Kernel P Systems - Version 1

Marian Gheorghe[1,2], Florentin Ipate[2], Ciprian Dragomir[1], Laurenţiu Mierlă[2], Luis Valencia-Cabrera[3], Manuel García-Quismondo[3], and Mario J. Pérez-Jiménez[3]

[1] Department of Computer Science
University of Sheffield
Portobello Street, Regent Court, Sheffield, S1 4DP, UK
`{m.gheorghe, c.dragomir}@sheffield.ac.uk`

[2] Department of Computer Science
University of Bucarest
Str Academiei, 14, Bucarest, Romania
`florentin.ipate@ifsoft.ro, laurentiu.mierla@gmail.com`

[3] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012 Seville, Spain
`{lvalencia, mgarciaquismondo, marper}@us.es`

Summary. A basic P system, called kernel P system[4] (kP system for short), combining features of different P systems introduced and studied so far is defined and discussed. The structure of such systems is defined as a dynamic graph, similar to tissue-like P systems, the objects are organised as multisets, and the rules in each compartment, rewriting and communication together with system structure changing rules, are applied in accordance with a specific execution strategy. The definition of kP systems is introduced and some examples illustrate this concept. Two classes of P systems, namely neural-like and generalised communicating P systems are simulated by kP systems. Some case studies prove the expressive power of these systems.

1 Introduction

Different classes of P systems have been introduced and studied for their computational power or for specifying or modelling various problems, like solving simple algorithms [4, 2], NP-complete problems [8] and other applications [5]. More recently various distributed algorithms and problems [13] have been studied with a new variant of P systems. In many cases the specification of the system investigated requires features, constraints or types of behaviour which are not always provided

[4] This concept was introduced initially in [10]; in this paper it is presented a revised version of it.

by the model in its initial definition. It helps in many cases to have some flexibility with modelling approaches, especially in the early stages of modelling, as it might simplify the model, shorten associated processes and clarify more complex or unknown aspects of the system. The downside of this is the lack of a coherent and well-defined framework that allows us to analyse, verify and test this behaviour and simulate the system. In this respect in [10] the concept of *kernel P system* (*kP system*) has ben introduced in order to include the most used concepts from P systems. It is intended to formally define these systems in an operational style and finally implement it within a model checker (SPIN [3], Maude [6]) and integrate it into the P–Lingua platform.

This new class of P systems use a graph-like structure (so called, *tissue P systems*) with a set of symbols, labels of membranes, and rules of various types. A broad range of strategies to run the rules against the multiset of objects available in each compartment is provided. The rules in each compartment will be of two types: (i) *object processing rules* which transform and transport objects between compartments or exchange objects between compartments and environment and (ii) *structure changing rules* responsible for changing the system's topology. Each rule has a guard resembling activators and inhibitors associated with certain variants of P systems. We consider rewriting and communication rules, membrane division, dissolution, bond creation and destruction.

The paper consists of five chapters. Chapter 2 introduces basic definitions, Chapter 3 compares the newly introduce kP systems with some other classes of P systems, Chapter 4 presents a case study based on a static sorting, studying how this problem is solved with various variants of P systems. Finally Chapter 5 briefly describes two specification languages for kP systems with their implementations and some examples.

2 kP Systems

A kP system is a formal model that uses some well-known features of existing P systems and includes some new elements and, more importantly, it offers a coherent view on integrating them into the same formalism. The key elements of a kP system will be formally defined in this section, namely objects, types of rules, internal structure of the system and strategies for running such systems. Some preliminary formal concepts describing the syntax of kP systems and an informal description of the way these systems are executed will be introduced.

We consider that standard concepts like strings, multisets, rewriting rules, and computation are well-known concepts in P systems and indicate [15] as a comprehensive source of information in this respect. First we introduce the key concept of a compartment.

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

Remark 1. The compartments used by the definition of the kP systems will be instantiated from the compartment types defined above. The types of rules and the execution strategy will be discussed later.

Definition 2. *A* kernel P (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* membrane structure*, which is a graph, (V, E), where V are vertices indicating components, and E edges; $C_i = (t_i, w_i)$, $1 \leq i \leq n$, is a* compartment *of the system consisting of a compartment type from T and an* initial multiset*, w_i over A; i_o is the* output compartment *where the result is obtained.*

Remark 2. The inner part of each compartment is called *region*, which is delimited by a *membrane.*

2.1 kP System Rules

The discussion below assumes that the rules introduced belong to the same compartment, C_i.

Each rule r may have a **guard** g, its generic form is r $\{g\}$. The rule r is applicable to a multiset w when its left hand side is contained into w and g is true for w. In the sequel we will analyse how the guards are specified and evaluated.

The guards are constructed using multisets over A and relational and Boolean operators – like Boolean expressions. Before presenting the definition we introduce some notations.

For a multiset w over A and an element $a \in A$, we denote by $\#_a(w)$ the number of a's occurring in w. Let $Rel = \{<, \leq, =, \neq, \geq, >\}$ be the set of relational operators, $\gamma \in Rel$, a relational operator, a^n a multiset and r $\{g\}$ a rule with guard g.

Definition 3. *If g is the* abstract relational expression γa^n *and the current multiset is w, then the guard denotes the* relational expression $\#_a(w)\gamma n$*. The guard g is true for the multiset w if $\#_a(w)\gamma n$ is true.*

Let us consider the Boolean operators $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction), listed wrt decreasing precedence order. Abstract relational expressions can be connected by Boolean operators generating *abstract Boolean expressions.*

Definition 4. *If g is the* abstract Boolean expression *and the current multiset is w, then the guard denotes the* Boolean expression *for w, obtained by replacing abstract relational expressions with relational expressions for w. The guard g is true for the multiset w when the Boolean expression for w is true.*

Definition 5. *A guard is: (i) one of the Boolean constants $true$ or $false$; (ii) an abstract relational expression; or (iii) an abstract Boolean expression.*

Example 1. If the rule is $r : ab \to c \; \{\geq a^5 \wedge \geq b^5 \vee \neg > c\}$, then this can be applied iff the current multiset, w, includes the left hand side of r, i.e., ab and the guard is true for w - it has at least 5 a's and 5 b's or no more than a c.

Definition 6. *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 indicates a compartment type from T – see Definition 2 – with instance compartments linked to the current compartment; t_j might indicate the type of the current compartment, i.e., t_{l_i} – in this case it is ignored; if a link does not exist (the two compartments are not in E) then the rule is not applied; if a target, t_j, refers to a compartment type that has more than one instance connected to l_i, then one of them will be non-deterministically chosen;
- *(b)* **structure changing rules***; the following types 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 has the form $y_j = (a_{j,1}, t_{j,1}) \dots (a_{j,h_j}, t_{j,h_j})$ like in rewriting and communication rules; the compartment l_i will be replaced by p compartments; the j-th compartment, instantiated from the compartment type t_{i_j} contains the same objects as l_i, but x, which will be replaced by y_j; all the links of l_i are inherited by each of the newly created compartments;
  - *(b2)* **membrane dissolution** *rule: $[]_{t_{l_i}} \to \lambda \; \{g\}$;*
    the compartment 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 instance of the compartment type t_{l_j} exists then one of them will be non-deterministically picked up; g is a guard that refers to the compartment instantiated from the compartment type t_{l_1};
  - *(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.

Input-output rules considered in [10] will be expressed as rewriting and communication rules.

2.2 kP System Execution Strategy

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

Definition 7. *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;*
- $\sigma = \{r_1, \ldots, r_s\}$ – *one of the rules labelled $r_1, \ldots, r_s$ will be chosen non-deterministically and executed; if none is applicable then none is executed; this is called* alternative *or* choice*;*
- $\sigma = \{r_1, \ldots, r_s\}^*$ – *the rules are applied an arbitrary number of times (* arbitrary parallelism*);*
- $\sigma = \{r_1, \ldots, r_s\}^\top$ – *the rules are executed according to* maximal parallelism *strategy [15];*
- $\sigma = \sigma_1 \& \ldots \& \sigma_s$*, means executing sequentially $\sigma_1, \ldots, \sigma_s$, where σ_i, $1 \leq i \leq s$, describes any of the above cases, namely λ, one rule, a choice, arbitrary parallelism or maximal parallelism; 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.*

Remark 3. Let us suppose that a certain order relationship exists, e.g., $r_1, r_2 > r_3, r_4$, which means that when weak priority is applied, the first two rules are executed first, if possible, then the next two. If both are executed with maximal parallelism, this is described by $\{r_1, r_2\}^\top \ \{r_3, r_4\}^\top$.

Remark 4. The result of a computation will be the number of objects collected in the output compartment. For a kP systems $k\Pi$, the set of all these numbers will be denoted by $M(k\Pi)$.

2.3 kP System Examples

In this section we illustrate the newly introduced P system model with some examples.

Example 2. Let us consider the set of component types
$T = \{t_1, t_2, t_3\}$, where $t_1 = (R_1, \sigma_1)$, $t_2 = (R_2, \sigma_2)$, $t_3 = (R_3, \sigma_3)$, with
$R_1 = \{r_1 : a \to a(b,2)(c,3)\ \{\geq p\}; r_2 : p \to p; r_3 : p \to \lambda\}$, and $\sigma_1 = Lab(R_1)^\top$,
$R_2 = \{r_1 : b \to (b,0)c\ \{\geq p\}; r_2 : p \to p; r_3 : p \to \lambda\}$, and $\sigma_2 = Lab(R_2)^\top$,
$R_3 = \emptyset$ and $\sigma_3 = Lab(R_3)^\top$.
A kP system with $n = 4$ compartments is $k\Pi_1 = (A, \mu, C_1, \ldots, C_4, 1)$, where
$A = \{a, b, c, p\}$, $C_1 = (t_1, w_{1,0}), C_2 = (t_2, w_{2,0}), C_3 = (t_2, w_{3,0}), C_4 = (t_3, w_{4,0})$; with $w_{1,0} = a^3 p$, $w_{2,0} = w_{3,0} = p$, $w_{4,0} = \lambda$;
μ is given by the graph with nodes $\{C_1, C_2, C_3, C_4\}$ and edges $\{C_1, C_2\}, \{C_1, C_3\}, \{C_1, C_4\}$.

One can note that we do not use targets for objects meant to stay in the current compartment (i.e., we have $r_1 : a \to a(b,2)(c,3)\ \{\geq p\}$ instead of $r_1 : a \to (a,1)(b,2)(c,3)\ \{\geq p\}$). The rule r_1 in R_2 simulates an input/output rule [10] which is meant to bring a c from the environment (0) and to send out a b instead.

In this example there are only rewriting and communication rules; some rules have a guard, $\geq p$ (p is a promoter), others do not have any and in each compartment the rules are applied in maximal parallel way in every step, as indicated by

σ_j, $1 \leq j \leq 3$. As two instances of the compartment type t_2, C_2, C_3, appear in the system, when the rule r_1 from the compartment C_1 is applied, the object b goes non-deterministically to one of the two compartments labelled 2 (from t_2) as long as p remains in compartment C_1; object c goes always to the compartment C_4, of type t_3.

The initial configuration of $k\Pi_1$ is $M_0 = (a^3p, p, p, \lambda)$. The only applicable rules are r_1, r_2 and r_3 from C_1 and r_2, r_3 from C_2, C_3. If r_1, r_2 are chosen in C_1 and r_2 in C_2, C_3, then a^3p is rewritten by r_1, r_2 in C_1 and p in C_2, C_3 by r_2; then three a's stay in C_1, three b's go non-deterministically to C_2, C_3, three c's go to compartment C_4, and each p in C_2, C_3 stays in its compartment. Let us assume that two of them go to C_2 and one to C_3. Hence, the next configuration is $M_1 = (a^3p, b^2p, bp, c^3)$. If in the next step the same rules are applied identically in the first compartment, C_1, and rules r_1, r_2 are used in C_2 and r_1, r_3 in C_3, then the next configuration is $M_2 = (a^3p, b^2c^2p, bc, c^6)$. If now r_1, r_3 are used in C_1, with r_1 used in the same way and r_1, r_3 in C_2 (no rule is available in C_3) then $M_3 = (a^3, b^2c^4, b^2c, c^9)$; this is a final configuration as there is no p to trigger a further step.

Example 3. Let us reconsider the example above enriched with rules dealing with the system's structure. First the set T will be replaced by $T' = \{t_1, t'_2, t_3\}$, where $t'_2 = (R'_2, \sigma'_2)$, with $R'_2 = R_2 \cup R_2^{str}$ and $\sigma'_2 = Lab(R_2)^\top \& Lab(R_2^{str})^\top$. We can notice that σ'_2 tells us that first the rewriting and communication rules are applied in a maximal parallel manner and then one of system's structure rules is chosen to be executed. The set R_2^{str} denotes the set of membrane division rules for t'_2, i.e., $R_2^{str} = \{r_4 : []_2 \to []_2[]_2 \ \{\geq b^2 \wedge \geq p\}\}$. The new kP system, denoted $k\Pi_2$, will have the following four compartments:
$C_1 = (t_1, w_{1,0}), C'_2 = (t'_2, w_{2,0}), C'_3 = (t'_2, w_{3,0}), C_4 = (t_3, w_{4,0})$.

If the system follows the same pathway as $k\Pi_1$ then M_2 shows a different configuration given that in C'_2 after applying R_2 in a maximal parallel manner, R_2^{str} is applied as indicated by σ'_2, when the guard of r_4 is true. The compartment C'_2 is divided into two compartments, $C_{2,1}, C_{2,2}$, instantiated from the same compartment type t_2, with the content of C'_2 and appearing on positions 2 and 3 in the new configuration, $M'_2 = (a^3p, b^2c^2p, b^2c^2p, bc, c^6)$; the new compartments, $C_{2,1}, C_{2,2}$, are linked to compartment C_1. Compartment C'_3 is not divided as the guard of r_4 is not true for its current multiset. In the next step both $C_{2,1}, C_{2,2}$ are divided as they contain the guard triggering the membrane division rule r_4. The process will stop when either p will be rewritten to λ or b^2 stops coming to these compartments.

Remark 5. If we aim to dissolve one of the compartments instantiated from t_2 or to disconnect it from compartment C_1, once a certain condition is true, for instance $\{\geq b^2 \wedge \geq c^2 \wedge \geq p\}$, then two more rules will be added to R_2^{str}, namely
$r_5 : []_2 \to \lambda \ \{\geq b^2 \wedge \geq c^2 \wedge \geq p\}$, $r_6 : []_2 - []_1 \to []_2; []_1 \ \{\geq b^2 \wedge \geq c^2 \wedge \geq p\}$. The expression σ'_2 remains the same, but in this case R_2^{str} contains three elements and at most one is applied at each step, in every compartment with label 2. For this reason σ'_2 can also be written as $Lab(R_2)^\top \& Lab(R_2^{str})$.

2.4 Final Remarks

We will recap and summarise how various rules are applied in a compartment instantiated from a compartment type t_{l_i}.

The rules presented in Section 2.1 are applied as follows:

- **A rewrite - communication** rule, $x \rightarrow (a_1, t_1) \dots (a_h, t_h)\ \{g\}$, is executable if and only if
  1. its left hand side multiset, x, is contained within the current multiset;
  2. its associated guard, g, holds (i.e., it is true for the current multiset);
  3. for each right hand side element, (a_j, t_j), $t_j \in T$ (see Definition 1), there is at least one connected compartment of type t_j to receive a_j, otherwise the rule is not applicable.
- **A membrane division** rule, $[x]_{t_{l_i}} \rightarrow [y_1]_{t_{i_1}} \dots [y_p]_{t_{i_p}}\ \{g\}$, is executable if and only if
  1. the multiset on the left hand side, x, is included in the current multiset;
  2. its associated guard, g, is true for the current multiset;
  3. no other structure changing rule (see Definitions 6 and 7) has been applied in the same step;
  4. the compartment instantiated from t_{l_i} is replaced by p compartments instantiated from the compartment types $t_{i_1}, \dots, t_{i_p}$; their contents will be the same as the content of the compartment on the left hand side, but x will be replaced by $y_1, \dots, y_p$, respectively;
  5. all the links of the compartment instantiated from t_{l_i} will be inherited by each of those instantiated from the compartment types $t_{i_1}, \dots, t_{i_p}$.
- **A membrane dissolution** rule, $[]_{t_{l_i}} \rightarrow \lambda\ \{g\}$, will destroy the compartment obtained from l_i and its links, given the guard g is true for the current multiset; no other structure changing rule (see Definitions 6 and 7) has been applied in the same step;.
- **A link creation** rule, $[x]_{t_{l_i}}; []_{t_{l_j}} \rightarrow [y]_{t_{l_i}} - []_{t_{l_j}}\ \{g\}$, is executable if and only if
  1. its left hand side multiset, x, is contained in the current multiset in the compartment;
  2. its associated guard, g, holds;
  3. there exists at least one membrane instance of type t_{l_j} which is not connected to the instance of type t_{l_i} (i.e there is at least one instance to link to); if there are more instances then one will be non-deterministically chosen;
  4. no other structure changing rule (see Definitions 6 and 7) has been applied in the same step.
- **A link destruction** rule is executed when conditions similar to those mentioned for link creation hold.

3 Neural-like P Systems and P Systems with Active Membranes versus kP Systems

In order to prove how powerful and expressive kP systems are, we will show how two of the most used variants of P systems are simulated by kP systems. More precisely, we will show how neural-like P systems and P systems with active membranes are simulated by some reduced versions of kP systems.

Definition 8. *A* neural-like P system *(tissue P system with states) of degree n is a construct $\Pi = (O, \sigma_1, ...\sigma_n, syn, i_0)$ ([14], p. 249), where:*

- *O is a finite, non-empty set of objects, the* alphabet*;*
- *$\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i)$, $1 \leq i \leq n$, represents a* cell *and*
  - *Q_i is the finite set of* states *of cell σ_i;*
  - *$s_{i,0} \in Q_i$ is the* initial state*;*
  - *$w_{i,0} \in O^*$ is the* initial multiset *of objects contained in cell σ_i;*
  - *R_i is a finite set of* rewriting and communication rules*, of the form $sw \rightarrow s'xy_{go}z_{out}$; when such a rule is applied, x will replace w in cell σ_i, the objects from y will be sent to neighbouring cells, according to the transmission mode (see Remark 6) and the objects from z will be sent out into the environment; cell σ_i will move from state s to s';*
- *$syn \subseteq \{1, ..., n\} \times \{1, ..., n\}$, the connections between cells,* synapses*;*
- *i_0 is the output cell.*

Remark 6. We discuss here a special class of P systems introduced in Definition 8 that will help us to prove a first result.

1. For neural-like P systems, three processing modes are considered, called "max", "min", "par", and three transmission modes, namely "one", "repl", "spread". For formal definitions and other details we refer to [14].
2. We denote by *simple neural-like P systems* the class of P systems given by Definition 8, where the rewriting and communication rules have the form $sw \rightarrow s'x(a_1, t_1) \cdots (a_p, t_p)$, where t_h, $1 \leq h \leq p$, denotes the target cell (σ_h), and processing mode "max", transmission mode defined by the target indications mentioned in each rule.

Notation. For a given P system, Π, the set of numbers computed by Π will be denoted by $M(\Pi)$.

Theorem 1. *If Π is neural-like P system of degree n, then there is a kP system, Π', of degree n and using only rules of type (a), rewriting and communication rules, simulating Π and such that $M(\Pi') \subseteq M(\Pi) \cup \{2\}$.*

Proof. Let Π be a simple neural-like P systems of degree n, as defined by Remark 6.2. We construct the kP system, Π', as follows: the set of compartment types $T = \{t_1, \ldots, t_n\}$, where $t_i = (R'_i, \sigma_i)$, $1 \leq i \leq n$, (see below the definitions of t_i components) and $\Pi' = (A, \mu, C_1, \ldots, C_n, i_0)$ where:

- $A = O \cup (\bigcup_{1 \le i \le n} Q_i) \cup \{\gamma\}$; γ is a new symbol neither in O nor in $\bigcup_{1 \le i \le n} Q_i$;
- $\mu = syn$;
- $C_i = (t_i, w'_{i,0}), 1 \le i \le n$; and
  - $w'_{i,0} = \gamma, 1 \le i \le n$;
  - R'_i contains the following rules:
    1. $\gamma \to s_{i,0} t w_{i,0}$, where $s_{i,0}, w_{i,0}$ are the initial state and initial multiset, respectively, associated with cell σ_i, and $t \in Q^{(i)}_{s_{i,0}}$. For $s \in Q_i$, denote by $Q^{(i)}_s = \{t | t \in Q_i, sx \to ty \in R_i\}$; i.e., $Q^{(i)}_s$ gives, when the cell σ_i is in state s, all the states where σ_i can move to. In the first step, in compartment C_i, a rule $\gamma \to s_{i,0} t w_{i,0}$ is applied and the current multiset becomes $w'_i = s_{i,0} w_{i,0}$.
    2. For each pair $(s, t), t \in Q^{(i)}_s$, there are rules
       $sx_j \to ty_j \in R_i, 1 \le j \le p \quad (*)$.
       If there are no rules in R_i from s to t then another pair of states is considered. For the above rules from R_i, $(*)$, the following rules are considered in R'_i:
       $x_j \to y_j$ $\{= s \wedge = t\}$, $1 \le j \le p$, and $st \to tq$ $\{\ge x_1 \vee \ldots \vee \ge x_p\}$, $q \in Q^{(i)}_t \quad (**)$.
       In the above guards the notation $\ge x_j$, if $x_j = a_{j,1} \ldots a_{j,l_j}$, denotes $\ge a_{j,1} \wedge \ldots \wedge \ge a_{j,l_j}$. The rules $(**)$ make use of guards; the first p rules are applied iff the current multiset contains one s and one t, whereas the last one is applicable iff at least one or more of the occurrences of one of the multisets x_j, $1 \le j \le p$, is included in the current multiset. Clearly, in state s only the rules $(*)$ of Π are applicable for this P system, depending on the availability of the multisets occurring on the left hand side of them; the next state Π is moving to t. Similarly, in Π' only the rules denoted by $(**)$ are applicable; the rule $st \to tq$ $\{\ge x_1 \vee \ldots \vee \ge x_p\}$ is applied once whereas the first p rules are applied as many times as their corresponding $(*)$ rules are applied.

If the set $Q^{(i)}_t$ used in $st \to tq$ $\{\ge x_1 \vee \ldots \vee \ge x_p\}$ of $(**)$ is empty, i.e., there are no rules from state t, then the rule is replaced by $st \to \lambda$. When $Q^{(i)}_{s_{i,0}} = \emptyset$ then the rule $\gamma \to w_{i,0}$ is introduced in R'_i.

At any moment the component C_i of the kP system Π' contains a multiset which is the multiset of σ_i augmented by the current state of σ_i, s, and one of the next states, t, if it exists.

The process will stop in component C_i of Π' when no pair of rules of type $(**)$ is applicable, which means no $sx_i \to ty_i$ rule is applicable in state s.

The multiset $M(\Pi')$ contains $M(\Pi)$ and maybe two states s, t occurring in the last step of the computation. Hence $M(\Pi') \subseteq M(\Pi) \cup \{2\}$. $\square$

Remark 7. A few comments regarding Theorem 1.

1. The above simulation can be assessed with respect to number of compartments, objects and rules as well as the computation steps.

2. When rules $sw \rightarrow txy_{go} \in R_i$ are used in the "spread" mode, this means that any $a \in O$ occurring in y may go to any of the neighbours. In this case if $y = y_1ay_2$ then for each such $a \in O$, in the set R'_i the rule $w \rightarrow xy \ \{= s \wedge = t\}$, defined in the proof of Theorem 1, will be replaced by $w \rightarrow xy_1(a, j)y_2 \ \{= s \wedge = t\}$, where j the label of one of the neighbours of the current compartment. For "one" mode all a's in y will point to the same target, j, for all neighbours of the compartment i.
3. The transmission replicative mode - when a symbol is sent to all the neighbours, can also be simulated. Indeed if $j_1, \dots, j_h$ are the neighbours of i, then $w \rightarrow xy_1(a, j)y_2$ is transformed into $w \rightarrow xy_1(a, j_1) \dots (a, j_h)y_2$ for each a.
4. If a rewriting rule contains z_{out} on its right side, i.e., $sw \rightarrow txy_{go}x_{out}$ then in the set of rules transcribing it, $w \rightarrow \alpha$, we will have $\alpha = xy_1(a, j)y_2z'$, where if $z = a_1 \dots a_k$, then $z' = a'_1 \cdots a'_k$; also rules $a' \rightarrow \lambda$ will be added to R'_i, for any $a \in O$. In this way in the next step all the prime elements are removed from the compartment.
5. If we want to simulate the "min" processing mode then this can be obtained by specifying the sequential behaviour of the component i - by changing the definition of σ_i corresponding to the component.

We study now how P systems with active membranes are simulated by kP systems. In this case we are dealing with a cell-like system, so the underlying structure is a tree and a set of labels (types) for the compartments of the system. The system will start with a number of compartment and its structure will evolve. In the study below it will be assumed that the number of compartments simultaneously present in the system is bounded.

Definition 9. *A P system with active membranes of initial degree n is a tuple (see [15], Chapter 11)* $\Pi = (O, H, \mu, w_{1,0}, \dots, w_{n,0}, R, i_0)$ *where:*

- O*,* $w_{1,0}, \dots, w_{n,0}$ *and* i_0 *are as in Definition 8;*
- 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*:* $[u \rightarrow v]^e_h$*, for* $h \in H$*,* $e \in \{+, -, 0\}$ *(set of electrical charges),* $u \in O^+$*,* $v \in O^*$*;*
  - *(b)* in communication rules*:* $u[]^{e_1}_h \rightarrow [v]^{e_2}_h$*, for* $h \in H$*,* $e_1, e_2 \in \{+, -, 0\}$*,* $u \in O^+$*,* $v \in O^*$*;*
  - *(c)* out communication rules*:* $[u]^{e_1}_h \rightarrow []^{e_2}_h v$*, for* $h \in H$*,* $e_1, e_2 \in \{+, -, 0\}$*,* $u \in O^+$*,* $v \in O^*$*;*
  - *(d)* dissolution rules*:* $[u]^e_h \rightarrow v$*, for* $h \in H \backslash \{s\}$*, s denotes the skin membrane (the outmost one),* $e \in \{+, -, 0\}$*,* $u \in O^+$*,* $v \in O^*$*;*
  - *(e)* division rules for elementary membranes*:* $[u]^{e_1}_h \rightarrow [v]^{e_2}_h [w]^{e_3}_h$*, for* $h \in H$*,* $e_1, e_2, e_3 \in \{+, -, 0\}$*,* $u \in O^+$*,* $v, w \in O^*$*;*

The following result shows how a P system with active membranes starting with n_1 compartments and having no more than n_2 simultaneously present ones can be simulated by a kP system using only rules of type (a).

Theorem 2. *If Π is a P system with active membrane having n_1 initial compartments and utilising no more than n_2 compartments at any time, then there is a kP system, Π', of degree* 1 *and using only rules of type (a), rewriting and communication rules, such that Π' simulates Π.*

Proof. Let us denote $J_0 = \{(i,h) | 1 \leq i \leq n_2, h \in H\}$; for a multiset $w = a_1 \ldots a_m$, (w,i,h), $(i,h) \in J_0$, denotes $(a_1,i,h) \ldots (a_m,i,h)$. Let us consider the P system with active membranes, $\Pi = (O, H, \mu, w_{1,0}, \ldots, w_{n_1,0}, R, i_0)$. The polarizations of the n_1 compartments are all 0, i.e., $e_1 = \ldots = e_{n_1} = 0$.

We construct Π' using $T = \{t_1\}$, where $t_1 = (R'_1, \sigma_1)$ (where R'_1 and σ_1 will be defined later) as follows:
$\Pi' = (A, \mu', C_1, 1)$ where:

- $A = \bigcup_{(i,h) \in J_0} \{(a,i,h) | a \in O \cup \{+,-,0\} \cup \{\delta\}\}$, where δ is a new symbol; let us denote by $= \overline{\delta_{all}}$ the guard $\neg = (\delta, 1, 1) \wedge \ldots \wedge \neg = (\delta, n_2, |H|)$, $|H|$ is the number of elements in H ($= \overline{\delta_{all}}$ stands for none of the (δ,i,h), $(i,h) \in J_0$);
- $\mu' = [\,]_1$;
- $C_1 = (t_1, w'_{1,0})$, and
  - $w'_{1,0} = (w_{1,0}, 1, h_1) \ldots (w_{n_1,0}, n_1, h_{n_1})(e_1, 1, h_1) \ldots (e_{n_1}, n_1, h_{n_1})$, $e_1 = \ldots = e_{n_1} = 0$; let $J_c = J_0 \setminus \{(i,h_i) | 1 \leq i \leq n_1\}$ (J_c denotes indexes available for new compartments and $J_0 \setminus J_c$ the set of indexes of the current compartments);
  - R'_1 contains the following rules
    1. for each $h \in H$ and each rule $[u \to v]_h^e \in R$, $e \in \{+,-,0\}$, we add the rules $(u,i,h) \to (v,i,h)$ $\{= (e,i,h) \wedge = \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$; these rules are applied to every multiset containing elements with $h \in H$, only when the polarization (e,i,h) appears and none of the (δ,j,h') appears;
    2. for each $h \in H$ and each rule $u[\,]_h^{e_1} \to [v]_h^{e_2} \in R$, $e_1, e_2 \in \{+,-,0\}$, we add the rules $(u,j,l)(e_1,i,h) \to (v,i,h)(e_2,i,h)$ $\{= \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$, j is the parent of i of label l; these rules will transform (u,j,l) corresponding to u from the parent compartment j to (v,i,h) corresponding to v from compartment i of label h, the polarization is changed; for each polarization, (e_1,i,h) only one single rule can be applied at any moment of the computation;
    3. for each $h \in H$ and each rule $[u]_h^{e_1} \to [\,]_h^{e_2} v \in R$, $e_1, e_2 \in \{+,-,0\}$, we add the rules $(u,i,h)(e_1,i,h) \to (v,j,l)(e_2,i,h)$ $\{= \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$, j is the parent of i of label l;
    4. for each $h \in H$ and each rule $[u]_h^e \to v \in R$, $e \in \{+,-,0\}$, we add the rules $(u,i,h)(e,i,h) \to (v,j,l)(\delta,i,h)$ $\{= \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$, j is the parent of i of label l; all the elements corresponding to those in compartment i must be moved to j - this will happen in the presence of (δ,i,h) when no other transformation will take place; this is obtained by using rules $(a,i,h) \to (a,j,l)$ $\{= (\delta,i,h)\}$, $a \in O$ and $(\delta,i,h) \to \lambda$ $\{= (\delta,i,h)\}$; the set of available indexes will change now to $J_c = J_c \cup \{(i,h)\}$;

5. for each $h \in H$ and each rule $[u]_h^{e_1} \rightarrow [v]_h^{e_2}[w]_h^{e_3} \in R$, $e_1, e_2, e_3 \in \{+, -, 0\}$; if j_1, j_2 are the indexes of the new compartments, we add $\underline{(u, i, h)}(e_1, i, h) \rightarrow (v, j_1, k_1)(e_2, j_1, k_1)(w, j_2, k_2)(e_3, j_2, k_2)(\delta, i, h)$ $\{= \overline{\delta_{all}}\}$, $1 \leq i \leq n_2$; the content corresponding to compartment i should be moved to j_1 and j_2, hence rules $(a, i, h) \rightarrow (a, j_1, k_1)(a, j_2, k_2$ $\{= (\delta, i, h)\}$, $a \in O$ and finally $(\delta, i, h) \rightarrow \lambda$ $\{= (\delta, i, h)\}$; J_c is updated, $J_c = J_c \cup \{(i, h)\} \setminus \{(j_1, k_1), (j_2, k_2)\}$.

The size of the multiset obtained in i_0 by using Π computation is the same as the size of the multiset in Π, when only (a, i_0, h) are considered, minus 1 (the polarization is also included). □

4 Case Study - Static Sorting

In this section we analyse the newly introduced kP systems by comparing them with established P system classes by using them to specify a static sorting algorithm. This algorithm was first written with symport/antiport rules [4] and then reconsidered in some other cases [2]. The specification below mimics this algorithm.

4.1 Static Sorting with kP Systems

Let us consider a kP system having the following $n = 6$ compartment types:
$t_i = (R_i, \sigma_i)$ and corresponding compartments $C_i = (t_i, w_{i,0})$, $1 \leq i \leq n$, where $w_{1,0} = a^3; w_{2,0} = a^6p; w_{3,0} = a^9; w_{4,0} = a^5p; w_{5,0} = a^7; w_{6,0} = a^8p$.

The rules of the set R_i, $1 \leq i \leq n$, are :
$r_1 : a \rightarrow (b, i-1)$ $\{\geq p\}$, only for $i > 1$
$r_2 : p \rightarrow p'$
$r_3 : p' \rightarrow (p, i-1)$, for i even and $r_3' : p' \rightarrow (p, i+1)$, for i odd
$r_4 : ab \rightarrow a(a, i+1)$, $i < n$
$r_5 : b \rightarrow a$, $i < n$.

We assume that any two compartments, C_i, C_{i+1}, $1 \leq i < n$, are connected. The aim of this problem is to order the content of these compartments such that the highest element (a^9) will be in the leftmost compartment, C_1, and the smallest one (a^3) in the rightmost compartment, C_n, ($n = 6$).

Remark 8. The functioning of the kP systems is presented below:

- $A = \{a, b, p, p'\}$ is the set of objects;
- the rule r_1 is absent from the compartment C_1;
- the last two rules, r_4, r_5, are only present in compartments C_1 to C_{n-1};
- for $n = 2k + 1$ we need an auxiliary compartment, C_{n+1}, which will start with an initial multiset p and will contain a set of rules with $r_2 : p \rightarrow p'$ and $r_3 : p' \rightarrow (p, n)$; whereas C_n should have an additional rule $r_3' : p' \rightarrow (p, n+1)$;

- in each compartment C_i, $\sigma_i = \{r_1, r_2, r_3, r_4\}^\top\{r_5\}^\top$, if i is even; for odd values of i, r_3 is replaced by r_3'; σ_i tells us that firstly the rules from the first set are applied in a maximal parallel manner and then r_5, also in a maximal way;
- σ_i describes an order relationship, $r_1, r_2, r_3, r_4 > r_5$; so we can replace this kP system by a P system with promoters and having an order relationships on the set of rules associated with each membrane.

The table below presents the first steps of the computation. In the first step the only applicable rules are r_1, r_2; given the presence of p, rule r_1 moves all a's from each even compartment to the left compartment as b's and rule r_2 transforms p into p'. Next, rules r_3, r_4, r_5 are applicable; first r_3 and r_4 are applied, this means p' is moved as p to the left compartments and for each ab an a is kept in the current compartment and a b is moved as an a to the right compartment; finally, the remaining b's, if any, are transformed into a's. These two steps implement a sort of comparators between two adjacent compartments moving to the left bigger elements. In the previous steps the comparators have been considered between odd and even compartments. In the next step p's appear in even compartments and the comparators are now acting between an even and an odd compartment. The algorithm does not have a stopping condition. It must stop when no changes appear in two consecutive steps. Given that the algorithm must stop in maximum $2(n-1)$ steps, then we can introduce such a counter, c, in each compartment and rules $c \to c_1$, $c_i \to c_{i+1}$, $1 \le i \le 2(n-1) - 2$ and $c_{2(n-1)-1}p \to \lambda$.

Compartments - Step	C_1	C_2	C_3	C_4	C_5	C_6
0	a^3	a^6p	a^9	a^5p	a^7	a^8p
1	a^3b^6	p'	a^9b^5	p'	a^7b^8	p'
2	a^6p	a^3	a^9p	a^5	a^8p	a^7
3	a^6p'	a^3b^9	p'	a^5b^8	p'	a^7
4	a^6	a^9p	a^3	a^8p	a^5	a^7p
5	a^6b^9	p'	a^3b^8	p'	a^5b^7	p'

Remark 9. **Bounded number of compartment types.** The above solution is using n compartment types for n compartments. As the rules are the same in each compartment, with two exceptions involving the components at both ends of the system (compartments C_1 and C_n), it is natural to look for a solutions with a bounded number of types. If we use the same type everywhere except for the two margins then we face the problem of replacing the rules using targets with different rules where the targets are now the new types; if these are the same we can no longer distinguish between left and right neighbours, so we should have at least two distinct ones. Additionally, we have to distinguish odd and even positions. Consequently, four types, and two more for the two ends are enough. Are there further simplifications? The answer to this question and the solution in this case are left as exercises to the reader.

4.2 Static Sorting with States

We consider the same n-compartment tissue-like P system structure as in the previous subsection. Additionally, in this case, the rules in each compartment use states; an order relationship between rules in each compartment is also considered. Initial states are s_1 in odd compartments and s_0 otherwise; the content of the 6 regions is illustrated by the first line, step 0, of the table below.

The addition of states is potentially very useful from a modelling point of view since many widely-used modelling languages are state-based and, therefore, such rules were a strong candidate for inclusion in our kP system model. However, as shown below, states can be effectively simulated by rewriting rules, as shown below.

For the algorithm considered, the rules in each compartment and the order relationships are as follows
Compartment 1:
$r_1 : s_0x \rightarrow s_0y$
$r_2 : s_0y \rightarrow s_1x$
$r_3 : s_1ab \rightarrow s_0a(a, 2)$
$r_4 : s_1b \rightarrow s_0a$
The rules satisfy: $r_1, r_2, r_3 > r_4$.

Compartment i, $2 \leq i \leq n - 1$:
$r_1 : s_0a \rightarrow s_1(b, i - 1)$
$r_3 : s_1ab \rightarrow s_0a(a, i + 1)$
$r_4 : s_1b \rightarrow s_0a$
The rules satisfy: $r_1, r_3 > r_4$.

Compartment n:
$r_1 : s^0a \rightarrow s^1b_{n-1}$
$r_2 : s^1x \rightarrow s^1y$
$r_3 : s^1y \rightarrow s^1z$
$r_4 : s^1z \rightarrow s^0x$

Membranes - Step	C_1	C_2	C_3	C_4	C_5	C_6
0	$s^1 : a^3x$	$s^0 : a^6$	$s^1 : a^9$	$s^0 : a^5$	$s^1 : a^7$	$s^0 : a^8x$
1	$s^1 : a^3b^6x$	$s^1 : \_$	$s^1 : a^9b^5$	$s^1 : \_$	$s^1 : a^7b^8$	$s^1 : x$
2	$s^0 : a^6x$	$s^1 : a^3$	$s^0 : a^9$	$s^1 : a^5$	$s^0 : a^8$	$s^1 : a^7y$
3	$s^0 : a^6y$	$s^1 : a^3b^9$	$s^1 : \_$	$s^1 : a^5b^8$	$s^1 : \_$	$s^1 : a^7z$
4	$s^1 : a^6x$	$s^0 : a^9$	$s^1 : a^3$	$s^0 : a^8$	$s^1 : a^5$	$s^0 : a^7x$
5	$s^1 : a^6b^9$	$s^1 : \_$	$s^1 : a^3b^8$	$s^1 : \_$	$s^1 : a^5b^7$	$s^1 : x$

In the case where we have an odd number of compartments, the $n-$th region must contain an y instead of x. Thus the starting configuration for $n = 7$ is the

following:
$w_{1,0} = a^3x; w_{2,0} = a^6; w_{3,0} = a^9; w_{4,0} = a^5; w_{5,0} = a^7; w_{6,0} = a^8; w_{7,0} = a^{13}y.$

4.3 Static Sorting with P Systems Using Polarizations on Membranes

We now use cell-like P systems with active membranes to specify the same algorithm. P systems with active membranes were introduced with the primary aim of solving NP-complete problems in polynomial (often linear) time [15]. The key features of this variant is the possibility of multiplying the number of compartments during the computation process by using membrane division rules in addition to multiset rewriting and communication rules. Each membrane can have one of the three electrical charges $\{+, -, 0\}$ and a rule can only be executed if the membrane has the required electrical charge; a rule can also change the polarization of the membrane when objects cross it (either in or out).

In our static sorting example compartments with two states were used, so, when the algorithm is implemented using electrical charges, it is expected that two electrical charges would suffice. Indeed, from the list of rules below one may observe that 0 and + are the only polarizations utilised.

There is, however, a problem with this approach, arising from the rule application strategy. In P systems with membrane division and polarizations, only one rule which can change the polarization of a membrane can be applied per step [8]. The sorting algorithm however, employs maximal parallel communication rules to operate the *comparator* procedure between membranes. In order to correctly implement this procedure we will accept maximal parallel communication rules which change the charge of the membrane they traverse to/from *if and only if* they target the same final polarization.

In the case of P systems with polarizations on membranes we will use a cell-like structure with $n = 6$ regions defined below with the initial multisets included and initial polarizations; the implementation of the static sorting with P systems with polarization on membranes is using priorities over the sets of rules.

$$\mu = [[[[[[[a^3x_1]_1^0 a^6x_1]_2^+ a^9x_1]_3^0 a^5x_1]_4^+ a^7x_1]_5^0 a^8x_1]_6^+]_{aux}^0$$

Rules:
"Comparator" rules:
$r_1 : a[]_j^0 \to [b]_j^0, 1 \leq j \leq n;$
$r_2 : [ab]_j^0 \to a[a]_j^+, 1 \leq j \leq n;$
$r_3 : [b \to a]_j^0, 1 \leq j \leq n;$

Rules for switching polarities between adjacent membranes:
$r_4 : [x_1 \to x_2]_j^i, 1 \leq j \leq n;$
$r_5 : [x_2]_j^0 \to y_1[]_j^+, 1 \leq j \leq n;$
$r_6 : [x_2]_j^+ \to y_1[]_j^0, 1 \leq j \leq n;$
$r_7 : [y_1 \to y_2]_j^i, 1 < j \leq n + 1;$

$r_8 : y_2[]_j^0 \rightarrow [x_1]_j^+, 1 \leq j \leq n;$
$r_9 : y_2[]_j^+ \rightarrow [x_1]_j^0, 1 \leq j \leq n;$
where $i \in \{0, +\}$; and the order relationship $r_1, r_2, r_4, r_5, r_6, r_7, r_8, r_9 > r_3$.

M/S	$[]_1$	$[]_2$	$[]_3$	$[]_4$	$[]_5$	$[]_6$	$[]_{aux}$
0	$[a^3x_1]^0$	$[a^6x_1]^+$	$[a_9x_1]^0$	$[a^5x_1]^+$	$[a^7x_1]^0$	$[a^8x_1]^+$	$[\_]^0$
1	$[a^3b^6x_2]^0$	$[x_2]^+$	$[a^9b^5x_2]^0$	$[x_2]^+$	$[a^7b^8x_2]^0$	$[x_2]^+$	$[\_]^0$
2	$[a^6]^+$	$[a^3y_1]^0$	$[a^9y_1]^+$	$[a^5y_1]^0$	$[a^8y_1]^+$	$[a^7y_1]^0$	$[y_1]^0$
3	$[a^6]^+$	$[a^3b^9y_2]^0$	$[y_2]^+$	$[a^5b^8y_2]^0$	$[y_2]^+$	$[a^7y_2]^0$	$[y_2]^0$
4	$[a^6x_1]^0$	$[a^9x_1]^+$	$[a_3x_1]^0$	$[a^8x_1]^+$	$[a^5x_1]^0$	$[a^7x_1]^+$	$[\_]^0$
5	$[a^6b^9x_2]^0$	$[x_2]^+$	$[a_3b^8x_2]^0$	$[x_2]^+$	$[a^5b^7x_2]^0$	$[x_2]^+$	$[\_]^0$

There are no additional requirements in the case where $n = 2k + 1$, however we always entail an extra auxiliary membrane to enable *out* communication of the $n-$th membrane, therefore allowing it to switch polarity.

A similar implementation of the static sorting algorithm can be obtained by using P systems with labels on membranes. As illustrated in [1], we can encode electrical charges in strings of the membrane labels, in order to differentiate between the two necessary states. For each membrane h_i we synthesise its complementary label h'_i, which is changed to by a communication rule. We leave this as an exercise to the reader.

A number of (preliminary) conclusions can be drawn from the above case study:

- kP systems are conceptually closer to tissue P systems than cell-like P systems; in our case studies, this is reflected by the similarity between the specifications using kP systems and tissue P systems, respectively. On the other hand, the model realised using the cell-like P system variant is significantly more complex.
- In terms of complexity, the three implementations are roughly equivalent. The kP system executes in each step one more rule then the P system with states; this rule is either r_2 or r_3 (dealing with p). On the other hand, the number of rules applied in each compartment for every step by cell-like P systems is similar to the case of kP systems.

5 Specification Languages

In this section a specification language covering the entire model of the kP system will be described in Section 5.1 and a specification language with a different syntax for a subset of the same model will be presented in Section 5.2.

5.1 Specification Language for kP Systems

Our Kernel P system research is to be complemented by a set of tools targeting the simulation and trace analysis of various models on the one hand, but also the formal verification by means of automated model checking on the other hand. In order to utilise such applications, a formal, unambiguous representation of the model is required. We have attained this objective by designing a modelling language capable of mapping the kernel P system specification into a machine readable representation. We call this language kP–Lingua.

There are two principal and naturally opposing precepts which influenced and guided the development of kP–Lingua:

1. Conciseness, simplicity, minimalism: the language must consist of a minimal set of (meta-)descriptive symbols, keywords or constructs, such that it satisfies the necessity for an unambiguous syntax.
2. Clarity, coherence, intelligibility: each statement or sequence of statements must overly express an entity with its associated values or a binding between two or more entities in the model; each proposition should be transparent in its context and intuitive, intelligible in the absence of a reference manual.

This proposal stands at the confluence of these orthogonal aspirations and the formal description of the computational model. Since kP systems explicitly apply the *type - instance* paradigm with respect to compartments, a model definition reflects this distinction in its structure, which is bi-partitioned as follows:

1. Type definitions - encompassing the instruction set, organised in accordance with the type's associated execution strategy.
2. Instance definitions and interlinking - establish the set of compartments and related connections, assembling the graph-like structure of membranes.

A type is declared using the keyword `type` followed by the name of the type - any combination of alphanumeric characters excluding the restricted keywords. The body of a type declaration consists of a succession of guarded rules or rule ensembles (choice, arbitrary execution and maximal parallel execution blocks) as specified in the type's execution strategy. A rule is represented as a guarded transition, symbolised by an arrow, between two terms. We illustrate the syntax of a type definition and its constituents with a comprehensive example:

Example 4. A type definition in kP–Lingua.

```
type C1 {
    2a, 3b -> c .
    >= 2c & > 2b : b, c -> a .

    choice {
        b -> 2b .
        < 3b : b ->  3b .
    }

    max {
      a -> a, a(C2), {a, 2b}(C3) .
```

```
        }

        2c -> -(C2) .
        2b -> \-(C2) .
        = 5a : a -> [3a, 3b](C1) [3b](C2) [3a](C3) .
    }
```

In this example we define type C1 with the following sequence of rules: a rewriting rule which takes two a objects and three b objects and produces a c; a guarded rewriting rule which yields an object a if and only if there are at least two c's and more than two b's in the compartment the rule is applied on; next we have a choice block with two rewriting rules of which one is guarded, followed by a maximally parallel block where the rewrite communication rule is exhaustively executed, producing an object a inside the membrane and sending an object a to compartments of type C2, one a and two b's to membranes of type C3 respectively; next, a link creation rule expends two c objects to establish a new connection with an instance of type C2; conversely, the following rule breaks a link with a compartment of type C2, using two b's; finally, a guarded membrane division rule takes one object a and divides the compartment into three distinct compartments of types C1, C2, C3 respectively, if the number of a's in the membrane is precisely five.

The newly created cells are initialised with the a copy of the multiset contained in the divisible compartment, however, one a is substituted with the multiset denoted by the value enclosed in the square brackets.

In kP–Lingua every statement terminates with a full stop and, similar to other programming languages, each block which groups a set of statements together is enclosed in curly braces.

An instance is declared as a typed multiset which also designates the initial configuration of the compartment. Hence, a membrane of type C1 containing two a and three b objects is encoded as {2a, 3b} (C1). We underline our choice of a consistent notation both for type references and multiset values across different declarative contexts. A variable is considered to be a *type variable* if it is enveloped by parentheses. Any other identifier within the scope of a type definition is interpreted as an object and is prefixed by its multiplicity (if greater than one). Membrane instances may also bear an identifier, a variable name nominating a specific compartment in a 'link' statement. We further illustrate instantiation and binding in Example 5.

Example 5. Instantiation and interlinking of compartments expressed in kP–Lingua.

```
m1 {a, b, 3c} (C1).
m2 {10 xx, 10 xy} (C2).
m1 - m2 .
m2 - m3 {} (C2) - {5 xx, 5 xy} (C2) - {m, 2n} (C3) .
m1 - m3 .
```

In the first line, compartment m1 of type C1 is declared, with an initial multiset consisting of an a, a b and three c objects. m2 of type C2 is defined analogous. The third statement is a 'link' instruction, connecting the two previously instantiated compartments. In kernel P systems links are bidirectional and consequently, the *connect* binary operator (`-`) is neutral to the order of its arguments. This, however, becomes relevant when we consider a succession of contiguous vertices in our graph of compartments. Line four emphasises a more condensed way to link instances together: the first one is a reference to compartment `m2` which connects to a new compartment of type `C2` with an empty multiset and label `m3`; this is further joined with an anonymous membrane of type `C2` and finally, a second unlabelled instance of type `C3`. The last line of the example connects `m1` with `m3`, demonstrating the necessity for identifiers and referencing when organising instances in a non-linear structure (i.e., one that is more complex than a list).

We conclude this section by noting the two remaining elements which are not featured in the examples, namely membrane dissolution, symbolised by '#' and the arbitrary execution block respectively. We also acknowledge the expressive power of kP–Lingua whose syntax can intuitively represent a kernel P system model with its plethora of components, using no more than four keywords (`type`, `choice`, `arbitrary`, `max`), five delimiters (`()`, `{}`, `[]`, `:` and `.`), eight relational operators (`=`, `!=`, `<` , `<=`, `>`, `>=`, `&`, `|`) and four meta-symbols (`->`, `-`, `\-`, `#`) to identify an instruction. A complete EBNF formal description of kP–Lingua's syntax is available in the Appendix.

5.2 Specification Language for a Subset of kP Systems

The previous section has described in detail the specification language for kernel P systems (kP systems). However, a previous specification language was provided to specify and simulate simple kernel P systems (skP systems), a simpler version not taking into account some of the detailed current features of this model of computation. The next subsections describe the syntax, the methodology and the software environment provided with this alternative simpler model, illustrating the process of specification and simulation through a case study, the NP-complete Partition problem.

P–Lingua structures

The following structures, including kP–related features, have been added into P–Lingua version 4, which will be shortly available [16]. The current version of P–Lingua describes simple kP systems, a subset of kP systems [11]. Thus, it does not support link creation rules, link destruction rules nor execution strategies (only maximal parallelism). These features might be incorporated in future releases.

Guards: A guard g denoting a relational expression $\#_a(w)\gamma n$ (see Section 3) is represented as $\{\gamma' a * n\}$, where γ' is a representation of γ such as `<` (for $<$), `<=` (for $\leq$), `=` (for $=$), `<>` (for $\neq$), `>=` (for $\geq$) and `>` (for $>$). As illustrative examples, {`<=c*2`} represents the guard denoting $\#_c(w) \leq 2$, whereas{`>=b`} represents $\#_b(w) \geq 1$. As described in Section 4, guards denoting relational expressions (namely relational guards) can be used in Boolean expressions (namely

Boolean guards). Boolean operators involved in Boolean expressions, $\wedge$ and $\vee$, are represented as `&&` and `||`, respectively. For instance, `{<=a*2}&&{<=b}` represents the guard $\{\leq a^2 \wedge \leq b\}$. Similarly, `{<=a*2}||{<=b}` represents the guard $\{\leq a^2 \vee \leq b\}$. $\wedge$ and $\vee$ operators can be combined to describe complex guards, such as
`{<=a*2}&&{<=b}||{<=a*3}&&{<=c*3}`.
A rule r $\{g\}$ is defined as
`@guard` g `?` r.
For instance, rule $a \rightarrow b$ $\{= a^2\}$ is defined as
`@guard {=a*2} ? [a --> b].`

Membrane initialisation: Membrane initialisation enables users to define membranes in the initial configuration. According to the membrane structure of the system defined, the syntax for membrane differs. In this respect,
`mu(`*label*$_1$`)+=[`*multiset*`]'`*label*$_2$`;`
is used for cell–like membrane structures (i.e, those of PDP systems [7]), whereas
`mu(0)*=[`*multiset*`]'`*label*`;`
is used for tissue–like membrane structures, such as those of kernel P systems. The former instruction adds a new membrane labelled *label*$_2$ with associated multiset *multiset* as a child of membrane *label*$_1$, whereas the latter adds a new membrane labelled *label* with associated multiset *multiset* to the initial configuration.

New rules: In order to define the currently supported subset of kernel P systems, some rules defined in Section 6 have been incorporated, which are:
Rewriting and communication rules: A rewriting and communication rule $x \rightarrow y$ $\{g\}$, 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 indicates a compartment type from T, is represented as
`@guard` g `? [`x`]'`t_0 `--> [`a_1`]'`t_1`, ..., [`a_h`]'`t_h,
with t_0 being the current compartment. In contrast to the definition given in Section 6, the P–Lingua implementation of these rules does not require t_0 to be linked to every compartment $t_i, 1 \leq i \leq h$.
Structure changing rules: A structure changing rule
$[x]_{t_{l_i}} \rightarrow [y_1]_{t_{i_1}} \dots [y_p]_{t_{i_p}}$ $\{g\}$,
where $x \in A^+$ and y_j has the form $y_j = (a_{j,1}, t_{j,1}) \dots (a_{j,h_j}, t_{j,h_j})$ like in rewriting and communication rules, is represented in P–Lingua as
`@guard` g `? [`x`]'`t_{l_i} `|--> [`y_1`]'`t_{i_1}`, ..., [`y_p`]'`t_{i_p}`;`.
If any $y_j, 1 \leq j \leq p$ contains the special symbol `@d`, then membrane t_{i_j} is dissolved. These rules can be used in conjunction with membrane internal iterators (see below), resulting in arbitrary division and relabelling rules such as
`[a]'1 |--> [b]'2 &{[c,d{i}]'{i}}:{3<=i<=n};`.

Internal iterators: These iterators enable users to define arbitrary, parameter–dependent multisets, membrane structures and guards. These iterators expand the possibilities for defining rules and initial configurations. The syntax for internal iterators is `&{`*items*`}:{`*index_ranges*`}`, except for ∨–joined guards, which is `|{`*items*`}:{`*index_ranges*`}`. Unless otherwise stated, different internal iterators can be combined in the same rule or sentence, but they cannot be nested. Internal iterators can be used within three contexts:

Multiset internal iterators: The syntax for these iterators is `&{`*multiset*`}:{`*index_ranges*`}`. These iterators allow the extension of multisets in rule definitions. For instance, given a value for n and a set of values for $e_i, 1 \leq i \leq n$, the rules $[a \to b, c_i, d_i^{e_i}, 1 \leq i \leq n]_1$ are represented as `[a --> b, &{c{i}, d{i}*e{i}}:{1<=i<=n}'1;`.

Membrane internal iterators: The syntax for these iterators is `&{[`*multiset*`]'{`*label*`}}:{`*index_ranges*`}`. These iterators allow to specify communications to various compartments occurring on the right–hand side of the rules only. The left–hand side communication cannot be specified by using this mechanism. For instance, given a value for n and values for $e_i, 1 \leq i \leq n$, rewriting and communication rules $x \to (a_i^{e_i}, t_i), 1 \leq i \leq n$, where $x, a_i \in A, 1 \leq i \leq n$ and t_j indicates a compartment type from T, are represented as
`[x]'`t_0` --> &{[a{i}*e{i}]'{i}}:{1<=i<=n};`,
with t_0 being the current compartment.

Guard internal iterators: These iterators are a special case, as they have two possible syntaxes, according to the Boolean operators involved. These forms are
`&{`*guard*`}:{`*index_ranges*`}` `|{`*guard*`}:{`*index_ranges*`}`.
The construct guards joined by ∧ operators (∧–joined guards), whilst the latter uses ∨ operators (∨–joined guards). In addition, they are the only internal iterators which can be nested, in the form of an ∧–joined guard inside an ∨–joined guard. On the other hand, ∨–joined guards cannot be defined into ∧–joined guards. As an example, given a value for n and a value for m,
`@guard |{{&{{<=B{i,j}*2}}:{1<=j<=m}}}:{1<=i<=n} ? [a-->b]'1;`
can be applied iff, prior to the application of the rule, there exists at least one value $i, 1 \leq i \leq n$, such that the cardinality of each object $B_{i,j}, 1 \leq j \leq m$, in membrane 1, is greater than or equal to 2.

There are some constraints regarding indexes in internal iterators; they cannot be part of numerical expressions (such as `&{a{i+1}}:{1 <= i <= 10}`) nor be used as indexes for constants (such as `&{a{g{i}}}:{1 <= i <= 10}`). In addition, names used for indexes in any internal iterator cannot be used anywhere else, including another internal iterator.

A Case Study - Partition Problem

In the previous section we have introduced the subset of kP systems which is included in P–Lingua platform, as well as the main features of the software tool. This section describes a case study to be modelled and simulated, as explained in the next section.

The problem we will focus on is the well-known NP-complete problem called the *partition problem*. This is formulated as follows: let V be a finite set and $weight$, an additive function on V with positive integer values. It is requested to find, if exists, a partition of V, denoted V_1, V_2, such that $weight(V_1) = weight(V_2)$.

A solution to this problem is provided in [9] by using a recogniser tissue P system with cell division and symport/antiport rules. In this case study we adapt the solution to *skP systems*.

Let $V = \{v_1, \ldots, v_n\}$ be a finite set with $weight(v_i) = k_i$, where k_i is a positive integer, $1 \leq i \leq n$. The following *skP system* is built, depending on n (the number of elements in the set), in order to check whether there is a partition, V_1, V_2, with $weight(V_1) = weight(V_2) = k$ (please note we also check that the weights of both subsets are k). The set of component types, $T = \{t_1, t_2\}$, $t_i = (R_i, \sigma_i), 1 \leq i \leq 2$. R_1 and R_2 are given as follows:

- R_1 contains
  $r_{1,1} : S \rightarrow (yes, 0)\ \{\geq T\}$,
  $r_{1,2} : S \rightarrow (no, 0)\ \{\geq F \wedge < T\}$;
  $r_{1,1}$ or $r_{1,2}$ sends an answer *yes* or *no*, respectively, to the environment;
- R_2 contains
  membrane division rules:
  $r_{2,i} : [A_i]_2 \rightarrow [B_i A_{i+1}]_2 [A_{i+1}]_2, 1 \leq i < n$,
  $r_{2,n} : [A_n]_2 \rightarrow [B_n X]_2 [X]_2, 1 \leq i < n$;
  these rules generate in n steps all the subsets of V (2^n subsets); each of them being a potential V_1 and V_2 its complement;
  rewriting rules:
  $r_{2,i,j} : v_i v_j \rightarrow v\ \{= B_i \wedge \neq B_j \wedge = X \vee \neq B_i \wedge = B_j \wedge = X\}$,
  $1 \leq i < j \leq n$,
  $r_{2,n+1} : X \rightarrow Y$; and
  rewriting and communication rules:
  $r_{2,n+2} : Y \rightarrow (F, 1)\ \{\geq v_1 \ \vee \ldots \ \vee \geq v_n \vee \neq v^k\}$,
  $r_{2,n+3} : Y \rightarrow (T, 1)\ \{< v_1 \ \wedge \ldots \ \wedge < v_n \wedge = v^k\}$.

The execution strategies are given by $\sigma_i = Lab(R_i)^{\top}, 1 \leq i \leq 2$, i.e., maximal parallelism. The skP system is given by $sk\Pi_3(n) = (A, \mu, C_1, C_2, 0)$, where:

- A is the alphabet;
- μ is given by the graph with edge $(1, 2)$;
- $C_1 = (t_1, w_{1,0})$, $C_2 = (t_2, w_{2,0})$, where $w_{1,0} = S$, $w_{2,0} = A_1 code(n)$, with $code(n) = {v_1}^{k_1} \ldots {v_n}^{k_n}$ being the code of the weights of the elements in V, and being k, half the sum of the k_i values.

The computation leads to an answer, *yes* or *no*, in $n+3$ steps. In fact, in n steps all the subsets of V are generated, as it can be observed above. In step $n+1$, in each compartment C_2 every occurrence of an element v_i of the subset of V is paired up with an element v_j of the complement as many times as the weights allow to, by using $r_{2,i,j}$ as many times as possible; objects X are simultaneously transformed into objects Y. The step $n+2$ consists of sending either T or F to compartment C_1 depending on whether all the elements of the subset and their complements are paired up and the number of pairs is k (i.e., the weight of the partition), or the weights of the subsets are different or are not equal to k (rules $r_{2,n+3}$ and $r_{2,n+2}$ are respectively used). Finally, in step $n+3$ the answer is provided by using one of the rules of C_1.

This solution might be easily changed to only verify that $weight(V_1) = weight(V_2)$, by simply removing the condition referred to v^k in guards attached to rules $r_{2,n+2}$ and r_{2n+3}.

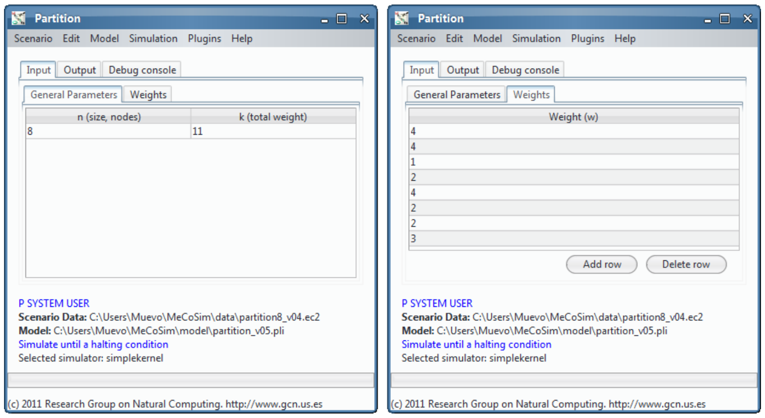
skP Systems Simulation

The previous sections have introduced some background details about P–Lingua specification language and its new features with regard to skP systems, along with the description of a case study, the Partition problem, modelled within this approach. This section has the aim of outlining the basic facts concerning the simulation environment to work with the above presented model.

An integrated methodology for modelling, simulation, analysis and formal verification of skP systems was first presented in [11]. This methodology was supported by the software environment provided by MeCoSim and P–Lingua. P–Lingua provides the P systems designer with a powerful and expressive language to specify skP systems or families of them, possibly including variable parameters whose values depend on the specific instance of the skP system to be generated, such as n for the size of the input set. MeCoSim provides a customisable and extensible visual environment to enable the end user interacting with the P system model implementation. The user provides the input data by custom visual tables, and then runs the simulations (by using the simulation engine of pLinguaCore, that includes simulators for skP systems). Finally, custom visual outputs are generated in the form of tables, charts and/or graphs.

The cited methodology has been applied to our case study, that is, partition problem. The described model has been specified in P–Lingua format, having n and the *weights* in *code*(n) as variable parameters, depending on the specific set to analyse. P–Lingua file and some additional details about the interface and simulation are available at [17].

In addition, a custom application has been defined in MeCoSim, enabling the user entering n, k and the different weights k_1 to k_n for the elements in the input set, as showed in Fig 1. As part of the custom application definition, a mapping is set to translate the input data into parameters for the model written in P–Lingua, possibly after performing additional tasks over the input data. This way, when the user introduces the specific input data and clicks *Simulate!*, the values for the parameters are calculated, the initial configuration is generated, the specific skP system is instantiated, and then the computation runs until a halting configuration is reached.


Partition
Scenario Edit Model Simulation Plugins Help
Input Output Debug console
General Parameters Weights
Partition
Scenario Edit Model Simulation Plugins Help
Input Output Debug console
General Parameters Weights
Add row
Delete row

n (size, nodes)	k (total weight)
8	11

P SYSTEM USER
Scenario Data: C:\Users\Muevo\MeCoSim\data\partition8_v04.ec2
Model: C:\Users\Muevo\MeCoSim\model\partition_v05.pli
Simulate until a halting condition
Selected simulator: simplekernel

(c) 2011 Research Group on Natural Computing. http://www.gcn.us.es

Weight (w)
4
4
1
2
4
2
2
3

P SYSTEM USER
Scenario Data: C:\Users\Muevo\MeCoSim\data\partition8_v04.ec2
Model: C:\Users\Muevo\MeCoSim\model\partition_v05.pli
Simulate until a halting condition
Selected simulator: simplekernel

(c) 2011 Research Group on Natural Computing. http://www.gcn.us.es

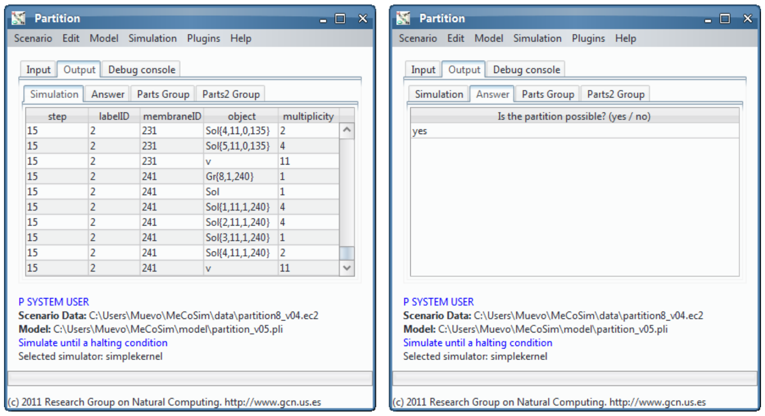
Fig. 1. Input parameters in MeCoSim for Partition

After the simulation has finished, the desired custom outputs are provided to the end user from the results of the computation, depending on the required information to be shown. For instance, a skP systems designer might be interested to know the contents of every membrane, or find out the answer, *yes* or *no*, to the problem. Both outputs have been set in MeCoSim custom application, as shown in Fig 2.

The explained process should be enough to solve the decision problem, but a small additional effort could lead us to provide additional information, such as the elements contained in every specific valid partition, in an automatic way. This additional stuff has been provided by extending the original model with some additional informational objects, and some extra output charts defined for the custom output. The details can be found again in [17], but an example of output chart for a specific partition is shown in Fig. 3.

6 Conclusions

The kP system introduced in this work represents a low level modelling language. Its syntax and informal semantics and some examples have been introduced and discussed. A case study based around a simple sorting algorithm has allowed us to compare different specifications of this using various types of P systems. Finally


Partition
Scenario Edit Model Simulation Plugins Help
Input Output Debug console
Simulation Answer Parts Group Parts2 Group

step	labelID	membraneID	object	multiplicity
15	2	231	Sol{4,11,0,135}	2
15	2	231	Sol{5,11,0,135}	4
15	2	231	v	11
15	2	241	Gr{8,1,240}	1
15	2	241	Sol	1
15	2	241	Sol{1,11,1,240}	4
15	2	241	Sol{2,11,1,240}	4
15	2	241	Sol{3,11,1,240}	1
15	2	241	Sol{4,11,1,240}	2
15	2	241	v	11

P SYSTEM USER
Scenario Data: C:\Users\Muevo\MeCoSim\data\partition8_v04.ec2
Model: C:\Users\Muevo\MeCoSim\model\partition_v05.pli
Simulate until a halting condition
Selected simulator: simplekernel

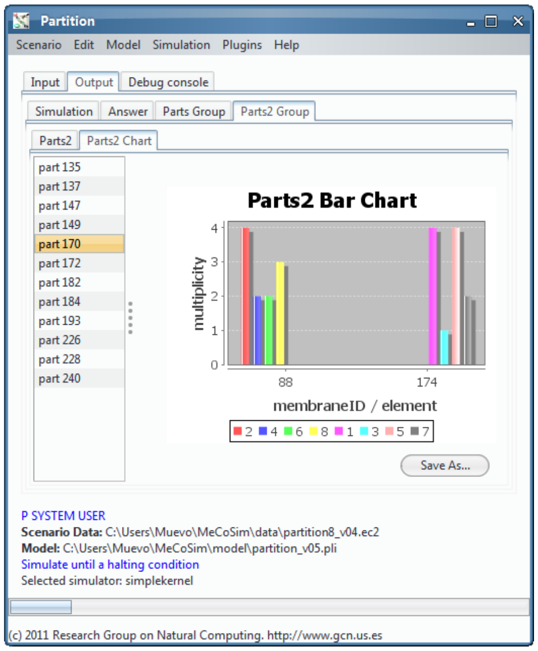
(c) 2011 Research Group on Natural Computing. http://www.gcn.us.es

Is the partition possible? (yes / no)
yes

P SYSTEM USER
Scenario Data: C:\Users\Muevo\MeCoSim\data\partition8_v04.ec2
Model: C:\Users\Muevo\MeCoSim\model\partition_v05.pli
Simulate until a halting condition
Selected simulator: simplekernel

(c) 2011 Research Group on Natural Computing. http://www.gcn.us.es

Fig. 2. Output tables in MeCoSim for Partition



part 135
part 137
part 147
part 149
part 170
part 172
part 182
part 184
part 193
part 226
part 228
part 240

P SYSTEM USER
Scenario Data: C:\Users\Muevo\MeCoSim\data\partition8_v04.ec2
Model: C:\Users\Muevo\MeCoSim\model\partition_v05.pli
Simulate until a halting condition
Selected simulator: simplekernel

(c) 2011 Research Group on Natural Computing. http://www.gcn.us.es

Fig. 3. Partitions Chart in MeCoSim

two specification languages and an implementation are discussed. In the next stage an implementation using the SPIN model checker is expected.

Acknowledgement. The work of MG, FI and LM was partially supported by the MuVet project, Romanian National Authority for Scientific Research (CNCS – UEFISCDI), grant number PN-II-ID-PCE-2011-3-0688; CD's work was supported by a DTA PhD studentship. MGQ, LVC and MJPJ were supported by project TIN 2012-37434 from "Ministerio de Economía y Competitividad" of Spain, and "Proyecto de Excelencia con Investigador de Reconocida Valía P08-TIC-04200" from Junta de Andalucía, both co-financed by FEDER funds. MGQ was also supported by the National FPU Grant Programme from the Spanish Ministry of Education.

References

1. A. Alhazov, L. Pan, Gh. Păun, Trading Polarizations for Labels in P Systems with Active Membranes, *Acta Informatica*, 41, 111-144, 2004.
2. A. Alhazov, D. Sburlan, Static Sorting P Systems. In [5], 215 – 252, 2006.
3. M. Ben-Ari, *Principles of the SPIN Model Checker*, Springer, 2008.
4. R. Ceterchi, C. Martín-Vide, P Systems with Communication for Static Sorting. In *Pre-Proceedings of Brainstorming Week on Membrane Computing, Tarragona, February 2003*, M. Cavaliere, C. Martín-Vide, Gh. Păun, eds., Technical Report no 26, Rovira i Virgili Univ., Tarragona, 101–117, 2003.
5. G. Ciobanu, Gh. Păun, M. J. Pérez-Jiménez, eds., *Applications of Membrane Computing*, Springer, 2006.
6. M. Clavel, F.J. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, Maude: Specification and Programming in Rewriting Logic, *Theoretical Computer Science*, 285, 187 – 243, 2002.
7. M.A. Colomer, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. Comparing simulation algorithms for multienvironment probabilistic P system over a standard virtual ecosystem. *Natural Computing*, 11, 369–379, 2012.
8. D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A Uniform Family of Tissue P Systems with Cell Division Solving 3-COL in a Linear Time, *Theoretical Computer Science*, 404, 76 – 87, 2008.
9. D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A. Riscos-Núñez. A linear time solution to the partition problem in a cellular tissue-like model. *Journal of Computational and Theoretical Nanoscience*, 7, 5, 884 – 889, 2010.
10. M. Gheorghe, F. Ipate, C. Dragomir, Kernel P Systems. In *Membrane Computing, Tenth Brainstorming Week, BWMC 2012, Sevilla, Spain, February 2009*, M. A. Martínez-del-Amor, Gh. Păun, F. Romero-Campero, eds., Universidad de Sevilla, 153 – 170, 2012.
11. M. Gheorghe, F. Ipate, R. Lefticaru, M.J. Pérez-Jiménez, A. Turcanu, L. Valencia, M. Garca-Quismondo, L. Mierlă. 3-COL problem modelling using simple kernel P systems. International Journal of Computer Mathematics, online version (http://dx.doi.org/10.1080/00207160.2012.743712).
12. M. Gheorghe, V. Manca, F.J. Romero-Campero, Deterministic and Stochastic P Ssytems for Modelling Cellular Processes, *Natural Computing*, 9, 457–473, 2010.

13. R. Nicolescu, M. J. Dinneen, Y.-B. Kim, Structured Modelling with Hyperdag P Systems: Part A. In *Membrane Computing, Seventh Brainstorming Week, BWMC 2009, Sevilla, Spain, February 2009*, R. Gutiérrez-Escudero, M. A Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, eds., Universidad de Sevilla, 85 – 107, 2009.
14. Gh. Păun, *Membrane Computing: An Introduction*, Springer, 2002.
15. Gh. Păun, G. Rozenberg, A. Salomaa, eds., *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
16. http://www.p-lingua.org – P–Lingua web page.
17. http://www.p-lingua.org/mecosim/doc/case_studies/partition.html – Partition case study at MeCoSim web page.

7 Appendix

7.1 The EBNF formal description of kP–Lingua's syntax

kpsystem ::= {statement};

statement ::= type definition | instantiation | link;

type definition ::= 'type', space, [space], identifier, [space], '{', [space], {(rule | rule ensemble), [space]}, '}';

rule ensemble ::= ('choice' | 'max' | 'arbitrary'), space, [space], '{', [space], {rule}, [space], '}';

rule ::= [guard, [space], ':'], non-empty multiset, '->' (empty multiset | non-empty multiset | targeted multiset | link creation | link destruction | dissolution | division), [space], '.';

multiset ::= empty multiset | non-empty multiset;

empty multiset ::= '{}';

multiset atom ::= [multiplicity], [space], object;

non-empty multiset ::= multiset atom | (multiset atom, [space], ',', [space], non-empty multiset);

type reference ::= '(', identifier, ')';

targeted multiset atom ::= (multiset atom, [space], type reference) | ('{', non-empty multiset, '}', [space], type reference);

targeted multiset ::= targeted multiset atom | (targeted multiset atom, [space], ‘,’ [space], targeted multiset);

link creation ::= ‘-’, [space], type reference;

link destruction ::= ‘\-’, [space], type reference;

dissolution ::= ‘#’;

division atom ::= ‘[’, [space], [non-empty multiset], [space], ‘]’;

division ::= {division atom, [space], type reference [space]};

instance ::= [identifier], space, [space], ‘{’, multiset, ‘}’, [space], type reference;

instantiation ::= (instance, [space], ‘.’) | (instance, [space], ‘,’, [space], instantiation);

link operand ::= instance | identifier;

link ::= (link operand, [space], ‘-’, [space], link operand) | link, [space], ‘-’, [space], link operand, ‘.’;

letter ::= ? A-Za-z ?;

digit ::= ? 0-9 ?;

alphanumeric ::= letter | ‘_’ | digit;

identifier ::= letter, {alphanumeric};

object ::= letter, {alphanumeric | ‘”};

space ::= ? any space character ?;