems. In Section 4 we use kP systems to model two problems. In Section 4.1 we present a sorting algorithm based on a network of comparators, but different from the previous ones [19, 20, 17]. Section 4.2 is devoted to the problem of broadcasting and its modelling with kP systems. The model presented here is different from the model of [24], and is more general. Section 5 is devoted to the verification of kP system models presented in Section 4.1 and Section 4.2. In Section 6 the problem of testing kP systems using automata-based techniques is attacked. The approach is illustrated for the broadcasting problem of Section 4.2. Finally, Section 7 concludes the paper.

2. kP Systems - Main Concepts and Definitions

Standard P system concepts such as strings, multisets, rewriting rules, and computation are well-known and we refer the reader to [36] for their formal notations and precise definitions. The kP system concepts and definitions used in this paper are from [19, 20]. Some of them are slightly changed and this will be mentioned when these concepts are discussed. Some preliminary notations and concepts are introduced in the next section.

2.1. Preliminaries

For a finite alphabet $A = \{a_1, ..., a_p\}$, A^* denotes the set of all strings (sequences) over A. The empty string is denoted by λ and $A^+ = A \setminus \{\lambda\}$ denotes the set of non-empty strings. For a string $u \in A^*$, $|u|_a$ denotes the number of occurrences of a in u, where $a \in A$. For a subset $S \subseteq A$, $|u|_S$ denotes the number of occurrences of the symbols from S in u. The length of a string u is given by $\sum_{a_i \in A} |u|_{a_i}$. The length of the empty string is 0, i.e. $|\lambda| = 0$. A multiset over A is a mapping $f : A \to \mathbb{N}$. Considering only the elements from the support of f (where $f(a_{i_j}) > 0$, for some j, $1 \leq j \leq p$), the multiset is represented as a string $a_{i_1}^{f(a_{i_1})} \dots a_{i_p}^{f(a_{i_p})}$, where the order is not important. In the sequel multisets will be represented by such strings.

2.2. kP System Basic Definitions

We start by introducing the concept of a *compartment type* utilised later in defining the compartments of a kP system.

Definition 1. *T is a finite* set of compartment types, *$T = \{t_1, \dots, t_s\}$, where $t_i = (R_i, \sigma_i)$, $1 \leq i \leq s$, consists of a set of rules, R_i, over an alphabet A, and an execution strategy, σ_i, defined over $Lab(R_i)$, the labels of the rules of R_i.*

The compartments that appear in the definition of the kP systems will be constructed using compartment types introduced by Definition 1. Each compartment, C, will be defined by a tuple (t, w), where $t \in T$ is the type of the compartment and w the initial multiset of it. The types of rules and the execution strategies occurring in the compartment types will be introduced and discussed later.

Definition 2. *A* kP system *of degree n is a tuple*

$$k\Pi = (A, \mu, C_1, \ldots, C_n, i_0),$$

where A is a finite set of elements called objects*; μ defines the* initial membrane structure*, which is a graph, (V, E), where V is a finite set of vertices indicating compartments of the kP system, and E a finite set of edges; $C_i = (t_i, w_i)$, $1 \leq i \leq n$, is a* compartment *of the kP system; i_0 is the label of the* output compartment*, where the result is obtained.*

2.3. kP System Rules

Each rule occurring in a kP system definition has the form $r\ \{g\}$, where r stands for the rule itself and g is its **guard**. The guards are constructed using multisets over A, as operands, and relational or Boolean operators. The definition of the guards is now introduced. We start with somenotations.

Let us denote $Rel = \{<, \leq, =, /=, \geq, >\}$, the set of relational operators, $\gamma \in Rel$, a re- lational operator, and for $a \in A$, a^n a multiset. We first introduce an *abstract relational expression*.

Definition 3. *Let g be the* abstract relational expression *denoted γa^n and w a multiset, then the guard g applied to w denotes the* relational expression $|w|_a \gamma n$.

The abstract relational expression g is true for the multiset w, if $|w|_a \gamma n$ is true.

We consider now the following Boolean operators $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction). An *abstract Boolean expression* is defined by one of the following conditions

- any abstract relational expression is an abstract Boolean expression;
- if g and h are abstract Boolean expressions then $\neg g$, $g \wedge h$ and $g \vee h$ are abstract Boolean expressions.

The concept of a guard, introduced here, is a generalisation of the promotor and inhibitor concepts utilised by some variants of P systems.

Definition 4. *Let g be an* abstract Boolean expression *containing g_i, $1 \leq i \leq q$, abstract relational expressions and w a multiset, then g applied to w means the* Boolean expression *obtained from g by applying g_i to w for any i, $1 \leq i \leq q$.*

As in the case of an abstract relational expression, the guard g is true with respect to the multiset w, if the abstract Boolean expression g applied to w is true.

Example 1. *If g is the guard defined by the abstract Boolean expression $\geq a^5 \wedge < b^3 \vee \neg > c$ and w a multiset, then g applied to w is true if it has at least 5 $a^{\prime}$s and no more than 2 $b^{\prime}$s or no more than one c.*

Definition 5. *A rule from a compartment $C_{l_i} = (t_{l_i}, w_{l_i})$ can have one of the following types:*

- *(a)* **rewriting and communication** *rule: $x \to y\ \{g\}$, where $x \in A^+$ and y has the form $y = (a_1, t_1) \dots (a_h, t_h)$, $h \geq 0$, $a_j \in A$ and t_j, $1 \leq j \leq h$, indicates a compartment type from T (see Definition 2) of compartments linked to the current one; t_j might also indicate the type of the current compartment, $C_{t_{l_i}}$; if a link between C_{l_i} and a compartment of type t_j does not exists (i.e., there is no corresponding edge in E) then the rule is not applied; if a target, t_j, refers to a compartment type that appears in more than one compartment connected to C_{l_i}, then one of them will be non-deterministically chosen;*
- *(b)* **structure changing rules***; the following types of rules are considered:*
  - *(b1)* **membrane division** *rule: $[x]_{t_{l_i}} \to [y_1]_{t_{i_1}} \dots [y_p]_{t_{i_p}}\ \{g\}$, where $x \in A^+$ and $y_j \in A^*$, $1 \leq j \leq p$; the compartment C_{l_i} will be replaced by p compartments; the j-th compartment, $1 \leq j \leq p$, of type t_{i_j} contains the same objects as C_{l_i}, but x, which will be replaced by y_j; all the links of C_{l_i} are inherited by each of the newly created compartments;*
  - *(b2)* **membrane dissolution** *rule: $[]_{t_{l_i}} \to \lambda\ \{g\}$; the compartment C_{l_i} will be destroyed together with its links;*
  - *(b3)* **link creation** *rule: $[x]_{t_{l_i}}; []_{t_{l_j}} \to [y]_{t_{l_i}} - []_{t_{l_j}}\ \{g\}$; the current compartment is linked to a compartment of type t_{l_j} and x is transformed into y; if more than one compartment of type t_{l_j} exist and they are not linked with $C_{t_{l_i}}$, then one of them will be non-deterministically picked up; g is a guard that refers to the compartment of type t_{l_i};*
  - *(b4)* **link destruction** *rule: $[x]_{t_{l_i}} - []_{t_{l_j}} \to [y]_{t_{l_i}}; []_{t_{l_j}}\ \{g\}$; is the opposite of link creation and means that the compartments are disconnected.*

When in a rewriting and communication rule if one of the right hand side elements (a_j, t_j), $1 \leq j \leq h$, is such that $t_j = t_{l_i}$ then this is simply written as a_j.

The membrane division is defined slightly differently here compared to [19, 20], where each y_j, $1 \leq j \leq p$, is composed of objects with target compartments.

2.4. kP System Execution Strategies

In kP systems the way in which rules are executed is defined for each compartment type t from T – see Definition 1. As in Definition 1, $Lab(R)$ is the set of labels of the rules R.

Definition 6. *For a compartment type $t = (R, \sigma)$ from T and $r \in Lab(R)$, $r_1, \dots, r_s \in Lab(R)$, the* execution strategy, *σ, is defined by the following*

- *$\sigma = \lambda$, means no rule from the current compartment will be executed;*

- $\sigma = \{r\}$ – *the rule r is executed;*
- $\sigma = \{r_1, \dots, r_s\}$ – *one of the rules labelled $r_1, \dots, r_s$ will be non-deterministically chosen and executed; if none is applicable then nothing is executed; this is called* alternative *or* choice*;*
- $\sigma = \{r_1, \dots, r_s\}^*$ – *the rules are applied an arbitrary number of times (*arbitrary parallelism*);*
- $\sigma = \{r_1, \dots, r_s\}^T$ – *the rules are executed according to the* maximal parallelism *strategy;*
- $\sigma = \sigma_1 \& \dots \& \sigma_s$*, means executing sequentially $\sigma_1, \dots, \sigma_s$, where σ_i, $1 \leq i \leq s$, describes any of the above cases; if one of σ_i fails to be executed then the rest is no longer executed;*
- *for any of the above σ strategy only one single structure changing rule is allowed.*

A *configuration* of a kP system with n compartments, $C_1, \dots, C_n$, is a tuple $c = (u_1, \dots, u_n)$, where u_i is a multiset of compartment C_i, $1 \leq i \leq n$. Structure changing rules might be applied and the number of compartments will change. A configuration $c' = (v_1, \dots, v_m)$ is obtained in one step from $c = (u_1, \dots, u_n)$, if in each compartment C_i the execution strategy σ_i is applied to u_i, $1 \leq i \leq n$. A *computation*, as usual in membrane computing, is defined as a finite sequence of steps starting from the initial configuration, $(w_1, \dots, w_n)$, and applying in each step and each compartment the rules of the corresponding execution strategy.

Remark 1. *The result of a computation will be the number of objects collected in the output compartment. For a kP system, $k\Pi$, the set of all these numbers will be denoted by $M(k\Pi)$.*

Remark 2. *When a terminal alphabet, $F \subseteq A$, is considered, the result of a computation will be the number of objects from F collected in the output compartment and this will be denoted by $M_t(k\Pi)$.*

3. kP Systems and Other Classes of P Systems

In this section we will investigate the relationship between kP systems and P systems with active membranes and symport/antiport P systems. In [20, 21] the relationships with neural-like P systems have been also considered.

3.1. kP Systems versus P Systems with Active Membranes

The way the relationship between these P systems is presented in the sequel is a natural extension of the method proposed in [19, 20, 21]. In previous investigations the set of objects from the output compartment has been mixed up with the rest of the objects of the system. In this paper we separate the objects corresponding to the output compartment and provide a more consistent notation for the kP system involved. We also deal in this investigation with active membrane systems with an upper bound for the number of active components.

Definition 7. *A P system with active membranes of initial degree n is a tuple (see [37], Chapter 11)* $\Pi = (O, H, \mu, w_{1,0}, \dots, w_{n,0}, R, i_0)$ *where:*

- *O is an alphabet of objects, $w_{1,0}, \dots, w_{n,0}$ are the initial strings in the n initial compartments and i_0 is the output compartment;*
- *H is the set of labels for compartments;*
- *μ defines the tree structure associated with the system;*
- *R consists of rules of the following types:*
  - *(a)* rewriting rules*: $[u \to v]_h^e$, for $h \in H$, $e \in \{+, -, 0\}$ (set of electrical charges), $u \in O^+$, $v \in O^*$;*
  - *(b)* send-in communication rules*: $u[]_h^{e_1} \to [v]_h^{e_2}$, for $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, $u \in O^+$, $v \in O^*$;*
  - *(c)* send-out communication rules*: $[u]_h^{e_1} \to []_h^{e_2} v$, for $h \in H$, $e_1, e_2 \in \{+, -, 0\}$, $u \in O^+$, $v \in O^*$;*
  - *(d)* dissolution rules*: $[u]_h^e \to v$, for $h \in H ¥ \{s\}$, s denotes the skin membrane (the outmost one), $e \in \{+, -, 0\}$, $u \in O^+$, $v \in O^*$;*
  - *(e)* division rules for elementary membranes*: $[u]_h^{e_1} \to [v]_h^{e_2} [w]_h^{e_3}$, for $h \in H$, $e_1, e_2, e_3 \in \{+, -, 0\}$, $u \in O^+$, $v, w \in O^*$.*

The rules are executed in accordance with the maximal parallelism, but in each compartment only one of the rules (b)-(e) is executed. In the sequel we assume that the output compartment is neither dissolved nor divided. The result of a computation, obtained in i_0, is denoted by $M(\Pi)$.

The following result shows how the computation of a P system with active membranes starting with n_1 compartments and having an upper bound for the number of active compartments can be performed by a kP system using only rewriting and communication rules. More precisely, we will show that for every multiset w obtained in the output compartment of a P system with active membranes satisfying the above conditions, Π, there is a kP system, $k\Pi$, such that there is $e \in \{+, -, 0\}$ and we is obtained in the output compartment of $k\Pi$.

Theorem 1. *If Π is a P system with active membranes having n_1 initial compartments and an upper bound for the number of active compartments in any computation, then there exists a kP system, $k\Pi$, of degree 2 and using only rewriting and communication rules, such that $x \in M(k\Pi)$ iff $x + 1 \in M(\Pi)$.*

Proof. Let $\Pi = (O, H, \mu, w_{1,0}, \dots, w_{n_1,0}, R, i_0)$ be a P system with active membranes of initial degree n_1. Initially, the polarisations of the n_1 compartments are all 0, i.e., $e_1 = \cdots = e_{n_1} = 0$.

We will build a kP system with two compartments. Compartment C_1 will capture the contents and rules of all the compartments of Π. The other compartment, C_2, will be associated with i_0 and this will collect the result.

We will need to keep track of the dynamic structure of the system of membranes, since we have dissolution and division of elementary membranes. We will identify a membrane by a pair (i, h) where $i \in I$ is an index associated with a membrane and $h \in H$ is its label. We use the index in addition to the label, as the same label might appear several times in the system, especially after a membrane division rule has been applied. We work under the assumption that I is finite. Its cardinal is equal to the maximum number of active membranes that may appear in any computation – this is assumed to have an upper limit. We let $i_0 \in I$ and $i_0 \in H$. We will denote by $(I \times H)_c$ the currently used pairs (i, h). We assume that for any $(i, h) \in (I \times H)_c$ and $(j, h') \in (I \times H)_c$, we have $i \neq j$. This way we make sure that the cardinal of $(I \times H)_c$ is always at most the cardinal of I, i.e., the upper bound of the total number of compartments appearing in any computation. Whenever a membrane dissolution takes place, its index and label are removed from $(I \times H)_c$. When a membrane division rule is applied, the index and label of the divided compartment are removed from $(I \times H)_c$ and two new values of indices with the same label are selected and added to the set $(I\times H)_c$. The tuple (i_0, i_0) is always in $(I \times H)_c$.

A compartment of Π of label h, electrical charge e and containing the multiset w will be denoted by $[w]^e_h$. We will codify a compartment $[w]^e_h$ by two tuples $< e, i, h >$ and $< w, i, h >$, with $(i, h) \in (I \times H)_c$. For a multiset $w = a_1 \dots a_m$, $< w, i, h >$ denotes $< a_1, i, h > \cdots < a_m, i, h >$. For a compartment $[w]^e_h$, when $h \neq i_0$, the tuples $< e, i, h >$ and $< w, i, h >$ are added to C_1; when $h = i_0$ then in addition to the tuples present in C_1, we add e and w to C_2.

For $(i, h) \in I \times H$ we denote by $p(i, h)$ the parent of the membrane with label h and of index i. If $p(i, h) = (i', h')$ then the membrane with label h' and index i' is the parent of the membrane with label h and index i. By $< x, p(i, h) >$ and $< e, p(i, h) >$ we denote the tuples $< x, i', h' >$ and $< e, i', h' >$, respectively.

Two new symbols, δ_1 and δ_2, will be used for the membrane dissolution and division, respectively, to control the transfer of objects after these rules have been applied. The following guard will be used

$$\overline{=\delta_{all}} := \bigwedge_{(i,h)\in I\times H} (\neg =< \delta_1, i, h >) \wedge (\neg =< \delta_2, i, h >),$$

which is true for a multiset w when none of $< \delta_j, i, h >$, $1 \leq j \leq 2$, $i \in I$ and $h \in H$, appears in w. We also introduce a guard checking that the symbols γ_1 and γ_2, related to the communication with the output compartment, i_0, do not appear in the current multiset:

$$\overline{=\gamma_{all}} := (\neg = \gamma_1) \wedge (\neg = \gamma_2).$$

We construct the kP system, $k\Pi$, using $T = \{t_1, t_2\}$, where $t_j = (R'_j, \sigma_j)$ (with R'_j and σ_j

being defined later), $1 \leq j \leq 2$, as follows:

$$k\Pi = (A, \mu', C_1, C_2, 2),$$

with the elements of the system given below

- μ' is the graph with nodes C_1, C_2 and the edge linking them;
- the alphabet is

$$A = O \cup \{+, +', -, -', 0, 0', \gamma_1, \gamma_2\} \cup \{< b, i, h >| \; b \in \{\delta_1, \delta_2, +, -, 0\}, i \in I, h \in H\}$$

- $C_j = (t_j, w'_{j,0} w''_{j,0})$, $1 \leq j \leq 2$, and C_2 is the output compartment;
  - the initial multiset, $w'_{1,0} w''_{1,0}$, is given by

$$w'_{1,0} = < w_{1,0}, 1, h_1 > \cdots < w_{n_1,0}, n_1, h_{n_1} >,$$

$$w''_{1,0} = < e_1, 1, h_1 > \cdots < e_{n_1}, n_1, h_{n_1} >,$$

  where $e_1 = \cdots = e_{n_1} = 0$, for all the initial multisets and initial membranes of Π. The initial multiset $w'_{2,0} w''_{2,0}$, is given by

$$w'_{2,0} = w_{i_0,0}, w''_{2,0} = e_{i_0}.$$

  Initially, the indices $(I \times H)_1 = \{(1, h_1) \ldots (n_1, h_{n_1})\} \setminus \{(i_0, i_0)\}$ are used in association with compartment C_1 and (i_0, i_0) for C_2. The currently used indices are $(I \times H)_c = (I \times H)_1 \cup \{(i_0, i_0)\}$.
  - R'_1 and R'_2 contain the rules below.

(a.1) For each $(i, h) \in I \times H \setminus \{(i_0, i_0)\}$ and each rule $[u \rightarrow v]^e_h \in R$, $e \in \{+, -, 0\}$, we add to R_1 the rule $< u, i, h > \rightarrow < v, i, h > \{= < e, i, h > \wedge = \delta_{all} \wedge = \gamma_{all}\}$; these rules are applied only when the polarisation e appears in the compartment with index i and label h and none of the $< \delta_k, j, h' >$, $1 \leq k \leq 2$, $j \in I$, $h' \in H$, or γ_1, γ_2 appears, i.e., no dissolution or division has started and no communication with the output compartment, i_0, takes place – see below.

(a.2) For $(i, h) = (i_0, i_0)$, we add to R'_1 the rule $< u, i_0, i_0 > \rightarrow < v, i_0, i_0 > \{= < e, i_0, i_0 > \wedge = \delta_{all} \wedge = \gamma_{all}\}$ and the rule $u \rightarrow v \{= e \wedge = \gamma_{all}\}$ to R'_2.

(b.1) For each $(i, h) \in I \times H \setminus \{(i_0, i_0)\}$ and $p(i, h) \neq (i_0, i_0)$, and each rule $u[\,]^{e_1}_h \rightarrow [v]^{e_2}_h \in R$, $e_1, e_2 \in \{+, -, 0\}$, we add to R'_1 the rule $< u, p(i, h) >< e_1, i, h > \rightarrow < v, i, h >< e_2, i, h > \{= \delta_{all} \wedge = \gamma_{all}\}$; these rules will transform $< u, p(i, h) >$ corresponding to u from the parent compartment to $< v, i, h >$ corresponding to v from the compartment with label h and index i; the polarisation is changed; as there is only one object $< e_1, i, h >$, it follows that only one single rule corresponding to the compartment can be applied at any moment of the computation.

(b.2) When $(i, h) = (i_0, i_0)$, then the rules added to R'_1 are $< u, p(i_0, i_0) >< e_1, i_0, i_0 >\to< v, i_0, i_0 >< e_2, i_0, i_0 > (ve' \gamma_1, 2)\gamma_1 \{= \delta_{all} \wedge = \gamma_{all}\}$ and and $\gamma_1 e \to \lambda$, $e \in \{+, -, 0\}$. The first rule apart from simulating the communication rule, also introduces γ_1 in both compartments. In C_2 it helps changing the polarisation of it and in C_1 it helps with the synchronisation of the computation. Then the symbol disappears.

(b.3) When $p(i, h) = (i_0, i_0)$, then we add to R'_1 the rules $< u, i_0, i_0 >< e_1, i, h >\to<$ $, i, h > (\gamma, 2)\gamma \{$ $= \gamma_{all}\}$ and $\gamma_2 \to \lambda$. The rule $u\gamma_2 \to \lambda$ is added to R'_2. Similar to $(b.2)$, γ_2 is introduced in both compartments and in C_2 it helps removing u.

(c.1) For each $(i, h) \in I \times H ¥ \{(i_0, i_0)\}$ and $p(i, h) /= (i_0, i_0)$, and each rule $[u]^{e_1}_h \to []^{e_2}_h v \in R$, $e_1, e_2 \in \{+, -, 0\}$, we add the rule $< u, i, h >< e_1, i, h >\to< v, p(i, h) >< e_2, i, h > \{= \delta_{all} \wedge = \gamma_{all}\}$.

(c.2) When $(i, h) = (i_0, i_0)$, then we add to R'_1 the rule $< u, i_0, i_0 >< e_1, i_0, i_0 >\to< v, p(i_0, i_0) >< e'_2, i_0, i_0 > (e'_2\gamma_1, 2)\gamma_1 \{= \delta_{all} \wedge = \gamma_{all}\}$. As in $(b.2)$, we use $\gamma_1 \to \lambda$ in R'_1 and $e'_2 \to e_2 \{= \gamma_1\}$ in R'_2. We need to add to R'_2 the rule $u\gamma_1 e \to \lambda$. The rules make sure that in C_1 we simulate the communication rule and in C_2 u disappears and the polarization is changed to e_2.

(c.3) When $p(i, h) = (i_0, i_0)$, then the rule added to R'_1 is $< u, i, h >< e_1, i, h >\to< v, i_0, i_0 >< e_2, i, h > (v, 2) \{= \delta_{all} \wedge = \gamma_{all}\}$. This rule simulates the communication rule and introduces v into C_2.

(d.1) For each $(i, h) \in I \times H ¥ \{(i_0, i_0)\}$ and $p(i, h) /= (i_0, i_0)$, and each rule $[u]^e_h \to v \in R$, $e \in \{+, -, 0\}$, we add to R'_1 the rule $< u, i, h >< e, i, h >\to< v, p(i, h) >< \delta_1, i, h > \{= \delta_{all} \wedge = \gamma_{all}\}$; all the objects corresponding to those from the compartment of index i and label h must be moved to the parent compartment - this will happen in the presence of $< \delta_1, i, h >$ when no other transformation will take place; this is obtained by using in R'_1 rules $< a, i, h >\to< a, p(i, h) > \{=< \delta_1, i, h >\}$, $a \in O$ and $< \delta_1, i, h >\to \lambda$; the set $(I \times H)_c$ will change now by removing the pair (i, h) from it.

(d.2) When $p(i, h) = (i_0, i_0)$, then the rules above will become $< u, i, h >< e, i, h >\to$ $< v, i_0, i_0 >< \delta_1, i, h > (v, 2) \{= \delta_{all} \wedge = \gamma_{all}\}$ and $< a, i, h >\to< a, i_0, i_0 >$ $(a, 2) \{=< \delta_1, i, h >\}$, $a \in O$.

(e) For each $(i, h) \in I \times H ¥ \{(i_0, i_0)\}$ and each rule $[u]^{e_1}_h \to [v]^{e_2}_h [w]^{e_3}_h \in R$, $e_1, e_2, e_3 \in \{+, -, 0\}$; we add to R'_1 the rule $< u, i, h >< e_1, i, h >\to< v, j_1, h >< e_2, j_1, h >< w, j_2, h >< e_3, j_2, h >< \delta_2, i, h > \{= \delta_{all} \wedge = \gamma_{all}\}$ – the pair (i, h) is removed from $(I \times H)_c$ and two new pairs (j_1, h) and (j_2, h), existing in $I \times H$, with $j_1 /= j_2$, are added to $(I \times H)_c$ and one $< u, i, h >$ is transformed into $< v, j_1, h >$ and $< w, j_2, h >$ and their associated electrical charges; then the content corresponding to compartment of index i and label

h will be moved to those of index j_1 and j_2 and the same label h, hence rules $< a, i, h >\rightarrow< a, j_1, h >< a, j_2, h > \{=< \delta_2, i, h >\}$, $a \in O$ are added to R'_1; finally, $< \delta_2, i, h >\rightarrow \lambda$ is also included in the set of rules of C_1; it is clear that only one division rule for the same compartment is applied in any step of the computation.

We note that in C_2 there are no rules for dissolution and division as the output compartment is not affected by these rules.

For each rule in a compartment of label h in Π there is a corresponding rule in $k\Pi$ as defined by R'_1. If the rule is in i_0 then there are rules in both R'_1 and R'_2. Every rewriting rule that is not in i_0 has a corresponding rule defined in accordance with (a.1) in R'_1; every rule in i_0 has a corresponding rule in R'_1 and one in R'_2 according to (a.2). Similarly, for communication rules in Π are defined corresponding rules in R'_1 and R'_2 – rules (b.1) to (b.3) and (c.1) to (c.3). Dissolution rules in Π have their corresponding rules in $k\Pi$ defined by (d.1) and (d.2), whereas each membrane division rule of Π has its corresponding rule in $k\Pi$ defined by (e).

The execution strategy in both compartments, C_1 and C_2 is maximal parallelism.

For a sequence of rules applied in Π, we have a corresponding sequence of rules in $k\Pi$. Obviously, for every multiset w obtained in the output compartment of Π, i_0, there is $e \in \{+, -, 0\}$ and we is obtained in the output compartment of $k\Pi$, C_2.

3.2. P Systems with Symport/Antiport versus kP Systems

The following definition is from [37].

Definition 8. *A P system (of degree $d \geq 1$) with* antiport *and/or* symport *rules is a construct*

$$\Pi = (O, F, E, \mu, w_{1,0}, \cdots, w_{d,0}, R_1, \cdots, R_d, i_0) \text{ where}$$

O is the alphabet of objects; $F \subseteq O$ is the alphabet of terminal objects; $E \subseteq O$ is the set of objects occurring in an unbounded number in the environment; μ is a membrane structure consisting of d membranes (usually labelled with i and represented by corresponding brackets $[_i$ and $]_i$, $1 \leq i \leq d$); w_i, $1 \leq i \leq d$, are strings over O associated with regions $1, \cdots, d$ of μ, representing the initial multisets of objects present in the regions of μ; R_i, $1 \leq i \leq d$, are finite sets of rules of the form $(u, out; v; in)$, with u, v multisets, $u \neq \lambda$ and $v \neq \lambda$ (antiport rule*) and/or (x, out) or (x, in), with x multiset, $x \neq \lambda$ (* symport rules*); i_0, $1 \leq i_0 \leq d$, specifies the output membrane of* Π.

We will show now that one can construct for any symport/antiport P system a kernel P system, such that they compute the same result. We will adopt a slightly different way of computing the result of the kP system by allowing it to use a set of terminal objects. In this case, according to Remark 2, the result will be given by the number of terminal objects from the output compartment. We can now state the main result of this section.

Theorem 2. *For any P system with symport/antiport rules,* Π*, there is a kP system,* $k\Pi$*, using only rewriting and communication rules and having a terminal set of objects, such that* $M(\Pi) = M_t(k\Pi)$.

Proof. Let $\Pi = (O, F, E, \mu, w_{1,0}, \cdots, w_{d,0}, R_1, \cdots, R_d, i_0)$ be a P system, of degree d, with symport and antiport rules as given by Definition 8.

We construct a kP system $k\Pi$ of degree one in the following manner. We take one unique compartment C_1. Apart from the d membranes in system Π, numbered by $1, 2, \cdots, d$, we think of the environment as a new membrane, with label 0.

The kP system we build is $k\Pi = (A, F', \mu', C_1, 1)$. The alphabet, A, of $k\Pi$ will consist of objects given by pairs $< x, i > \in O \times \{0, 1, \cdots, d\}$. For a multiset $w = a_1 \cdots a_m$ in membrane i we use the notation $< w, i >$ for $< a_1, i > \cdots < a_m, i >$.

The initial multiset is

$$w_{1,0} = < w_{1,0}, 1 > \cdots < w_{d,0}, d >,$$

i.e., it contains all the pairs having the first element the initial multiset of membrane i and the second one i, $1 \leq i \leq d$. Initially, the environment associated with Π does not have any other objects apart from those in E. The set of rules, R'_1, of the kP system, includes the rules below.

- If a rule $(u, out; v, in)$, $u \neq \lambda$, $v \neq \lambda$, is in membrane i with parent j and $j \neq 0$, then we add the rule

  $< u, i >< v, j > \rightarrow < u, j >< v, i >$.

- If a rule $(u, out; v, in)$, $u \neq \lambda$, $v \neq \lambda$, is in membrane i with parent $j, j = 0$, then we decompose $u = u_1u_2$ and $v = v_1v_2$, such that $u_1, v_1 \in (O \setminus E)^*$ and $u_2, v_2 \in E^*$ and add the rule

  $< u, i >< v_1, 0 > \rightarrow < u_1, 0 >< v, i >$.

  If $u_1 = \lambda$ or $v_1 = \lambda$ we interpret $< \lambda, 0 >$ as λ, i.e., for $v_1 = \lambda$ and $u_1 \neq \lambda$ the rule becomes $< u, i > \rightarrow < u_1, 0 >< v, i >$.

- If a rule (u, out), $u \neq \lambda$, is in membrane i with parent j and $j \neq 0$, then we add the rule

  $< u, i > \rightarrow < u, j >$.

- If a rule (u, out), $u \neq \lambda$, is in membrane i with parent $j, j = 0$, we add the rule

  $< u, i > \rightarrow < u_1, 0 >$,

  where $u = u_1u_2$ with $u_1 \in (O \setminus E)^*$ and $u_2 \in E^*$.

  If $u_1 = \lambda$, then again $< \lambda, 0 >$ is λ, and the rule becomes $< u, i > \rightarrow \lambda$.

- If a rule (v, in), $v \neq \lambda$, is in membrane i with parent j, and $j \neq 0$, then we add the rule

$< v, j > \rightarrow < v, i >$.

- If a rule (v, in), $v \neq \lambda$, is in membrane i with parent j, $j = 0$, then we add the rule
  $< v_1, 0 > \rightarrow < v, i >$,
  where $v = v_1 v_2$ with $v_1 \in (O \setminus E)^+$ and $v_2 \in E^*$.
  Note that in this last case $v_1 \neq \lambda$.

Note that the environment (membrane 0) is treated differently by the above rules. We do not keep track of elements over E in the environment, which are in an unbounded number, but we must keep track of elements over $O \setminus E$ in the environment. If an u must go into the environment, then we decompose $u = u_1 u_2$ such that $u_1 \in (O \setminus E)^*$ and $u_2 \in E^*$, and only $< u_1, 0 >$ will appear in the right-hand side of the rule. Similarly, if a v comes from the environment, we have $v = v_1 v_2$ with $v_1 \in (O \setminus E)^+$ and $v_2 \in E^*$, and $< v_1, 0 >$ must be consumed by the rule.

The execution strategy of $k\Pi$ will be maximal parallelism.

The terminal alphabet is $F' = \{< a, i_0 > | \ a \in F\}$. Note that multisets over F' obtained in $k\Pi$ will correspond to multisets over F obtained in membrane i_0 by Π.

Remark 3. *It remains an open problem to devise a kP system with two compartments, where C_1 reflects the functioning of the entire system, while C_2 simulates membrane i_0.*

4. Applications

In this section we will present two case studies, a sorting algorithm and a broadcasting problem, illustrating the modelling capabilities of kP systems and the usage of various features of them, including structure changing rules.

4.1. Sorting with kP Systems

Several approaches to sorting have been investigated, for different P system models, based on different algorithms. A first approach was [3], in which a BeadSort algorithm was implemented with tissue P systems. Another approach was [7], in which algorithms inspired from sorting networks were implemented using P systems with communication. Other papers ([1], [38]) use different types of P systems. A dynamic sorting algorithm was proposed in [8]. The bitonic sort was implemented with P systems [9], spiking neural P systems were used for sorting [12], other network algorithms were implemented using P systems [10]. Overviews of sorting algorithms implemented with P systems are [2] and [11].

First implementations of sorting with kP systems were proposed in [19, 20]. Other kP systems for sorting were proposed in [17]. We continue here this modelling approach.

Sorting with n Compartments, based on Insertion

We present here a sorting algorithm implemented with kP systems, which has similarities with the algorithm developed in [19, 20], and also in [17], but still is different. Similar to the second sorting algorithm introduced in [7], which is implemented with P systems with symport/antiport rules and uses priorities and promoters and inhibitors of rules, it is based on an insertion network of comparators.

Like the algorithm of [19, 20], and the first algorithm of [17], it uses an object p to trigger the comparators, whose functioning resembles that of the promoter in the model of [7]. Unlike the previous algorithms whereby objects p are distributed across the system and move back and forth, in the current approach all the objects sit outside the set of components involved in sorting and they travel in one direction. We need $n - 1$ such objects to finish the sorting process.

We represent n positive integers in n separate compartments, each x as the number of occurrences of the same symbol a: a^x.

We construct a kP system, $k\Pi = (A, \mu, C_0, \ldots, C_{n+1})$, having $n + 2$ compartments, $C_i = (t_i, w_{i,0})$, where $t_i = (R_i, \sigma_i)$, $0 \leq i \leq n + 1$, and no output compartment as the results will be collected from the system as a whole, rather than a compartment.

In each compartment, C_i, $1 \leq i \leq n$, the initial multiset, $w_{i,0}$, $1 \leq i \leq n$, includes the representation of the positive integer number x_i, i.e., $w_{i,0} = a^{x_i}$. Compartments C_0 and C_{n+1} are auxiliary compartments, and their role will be made clear in the sequel. The initial contents of C_0 and C_{n+1} are $w_{0,0} = p^{n-1}q$ and $w_{n+1,0} = \lambda$, respectively.

The connections between compartments are given by the set of edges, $E = \{(C_i, C_{i+1}) \mid 0 \leq i \leq n\}$, which together with the vertices, V, (indicating compartments), define the membrane structure, μ, of Π. The alphabet of the system is $A = \{a, b, p, p', q, q'\}$.

The objects p, stored initially in C_0, will travel compartment by compartment towards C_{n+1}, triggering the comparators and stopping in C_{n+1}.

The comparators, triggered by p, act between two adjacent compartments, C_i and C_{i+1}, placing the minimum in C_i and the maximum in C_{i+1}. Next, p travels further, activating other comparators. When the first p arrives in C_{n+1}, we have the maximum value placed in C_n. In the meantime, another p is brought from C_0, two steps behind the first one, and the process repeats. The system works in parallel and each value arrives in its appropriate compartment, achieving the increasing order of the integers.

In C_0 we have the set of rules, R_0, consisting of a single rule

$r_{1,0} : pq \rightarrow (pq, 1)$.

In each compartment C_i, $2 \leq i \leq n$, the set of rules, denoted R_i, consists of

$r_{1,i} : a \rightarrow (b, i + 1)\ \{\geq p\}$;

$r_{2,i} : p \rightarrow p'$;

$r_{3,i} : p' \rightarrow (p, i + 1)$;

$r_{4,i} : ab \rightarrow a(a, i - 1)$;

$r_{5,i} : b \rightarrow a$.

In compartment C_1 we have the rules R_1 given by

$r_{1,1} : a \rightarrow (b, 2)\ \{\geq p\}$;
$r_{2,1} : p \rightarrow p'$;
$r_{3,1} : q \rightarrow q'$;
$r_{4,1} : p' \rightarrow (p, 2)$;
$r_{5,1} : q' \rightarrow (q, 0)$.

In compartment C_{n+1} we have the rules R_{n+1} given by
$r_{1,n+1} : b \rightarrow (a, n)$.

The execution strategies are maximal parallelism for C_0, C_1 and C_{n+1}, i.e., $\sigma_i = R_i^T$, $i = 0, 1, n+1$, and a sequence of two blocks of rules executed in accordance with maximal parallelism, for C_i, $2 \leq i \leq n$; more precisely $\sigma_i = \{r_{j,i} \mid 1 \leq j \leq 4\}^T\{r_{5,i}\}^T$, $2 \leq i \leq n$. This means that for compartments C_i, $2 \leq i \leq n$, the rule $r_{5,i}$ involved in comparators together with $r_{4,i}$ is executed after the later. This way we make sure that first all the pairs ab are processed and finally b's are transformed into a's.

The functioning of the system is as follows: a symbol p is sent from C_0 to C_1 together with a q; whenever p is present in any of the compartments C_i, $1 \leq i \leq n$, the comparisons of two neighbours, C_i and C_{i+1}, start, by using $r_{1,i}$; these rules send all a's as b's to C_{i+1} and simultaneously $r_{2,i}$ will change p into p'; in the next step the comparators finish their jobs by returning the smaller of integers to C_i and keeping the greater ones in C_{i+1} - rules $r_{4,i}$ and $r_{5,i}$ (as it was noted, in accordance with the execution strategy, σ_i, rules $r_{5,i}$ are executed after all the others are executed in C_i); simultaneously, p' will be moved to C_{i+1}, using rule $r_{3,i}$, simulating the journey of the symbols p towards C_{n+1}, moving from one compartment to the next one in two steps. The symbol q will return back to C_0, also in two steps - $r_{3,1}$, $r_{5,1}$ - for bringing another p to C_1. Symbols b from C_{n+1} will be returned back to C_n as a's. The process stops when all $n-1$ p's have travelled to the end and are collected in C_{n+1}.

Theorem 3. *The above algorithm sorts in ascending order a sequence of $n, n \geq 1$, positive integer numbers in $2(2n-1)+1$ steps.*

We illustrate the functioning of the algorithm on a sequence of four integers, 3, 6, 5, 2.

Comp.	Configurations															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C_0	p^3q	p^2	p^2	p^2q	p	p	pq	λ	λ	q	q	q	q	q	q	q
C_1	a^3	a^3pq	$p'q'$	a^3	a^3pq	$p'q'$	a^3	a^3pq	$p'q'$	a^2	a^2	a^2	a^2	a^2	a^2	a^2
C_2	a^6	a^6	a^6b^3	a^6p	p'	a^5b^3	a^5p	p'	a^2b^3	a^3p	p'	a^3	a^3	a^3	a^3	a^3
C_3	a^5	a^5	a^5	a^5	a^5b^6	a^6p	p'	a^2b^5	a^5p	p'	a^5b^3	a^5p	p'	a^5	a^5	a^5
C_4	a^2	a^2	a^2	a^2	a^2	a^2	a^2b^6	a^6p	p'	a^6b^5	a^6p	p'	a^6b^5	a^6p	p'	a^6
C_5	λ	λ	λ	λ	λ	λ	λ	λ	b^6	p	p	b^6p	p^2	p^2	b^6p^2	p^3

We can improve the above algorithm by eliminating superfluous comparisons. We can use a symbol s in the following manner: when the first p reaches C_{n+1}, s starts travelling upwards, and inhibits the comparators. How to implement this modification in the present formalism remains an open problem.

4.2. Broadcasting with kP Systems

In this section we present a variant of the model described in [24] for broadcasting a signal to all the nodes of a tree. The solution presented in [24] uses in each compartment corresponding to a node of the tree rules defined in accordance with the types of the compartments corresponding to the descendant nodes and their number. The solution presented here is much more general and uses one single type for all the components corresponding to nodes from the same level.

The broadcasting problem is formulated as follows: given a tree with n nodes and height (the number of edges on the longest path from the root) equal to m, a signal s is sent to all the nodes of a tree starting from the root. We consider the restrictive case of the broadcasting process whereby at any time the signal is sent from a node to only one of the descendants, non-deterministically chosen. The signal will be returned back, like an acknowledgement, stopping at the root node as f, after all the nodes of the tree have been visited.

In order to model the broadcasting problem we construct a kP system, $k\Pi$, with n compartments. The compartments, labelled $C_{i,j}$, are associated to the nodes of the tree as follows: $C_{1,1}$ corresponds to the root and $C_{i,j}$ corresponds to the node j from level i, $1 \leq i \leq m$, $1 \leq j \leq p_i$, where p_i is the number of nodes of level i. We can now formally define the kP system as follows

$$k\Pi = (A, \mu, C_{1,1}, \dots, C_{m,p_m}, C_{1,1}).$$

A is the set of objects consisting of: s the signal moving through the system, f the final object arriving in the component $C_{1,1}$ corresponding to the root of the tree, d the object assigned initially to the components corresponding to internal nodes of the tree, i.e., having descendants (the multiplicity of d equals the number of descendants of the corresponding node) and v the object appearing in the components that have been visited; μ contains a link between two components when the corresponding nodes of the tree are such that one is the child of the other one. $C_{1,1}$ is the output compartment as it will get the final f. $C_{i,j} = (t_i, w_{i,j,0})$, where t_i defines a type and $w_{i,j,0}$ is the initial multiset of $C_{i,j}$. Please note that all the components corresponding to the nodes from the same level i have the same type t_i, $1 \leq i \leq m$. Each t_i has the form, $t_i = (R_i, \sigma_i)$. The initial multisets are $w_{1,1,0} = sd^{p_2}$, where p_2 denotes the number of descendants of the root (i.e., the number of nodes of level 2); $w_{i,j,0} = \lambda$ if $C_{i,j}$ corresponds to a leaf node and $w_{i,j,0} = d^{p_{i,j}}$ if $C_{i,j}$ is a component corresponding to a node, different from the root, having $p_{i,j}$ descendants.

In the sequel for a component C corresponding to a node, we call descendant components, the components corresponding to the nodes that are descendants of it. Similarly, we will call C parent component with respect to its descendant components. For a component $C_{i,j}$, $1 < i \leq m$, $1 \leq j \leq p_i$, which corresponds to a node which is not a root, the set of rules, R_i, consists of

$r_{1,i} : sd \rightarrow (s, i+1)\ \{ \geq d \wedge < v\}$, – unvisited, with descendants;

$r_{2,i} : s \rightarrow v(s, i-1)\ \{ < d \wedge < v\}$, – unvisited, with no unvisited descendants;

$r_{3,i} : s \rightarrow (sd, i-1)\ \{ < d \wedge = v\}$, – visited, with no descendants.

The execution strategy is $\sigma_i = \{r_{1,i}, r_{2,i}, r_{3,i}\}$ – alternative or choice.

For the component $C_{1,1}$ corresponding to the root the set of rules R_1 contains $r_{1,1}$ derived from R_i for $i = 1$ and
$r_{2,1} : s \to vf \{ < d \wedge < v\}$.
The execution strategy is also alternative or choice: $\sigma_1 = \{r_{1,1}, r_{2,1}\}$.

The broadcasting model works as follows: the signal s goes down from the component corresponding to the root node, anytime there are unvisited descendant components; it will return back level by level after arriving to a component corresponding to a leaf or a node with all descendants being visited. The signal s is sent from the component $C_{1,1}$ to one of its descendant components (when at least a d exists) by using $r_{1,1}$ (this will consume a d and will send s to the descendant component). Anytime s is in a component $C_{i,j}$ corresponding to a node having descendants and with some of them unvisited, then s will be sent to one of the descendant components of $C_{i,j}$ (rule $r_{1,i}$) consuming a d. If the component has been visited and receives an s then both the signal s and a d are returned back to the parent component (rule $r_{3,i}$); otherwise, if it receives s, but is unvisited and with no unvisited descendant components then it marks it visited and returns s to the parent component (rule $r_{2,i}$). Finally, when s returns to $C_{1,1}$ after visiting all the nodes, it will be transformed into f and the process stops.

In the next section we will present using kPWorkbench the formal verification of this problem.

One can observe that nothing stops a signal s from a component to go down to a visited descendant component. A solution to this might be the use of a link destruction rule that will remove the connection between the nodes. This solution is presented in the next Section.

5. Verifying kP Systems

In Section 4, we have illustrated the fact that kP systems provide a coherent and expressive language that allows us to model various systems that were originally implemented by different P system variants. In addition to the modelling aspect, there has been a significant progress in analysing kP systems using various simulation and verification methodologies. The methods and tools developed in this respect have been integrated into a software platform, called kPWorkbench, to support the modelling and analysis of kP systems.

One important feature of kPWorkbench is *formal verification*, which does an exhaustive analysis of system models against some queries to be verified. The automatic verification of kP systems brings in some challenges as they feature a dynamic structure by preserving the structure changing rules such as membrane division, dissolution and link creation/destruction. kPWorkbench employs different verification strategies to alleviate these issues. The framework supports both *Linear Temporal Logic (LTL)* and *Computation Tree Logic (CTL)* properties by making use of the Spin [26] and NuSMV [14] model checkers.

In order to facilitate the formal specification, kPWorkbench features a property language, called *kP-Queries*, comprising a list of natural language statements representing formal property patterns, from which the formal syntax of the Spin and NuSMV formulas are

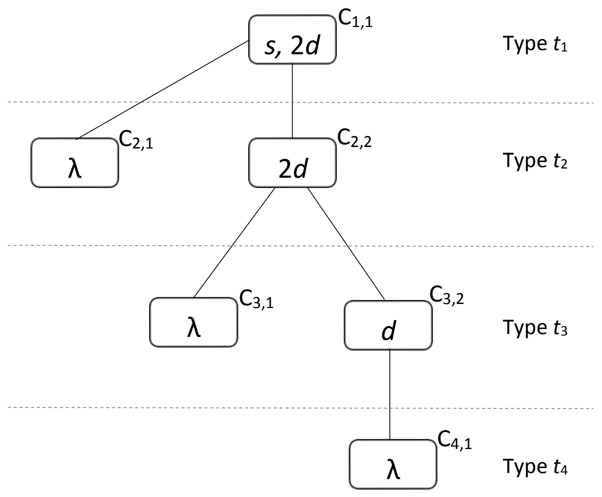

Figure 1: Broadcasting kP system membrane structure

automatically generated. The property language editor interacts with the kP-Lingua model in question and allows users to directly access the native elements in the model, which results in less verbose and shorter state expressions, and hence more comprehensible formulas. *kP-Queries* also features a grammar for the most common property patterns. These features and the natural language like syntax of the language make the property construction much easier. The details can be found in [24].

We now illustrate how verification works by using query patterns on the *broadcasting kernel P system* proposed in the previous section. In order to verify that the algorithm works as desired, we have constructed a set of properties specified as kP-Queries, listed in Table 1. These were prepared for a particular instance of the kP system, having the membrane structure presented in Figure 1 and the kP-Lingua specification in Figure 2. For each property we provide the following information: (i) informal description of each kP-Query, (ii) the formal kP-Query using the patterns, (iii) the LTL specification, and (iv) the CTL specification for NuSMV[1]. The last column shows the verification result of each property, obtained using the NuSMV model checker integrated into kPWorkbench.

We note that there is some difference in the semantics of some LTL and CTL formulas.

[1]When we translate a kP-Lingua model into the corresponding model checking language, we introduce a special predicate, called pInS, to distinguish configurations corresponding to the kP system from intermediate states. pInS is only true when the system is in a kP configuration, i.e., a computation step is completed, and it is false if intermediary steps are executed. When translating a kP-Query into its corresponding LTL and CTL specification, we therefore use pInS in the translated formula to take this into account. We refer the reader to [16] for further details. For the sake of readability, in Table 1 we illustrate properties in the LTL and CTL syntax, rather than showing the exact NuSMV translations which incorporate the pInS predicate.

Prop.	(i) Informal query, (ii) kP-Query, (iii) LTL (NuSMV), (iv) CTL (NuSMV)	Result
1	**(i)** *The process will halt with the root node having* $f = 1$	
	(ii) eventually $(C_{1,1}.f > 0)$	
	(iii) F C11.f > 0	*false*
	(iv) EF C11.f > 0	*true*
2	**(i)** *All the nodes will be visited* $v > 0$ *and the root will have* $f > 0$	
	(ii) eventually $(((C_{1,1}.f > 0$ **and** $C_{1,1}.v > 0)$ **and** $(C_{2,1}.v > 0$ **and** $C_{2,2}.v > 0))$ **and** $((C_{3,1}.v > 0$ **and** $C_{3,2}.v > 0)$ **and** $C_{4,1}.v > 0))$	
	(iii) F (((C11.f > 0 & C11.v > 0) & (C21.v > 0 & C22.v > 0)) & ((C31.v > 0 & C32.v > 0) & C41.v > 0))	*false*
	(iv) EF (((C11.f > 0 & C11.v > 0) & (C21.v > 0 & C22.v > 0)) & ((C31.v > 0 & C32.v > 0) & C41.v > 0))	*true*
3	**(i)** *The node* $C_{2,2}$ *will always receive the broadcast signal* s *before its children nodes (*$C_{3,1}$ *and* $C_{3,2}$*)*	
	(ii) $(C_{2,2}.s=1$ **and** $C_{2,2}.d > 0)$ **followed-by** $(C_{3,1}.s=1$ **or** $C_{3,2}.s=1)$	
	(iii) G ((C22.s = 1 & C22.d > 0) -> F (C31.s = 1 | C32.s = 1))	*true*
	(iv) AG ((C22.s = 1 & C22.d > 0) -> EF (C31.s = 1 | C32.s=1))	*true*
4	**(i)** *If all children nodes are visited* $(v = 1)$ *then parent has no children left to visit* $(d = 0)$	
	(ii) Steady-state $((C_{3,1}.v = 1$ **and** $C_{3,2}.v = 1)$ **implies** $C_{2,2}.d = 0)$	
	(iii) F (G ((C31.v = 1 & C32.v = 1) -> C22.d = 0))	*true*
	(iv) AF (AG ((C31.v = 1 & C32.v = 1) -> C22.d = 0))	*true*
5	**(i)***If* $C_{1,1}$ *and its descendants are visited, then they will remain forever visited*	
	(ii) eventually $((C_{1,1}.v = 1)$ **implies** (**always** $C_{1,1}.v = 1))$	
	(iii) F (C11.v = 1 -> G C11.v = 1)	*true*
	(iv) EF (C11.v = 1 -> AG C11.v = 1)	*true*

Table 1: List of properties for the Broadcasting model derived from the property language and their representations in different formats.

```
type  t1{
   choice{
      >=d  &  <v  :  s, d  ->  s(t2).
      <  d  &  <v  :  s  -> v,  f.
   }
}
type t2 {
   choice{
      >=d  &  <v  :  s, d  ->  s(t3).
      <  d  &  <v  :  s -> v,  s(t1).
      <  d  &  =v  :  s -> s(t1),   d(t1).
   }
}
type t3 {
   choice{
      >=d  &  <v  :  s, d  ->  s(t4).
      < d & <v :    s   ->  v,  s(t2).
      < d & =v :    s   ->  s(t2),  d(t2).
   }
}
type t4 {
   choice{
      <d & <v :   s   ->  v,  s(t3).
      <d  &  =v  : s   ->  s   (t3), d(t3).
   }
}
C11  {s,  2d} (t1).
C21  {} (t2) -  C11.
C22 {2d} (t2) -  C11.
C31  {} (t3) -  C22.
C32  {d} (t3) - C22.
C41  {} (t4) -  C32.
```

Figure 2: kP-Lingua specification for broadcasting

For example, in Prop. 1, the CTL formula EF C11.f > 0 will return true, if there *exists* an execution trace where C11.f > 0 *eventually* holds. However, the LTL formula F C11.f > 0 implicitly enforces to check *every* execution. Hence, it returns false if it finds an execution trace where C11.f > 0 never holds. Since the process might halt in some executions due to non-determinism, we expect different outcomes for LTL and CTL, as confirmed by the verification results. Similarly, we expect the LTL formula in Prop. 2 to be falsified as there could be some execution traces where all nodes might not have been visited. We expect the CTL formula to be satisfied, as there must be at least one execution that satisfies this condition. This has been confirmed by the verification results. In Prop. 3, we have verified the LTL formula, showing that "for all traces the node C_{22} always receives the broadcast signal before its children". This result implies that the property must hold for some traces. Hence, the CTL formula must be true. Our verification results have also confirmed this observation. The verification results for Prop. 5 can be explained similarly. In Prop. 4, the LTL and CTL have the same semantics, so they should return the same result.

As it has been shown through verification, the broadcasting kP model has a solution (EF C11.f > 0 is true), but there exist also infinite computations with loops (AF C11.f > 0 is false) (see Prop. 1-2 from the Table 1). As there could be infinite loops (passing the signal to a child node, then to parent and back again to the same visited child), we propose another model for broadcasting. The kP-Lingua code for this variant is given in Figure 3 and it represents a model which employs *link destruction* when a node and all its descendants have been visited. It can be easily noticed that the computation for this model is still nondeterministic, but avoids visiting the same node twice. Also, it finishes in $2n - 1$ steps, where n is the number of nodes in the tree structure representing the compartments. Please note that the execution strategy of the components different from the root component is maximal parallelism. This is due to the fact that multiple rules have to be executed at certain steps when the link destruction should be applied.

The properties that have been checked for the Sorting algorithm are presented in Table 2. All the properties in the table are intuitive and their meaning explained in the first row of each of them. As confirmed by our verification results, they all have been verified.

Another important feature of kPWorkbench is simulation. Currently, the tool employs two different simulators: kPWorkbench Simulator and Flame (Flexible Large-Scale Agent Modelling Environment). Both simulators receive as input a kP system model written in kP–Lingua and output a trace of the execution, which is mainly used for checking the evolution of a system and for extracting various results out of the simulation. The simulators provide traces of execution for a kP system model, and an interface displaying the current configuration (the content of each compartment) at each step. It is useful for checking the temporal evolution of a kP system and for inferring various information from the simulation results. The reader can find the details in [4].

```
type  t1{
   choice{
      >=d  &  <v  :  s, d  ->  s(t2).
      <  d  &  <v  :  s  -> v,  f.
   }
}
type t2 {
   max{
      >=d  &  <v  :  s, d  ->  s(t3).
      <  d  &  <v  :  s -> v,  s(t1).
      <  d  &  <v  &  =s  :  r -> \- (t1).
      <  d  &  =v  :  s -> s(t1),   d(t1).
   }
}
type t3 {
   max{
      >=d  &  <v  :  s, d  ->  s(t4).
      < d & <v :    s    ->  v,  s(t2).
      <  d  &  <v  &  =s  :  r -> \-  (t2).
      < d & =v :    s    ->  s(t2),  d(t2).
   }
}
type t4 {
   max{
      <d & <v :   s    ->  v,  s(t3).
      <d & <v & =s :  r -> \-     (t3).
      <d  &  =v  : s    ->   s   (t3), d(t3).
   }
}
C11  {s,  2d} (t1).
C21  {r} (t2) -  C11.
C22  {2d,r} (t2) -  C11.
C31  {r} (t3) -  C22.
C32  {d,r} (t3) -  C22.
C41  {r} (t4) -  C32.
```

Figure 3: kP-Lingua specification for broadcasting, using link destruction rules

Prop.	(i) Informal query, (ii) Formal kP-Query using patterns, (iii) CTL (NuSMV)
1	**(i)** *The numbers will be eventually sorted, i.e. the multisets representing the numbers will be in ascending order in the compartments*
	(ii) eventually ((C_1.a <= C_2.a **and** C_2.a <= C_3.a) **and** C_3.a <= C_4.a)
	(iii) EF ((C1.a <= C2.a & C2.a <= C3.a) & C3.a <= C4.a)
2	**(i)** *In the steady-state, the numbers are sorted*
	(ii) steady-state ((C_1.a <= C_2.a **and** C_2.a <= C_3.a) **and** C_3.a <= C_4.a)
	(iii) EF (EG (((C1.a <= C2.a & C2.a <= C3.a) & C3.a <= C4.a)))
3	**(i)** *An unsorted state of two adjacent compartments will always be followed by a sorted one*
	(ii) (C_1.a >C_2.a) **followed-by** (C_1.a <= C_2.a)
	(iii) EG ((C1.a > C2.a) -> EF (C1.a <= C2.a))

Table 2: List of properties for the Sorting model.

6. Testing kP Systems Using Automata Based Techniques

In this section we show how a kP system can be tested using automata based testing methods. We illustrate our approach on the model described in Section 4.2 for broadcasting a signal to the nodes of a tree. The approach presented here follows the blueprint presented in [28] and [18] for cell-like P systems. In [17] it was illustrated on a kP system model of a sorting algorithm, but in that case the model was deterministic. The broadcasting model used here is non-deterministic, which adds generality to the method.

Naturally, in order to apply an automata based testing method to a kP model, a finite automata needs to be obtained first. In general, the computation of a kP system cannot be fully modelled by a finite automaton and so an *approximate* automaton will be sought. The problem will be addressed in two steps.

- Firstly, the computation tree of a P system will be represented as a deterministic finite automaton. In order to guarantee the finiteness of this process, an upper bound *k* on the length of any computation will be set and only computations of maximum *k* transitions will be considered at a time.
- Secondly, a *minimal* model, that preserves the required behaviour, will be defined on the basis of the aforementioned derivation tree.

Let $M_k = (A_k, Q_k, q_{0,k}, F_k, h_k)$ be the finite automaton representation of the computation tree, where A_k is the finite input alphabet, Q_k is the finite set of states, $q_{0,k} \in Q_k$ is the initial state, $F_k \subseteq Q_k$ is the set of final states, and $h_k : Q_k \times A_k \longrightarrow Q_k$ is the next-state function. A_k is composed of the tuples of multisets that label the transitions of the computation tree. The states of Q_k correspond to the nodes of the tree. For testing purposes we will consider all the states as final. It is implicitly assumed that a non-final "sink" state q_{sink} that receives all "rejected" transitions, also exists.

Consider $k\Pi_1$, the kP system in Section 4.2, $n = 5$, the tree having nodes $\{1, 21, 22, 31, 32\}$, 1 the root, and edges $\{(1, 21), (1, 22), (21, 31), (21, 32)\}$. Then $k\Pi_1$ will have 5 compartments, $C_{1,1}$, $C_{2,1}$, $C_{2,2}$, $C_{3,1}$, $C_{3,2}$, and initial multisets $w_{1,1,0} = sd^2$, $w_{2,1,0} = d^2$, $w_{2,2,0} = \lambda$, $w_{3,1,0} = \lambda$, $w_{3,2,0} = \lambda$. The computation of $k\Pi_1$ starts with the application of rule $r_{1,1}$ in compartment $C_{1,1}$. As a consequence, the symbol s will be passed (in a non-deterministic fashion) to one of its successors, $C_{2,1}$ or $C_{2,2}$. If $C_{2,1}$ has received the symbol s, the resulting multiset, sd^2, will trigger rule $r_{1,2}$ and so the symbol s will be passed to either $C_{3,1}$ or $C_{3,2}$, enabling the application of rule $r_{2,3}$ in the destination compartment ($C_{3,1}$ or $C_{3,2}$). If the compartment $C_{2,2}$ has received the symbol s, the resulting multiset, s, will trigger $r_{2,2}$ and so s will be returned to $C_{1,1}$ and the symbol v will mark $C_{2,2}$ as visited; the resulting multiset, sd^2, in compartment $C_{1,1}$ will trigger again rule $r_{1,1}$. For $k = 3$, the deterministic automaton M_k representing the computation tree is given below.[2] In the computation tree, each computation step is labelled by a 5-tuple, each component corresponding to a compartment in the kP system. Since the symbol s, that triggers all rules, is found in one single compartment at a time, only one component is different from λ.

Let us denote
$a_1 = (r_{1,1}, \lambda, \lambda, \lambda, \lambda)$,
$a_2 = (\lambda, r_{1,2}, \lambda, \lambda, \lambda)$,
$a_3 = (\lambda, \lambda, r_{2,2}, \lambda, \lambda)$,
$a_4 = (\lambda, \lambda, \lambda, r_{2,3}, \lambda)$,
$a_5 = (\lambda, \lambda, \lambda, \lambda, r_{2,3})$.

Then, for $k = 3$, $M_k = (A_k, Q_k, q_{0,k}, F_k, h_k)$, where $A_k = \{a_1, a_2, a_3, a_4, a_5\}$, $Q_k = \{q_{0,k}, q_{1,k}, q_{2,k}, q_{3,k}, q_{4,k}, q_{5,k}, q_{6,k}\}$, $F_k = Q_k$, and h_k, the next-state function, is defined by: $h_k(q_{0,k}, a_1) = q_{1,k}$, $h_k(q_{1,k}, a_2) = q_{2,k}$, $h_k(q_{1,k}, a_3) = q_{3,k}$, $h_k(q_{2,k}, a_4) = q_{4,k}$, $h_k(q_{2,k}, a_5) = q_{5,k}$, $h_k(q_{3,k}, a_1) = q_{6,k}$.

As M_k is a deterministic finite automaton over A_k, one can find the minimal deterministic finite automaton that accepts *exactly* the language defined by M_k. However, as only sequences of at most k transitions are considered, it is irrelevant how the constructed automaton will behave for longer sequences. Consequently, a deterministic finite cover automaton of the

[2]For $k = 3$, the computation tree is not sufficiently large to illustrate the functionality of the system (a solution is produced along a path of length at least 9), but this low value is sufficient to illustrate the method and its concepts and avoids unnecessary complexity that would hinder our presentation.

language defined by M_k will be sufficient.

A *deterministic finite cover automaton* (*DFCA*) of a finite language U is a deterministic finite automaton that accepts all sequences in U and possibly other sequences that are longer than any sequence in U [5], [6]. A *minimal DFCA* of U is a DFCA of U having the least possible states. A minimal DFCA may not be unique (up to a renaming of its states). The great advantage of using a minimal DFCA instead of the minimal deterministic automaton that accepts precisely the language U is that the size (number of states) of the minimal DFCA may be much less than that of the minimal deterministic automaton that accepts U. Several algorithms for constructing a minimal DFCA (starting from the deterministic automaton that accepts the language U) exist, the best known algorithm [29] requires $O(n \log n)$ time, where n denotes the number of states of the original automaton. For details about the construction of a minimal DFCA we refer the reader to [28] and [29].

A minimal DFCA of the language defined by M_k in our example, $k = 3$, is $M = (A, Q, q_0, F, h)$, where $A = A_k$, $Q = \{q_0, q_1, q_2, q_3\}$, $F = Q$ and h defined by: $h(q_0, a_1) = q_1$, $h(q_1, a_2) = q_2$, $h(q_1, a_3) = q_3$, $h(q_2, a_4) = q_0$, $h(q_2, a_5) = q_0$, $h(q_3, a_1) = q_0$.

Now, suppose we have a finite state model (automaton) of the system we want to test. In *conformance testing* one constructs a finite set of input sequences, called *test suite*, such that the implementation passes all tests in the test suite if and only if it behaves identically to the specification on any input sequence. Naturally, the implementation under test can also be modelled by an unknown deterministic finite automaton, say M^I. This is not known, but one can make assumptions about it (e.g., that it may have a number of incorrect transitions, missing or extra states). One of the least restrictive assumptions refers to its size (number of states). The *W-method* [13] assumes that the difference between the number of states of the implementation model and that of the specification has to be at most β, a non-negative integer estimated by the tester. The W-method involves the selection of two sets of input sequences, a state cover S and a characterization set W [13].

In our case, we have constructed a DFCA model of the system and we are only interested in the behaviour of the system for sequences of length up to an upper bound k. Then, the test suite will only contain sequences of up to length k and its successful application to the implementation under test will establish that the implementation will behave identically to the specification for any sequence of length less than or equal to k. This situation is called conformance testing for bounded sequences. Recently, it was shown that the underlying idea of the W-method can also be applied in the case of bounded sequences, provided that the sets S and W used in the construction of the test suite satisfy some further requirements; these are called a proper state cover and strong characterisation set, respectively [27]. In what follows we informally define these two concepts and illustrate them on our working example. For formal definitions we refer the reader to [27] or [28].

A *proper state cover* of a deterministic finite automaton $M = (A, Q, q_0, F, h)$ is a set of sequences $S \subseteq A^*$ such that for every state $q \in Q$, S contains a sequence of *minimum length* that reaches q. Consider M the DFCA in our example. Then λ is the sequence of minimum length that reaches q_0, a_1 is a sequence of minimum length that reaches q_1, $a_1 a_2$ is a sequence of minimum length that reaches q_2, a_1a_3 is a sequence of minimum length that reaches q_3. Furthermore, we can use any input symbol in $A ¥ \{a_1\}$ to reach the (implicit) "sink" state, for example a_2. Thus, $S = \{\lambda, a_1, a_1a_2, a_1a_3, a_2\}$ is a proper state cover of M.

A *strong characterization set* of a minimal deterministic finite automaton $M = (A, Q, q_0, F, h)$ is a set of sequences $W \subseteq A^*$ such that for every two distinct states $q_1, q_2 \in Q$, W contains a sequence of minimum length that distinguishes between q_1 and q_2. Consider again our running example. λ distinguishes between the (non-final) "sink" state and all the other (final) states. A transition labelled a_1 is defined from q_0, but not from q_1 or q_2, so a_1 is a sequence of minimum length that distinguishes q_0 from q_1 and q_2. A sequence of minimum length that distinguishes between q_0 and q_3 is a_1a_2 since this is defined from q_0 but not from q_3 and no sequence of length 1 distinguishes between these two states. a_2 is a sequence of minimum length that distinguishes q_1 from q_2 and q_3 (it is defined from q_1 but not from q_2 or q_3) and a_1 is a sequence of minimum length that distinguishes between q_2 and q_3 (it is defined from q_3 but not from q_2). Thus $W = \{\lambda, a_1, a_1a_2, a_2\}$ is a strong characterization set of M.

Once we have established the sets S and W and the maximum number β of extra states that the implementation under test may have, a test suite is constructed by extracting all sequences of length up to k from the set

$$S(A^0 \cup A^1 \cup \cdots \cup A^{\beta})W,$$

where A^i denotes the set of input sequences of length $i \geq 0$.

Note that some test sequences may be accepted by the DFCA model - these are called *positive tests* - but some others may not be accepted (they end up in the (non-final) "sink" state) - these are called *negative tests*.

7. Conclusions

We have investigated in the present paper the relationship between kP systems, on one hand, and other variants of P systems, on the other hand: active membrane systems with polarisation and symport/antiport membrane systems. In both cases we have shown how kP systems can simulate the behaviour of the other P systems.

We have further explored the modelling capabilities of kP systems. We have developed a new sorting algorithm, continuing work in this area with various classes of P systems as well as with kP systems. Also, we have developed a kP system for a special variant of the broadcasting model when, in a step, only one of the descendants of a node receives the broadcasting symbol.

Furthermore, we have shown how formal verification can be used to validate the fact that the given models work as desired.

Finally, we have also proposed a test generation method based on automata, continuing previous work on this topic, but with novel features.

Acknowledgements.

The work of MG, FI and RL were supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI (project number: PN-II-ID-PCE-2011-3-0688). The authors would like to thank the reviewers for their valuable comments and suggestions made.

References

[1] A. Alhazov, D. Sburlan, Static Sorting Algorithms for P Systems, *Pre-Proc. 4th Workshop on Membrane Computing* (A. Alhazov et al., eds.), *GRLMC Rep.* 28/03, Tarragona, 17 – 40, 2003.

[2] A. Alhazov, D. Sburlan, Static Sorting P Systems. In [15], 215 – 252, 2006.

[3] J.J. Arulanandham, Implementing Bead-Sort with P Systems, *Unconventional Models of Computation* (C.S. Calude et al., eds.), *Lecture Notes in Computer Science*, 2509, 115 – 125, 2002.

[4] M. E. Bakir, S. Konur, M. Gheorghe, I. Niculescu, F. Ipate, High Performance Simulations of kernel P Systems, *Proceedings of the 2014 IEEE 16th International Conference on High Performance Computing and Communication*, HPCC'14, 409 – 412, Paris, France, 2014.

[5] C. Cˆampeanu, N. Santean, S. Yu, Minimal Cover-Automata for Finite Languages, *Workshop on Implementing Automata* (J.-M. Champarnaud et al., eds.), *Lecture Notes in Computer Science*, 1660, 43 – 56, 1998.

[6] C. Cˆampeanu, N. Santean, S. Yu, Minimal Cover-Automata for Finite Languages, *Theoretical Computer Science*, 267(1-2), 3 – 16, 2001.

[7] R. Ceterchi, C. Mart´ın-Vide, P Systems with Communication for Static Sorting, *Pre-Proc. 1st Brainstorming Week on Membrane Computing* (M. Cavaliere et al., eds.), *Technical Report* no 26, Rovira i Virgili Univ., Tarragona, 101 – 117, 2003.

[8] R. Ceterchi, C. Mart´ın-Vide, Dynamic P Systems, *Proc. 4th Workshop on Membrane Computing* (Gh. P˘aun et al., eds.), *Lecture Notes in Computer Science*, 2597, 146 – 186, 2003.

[9] R. Ceterchi, M.J. P´erez-Jim´enez, A.I. Tomescu, Simulating the Bitonic Sort Using P Systems, *Proc. 8th Workshop on Membrane Computing* (G. Eleftherakis et al., eds.), *Lecture Notes in Computer Science*, 4860, 172 – 192, 2007.

[10] R. Ceterchi, M.J. P´erez-Jim´enez, A.I. Tomescu, Sorting Omega Networks Simulated with P Systems: Optimal Data Layouts,, *Pre-Proc. 6th Brainstorming Week on Membrane Computing*, (D. Diaz-Pernil et al., eds.), F´enix Editora, Universidad de Sevilla,, 79 – 92, 2008.

[11] R. Ceterchi, D. Sburlan, Membrane Computing and Computer Science, Chapter 22 of [37], 553 – 583, 2010.

[12] R. Ceterchi, A. I. Tomescu, Implementing Sorting Networks with Spiking Neural P Systems, *Fundamenta Informaticae*, 87(1), 35 – 48, 2008.

[13] T. S. Chow, Testing Software Design Modeled by Finite-State Machines, *IEEE Transactions on Software Engineering*, 4(3), 178 – 187, 1978.

[14] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV Version 2: An Open Source Tool for Symbolic Model Checking, *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, (W.A. Hunt et al., eds.), *Lecture Notes in Computer Science*, 2404, 359 – 364, 2002.

[15] G. Ciobanu, Gh. P˘aun, M. J. P´erez-Jim´enez, eds., *Applications of Membrane Computing*, Springer, 2006.

[16] C. Dragomir, F. Ipate, S. Konur, R. Lefticaru, L. Mierla˘, Model Checking kernel P Systems, *Proc. 14th International Conference on Membrane Computing*, (A. Alhazov et al., eds.), *Lecture Notes in Computer Science*, 8340, 151 – 172, 2013.

[17] M. Gheorghe, R. Ceterchi, F. Ipate, S. Konur, Kernel P Systems Modelling, Testing and Verification, *Pre-Proc. 17th International Conference on Membrane Computing, Milan, 25 – 29 July, 2016*, (A. Leporati, C. Zandron, eds.), 161 – 174, 2016.

[18] M. Gheorghe, F. Ipate, On Testing P Systems, *Proc. 9th Workshop on Membrane Computing*, (D.W. Corne et al., eds.), *Lecture Notes in Computer Science*, 5391, 204 – 216, 2009.

[19] M. Gheorghe, F. Ipate, C. Dragomir, Kernel P Systems, *Pre-Proc. 10th Brainstorming Week on Membrane Computing*, (M. A. Mart´ınez-del-Amor et al., eds.), F´enix Editora, Universidad de Sevilla, 153 – 170, 2012.

[20] M. Gheorghe, F. Ipate, C. Dragomir, L. Mierla˘, L. Valencia-Cabrera, M. Garc´ıa-Quismondo, M.J. P´erez-Jim´enez, Kernel P Systems – Version 1, *Pre-Proc. 11th Brainstorming Week on Membrane Computing*, (L. Valencia-Cabrera et al., eds.), F´enix Editora, Universidad de Sevilla, 97 – 124, 2013.

[21] M. Gheorghe, F. Ipate, S. Konur, Kernel P Systems and Relationships with other Classes of P Systems, *Multidisciplinary Creativity*, (M. Gheorghe et al., eds.), Spandugino Publishing House, 64 – 76, 2015.

[22] M. Gheorghe, F. Ipate, R. Lefticaru, M. J. P´erez-Jim´enez, A. T¸urcanu, L. Valencia-Cabrera, M. Garc´ıa-Quismondo, L. Mierla˘, 3-COL Problem Modelling Using Simple kernel P Systems, *International Journal of Computer Mathematics*, 90(4), 816 – 830, 2013.

[23] M. Gheorghe, S. Konur, F. Ipate, Kernel P Systems and Stochastic P Systems for Modelling and Formal Verification of Genetic Logic Gates, in *Advances in Unconventional Computing*, (A. Adamatzky, ed.), Emergence, Complexity and Computation Series, Volume 1: Theory, Springer, 661 – 676, 2015.

[24] M. Gheorghe, S. Konur, F. Ipate, L. Mierla˘, M. E. Bakir, M. Stannett, An Integrated Model Checking Toolset for kernel P Systems, *Proc. 16th Conference on Membrane Computing*, (G. Rozenberg et al., eds.), *Lecture Notes in Computer Science*, 9504, 153 – 170, 2015.

[25] M. Gheorghe, Gh. P˘aun, M.J. P´erez-Jim´enez, G. Rozenberg, Research Frontiers of Membrane Computing: Open Problems and Research Topics, *International Journal of Foundations of Computer Scienve*, 24, 547 – 624, 2013.

[26] G. J. Holzmann, The Model Checker SPIN, *IEEE Transactions on Software Engineering*, 23(5), 275 – 295, 1997.

[27] F. Ipate, Bounded Sequence Testing from Deterministic Finite State Machines, *Theoretical Computer Science*, 411(16-18), 1770 – 1784, 2010.

[28] F. Ipate, M. Gheorghe. Finite State Based Testing of P Systems, *Natural Computing*, 8(4), 833 – 846, 2009.

[29] H. K¨orner, On Minimizing Cover Automata for Finite Languages in O(n log n) Time, *Proc. 7th Conference on Implementation and Application of Automata*, (J.-M. Champarnaud and D. Morel, eds.), *Lecture Notes in Computer Science*, 2608, 117 – 127, 2002.

[30] S. N. Krishna, M. Gheorghe, C. Dragomir, Some Classes of Generalised Communicating P Systems and Simple kernel P Systems, *Proc. 9th International Conference on Computability in Europe, 1 – 5 July, Milan*, (P. Bonizzoni et al., eds.), *Lecture Notes in Computer Science*, 7921, 284 – 293, 2013.

[31] R. Lefticaru, L. F. Mac´ıas-Ramos, I. M. Niculescu, L. Mierla˘, Agent-Based Simulation of kernel P Systems with Division Rules using FLAME, *Pre-Proc. 17th International Conference on Membrane Computing, Milan, 25 – 29 July, 2016*, (A. Leporati, C. Zandron, eds.), 195 – 216, 2016.

[32] L. F. Mac´ıas-Ramos, B. Song, L. Valencia-Cabrera, L. Pan, M. J. P´erez-Jim´enez, Membrane Fission: A Computational Complexity Perspective, *Complexity*, 21(6), 321 – 334, 2016.

[33] L. F. Mac´ıas-Ramos, L. Valencia-Cabrera, B. Song, T. Song, L. Pan, M. J. P´erez-Jim´enez, A P-Lingua Based Simulator for P Syatems with Symport/Antiport Rules, *Fundamenta Informaticae*, 139(2), 211 – 227, 2015,

[34] G. Mauri, A. Leporati, A. E. Porreca, C. Zandron, Recent Complexity-Theoretic Results on P Systems with Active Membranes, Journal of Logic and Computation, 25(4), 1047 – 1071, 2015.

[35] Gh. P˘aun, Computing with Membranes, *Journal of Computer and System Sciences*, 61(1), 108 – 143, 2000.

[36] Gh. P˘aun, Membrane Computing - An Introduction, Springer, 2002.

[37] Gh. P˘aun, G. Rozenberg, A. Salomaa, eds., *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.

[38] D. Sburlan, A Static Sorting Algorithm for P Systems with Mobile Catalysts, *Analele S¸tiin¸tifice Universitatea Ovidius Constan¸ta*, 11(1), 195 – 205, 2003.

[39] B. Song, M. J. P´erez-Jim´enez, L. Pan, Efficient Solutions to Computational Problems by P Systems with Symport/Antiport Rules and Membrane Division, *Biosystems*, 130, 51 – 58, 2015.

[40] L. Valencia-Cabrera, D. Orellana-Mart´ın, M. A. Mart´ınez-del-Amor, A. Riscos-Nu´n˜ez, M. J. P´erez-Jim´enez, Polarizationless P Systems with Active Membranes: Computational Complexity Aspects, *Journal of Automata, Languages and Combinatorics*, 21(1–2), 107 – 123, 2016.