13th International Conference on Membrane Computing, CMC13,
Budapest, Hungary, August 28 - 31, 2012. Proceedings, pages 243 - 258.

Using A Kernel P System to Solve The 3-Col Problem

Florentin Ipate[1], Ciprian Dragomir[2], Raluca Lefticaru[1], Laurentiu Mierla[1], and Mario de J. Pérez-Jiménez[3]

[1] Department of Mathematics and Computer Science, University of Piteşti
Str. Târgu din Vale 1, 110040, Piteşti, Romania
{name.surname}@upit.ro
[2] Department of Computer Science, University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
c.dragomir@sheffield.ac.uk
[3] Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n. 41012, Sevilla, Spain
marper@us.es

Abstract. The newly introduced Kernel P systems offer an unitary and elegant way of integrating established features of existing P system variants with new elements with potential value for formal modelling. This paper presents a case study illustrating the expressive power and efficiency of kernel P systems on the 3-Col problem. The use of model checking (in particular of Spin) for formal verification of kernel P systems is also discussed and illustrated in this case.

1 Introduction

Inspired by the behaviour and structure of the living cell, membrane systems, also called P systems, have been intensely studied in recent years [12]. Many variants of P systems have been introduced and investigated in terms of computational power and their capability to solve computationally hard problems [13]. Furthermore, in the last years, significant progress has been made in using different P system models to model, simulate and formally verify various systems [2], including a number of well-known distributed algorithms and problems [11]. In many cases, however, the specifications produced required additional features or constraints compared to the original definition of the P system variant used; such additional features added expressiveness to the specification and clarified complex aspects of the system involved. Although extremely useful for the actual modelling, the ad-hoc addition of such extra features is bound to have an adverse effect on the capability of P systems to provide a coherent analysis and verification framework.

To alleviate this problem, a *kernel P system* (*kP system*, for short) has been recently defined [5]. This is a low level specification language that uses established features of existing P system variants and also includes some new elements.

Most importantly, kP systems offer a coherent way of integrating these elements into the same formalism. In a longer term, it is envisaged that a kP system simulator will be developed and integrated into the P-lingua platform [4] and model checking facilities will also be available, thus providing a coherent platform for system analysis and verification.

Kernel P systems use a graph-like structure (similar to that of tissue P systems) and rules of two types:

- *object processing rules*, which transform and move objects across compartments; these include rewriting and communication rules guarded by promoters and inhibitors, but also symport/antiport like rules;
- *system structure rules*, which change the topology of the system and include membrane division and dissolution and also creation or removal of vertexes in the graph.

The execution of a rule is conditioned by a guard, defined in a general manner using activators and inhibitors. The execution strategy is defined in a more general way that in traditional P systems, using context-free languages; this allows established rule selection strategies, such as maximal parallelism and sequential modes, but also more complex strategies to be expressed in an unitary and elegant way.

This paper illustrates the modelling power of the newly proposed kP system language. For this purpose, it considers a well known NP-complete problem, the 3-colouring (3-Col) problem. A kP system that models this problem is compared with a tissue P system model with cell division available in the literature [3]. The comparison shows that the kP system is more efficient in terms of time complexity and number of rules used, while requiring the same space resources (number of cells). The kP system is also implemented in Spin and a number of interesting properties are extracted and formally verified.

The paper is structured as follows. Section 2 introduces the main elements of kP systems. The modelling of the 3-Col problem is discussed in Section 3. The next two sections are concerned with the formal verification of the kP system model: Section 4 presents the Spin implementation while Section 5 identifies a number of useful properties and discusses their verification using LTL. Finally, conclusions are drawn and future work is outlined in Section 6.

2 Background: kP Systems

This section presents the kP system formalism as defined in [5].

Definition 1. *Given a finite set, A, called* alphabet*, of elements, called* objects*, and a finite set, L, of elements, called* labels*, a* compartment *is a tuple $C = (l, w_0, R^\sigma)$, where $l \in L$ is the label of the compartment, w_0 is the initial multiset over A and R^σ denotes the DNA code of C, which comprises the set of rules, denoted R, applied in this compartment and a regular expression, σ, over $Lab(R)$, the labels of the rules of R.*

The format and the types of rules used by kP systems will be given later, in Section 2.1.

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

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

where A and L are, as in Definition 1, the alphabet *and the set of* labels*, respectively; IO is a multiset of objects from A, called* environment*; μ defines the membrane structure, which is an undirected graph, (V, E), where V are vertices, $V \subseteq L$ (the nodes are labels of these compartments), and E edges; $C_1, \ldots, C_n$ are the n compartments of the system - each compartment is specified according to Definition 1; the labels of the compartments are from L and initial multisets are over A; i_o is the* output compartment *where the result is obtained.*

2.1 kP System Rules

Before proceeding we introduce the notation used. We consider multisets over $A \cup \bar{A}$, where A and $\bar{A}$ are interpreted as **promoters** and **inhibitors**, respectively; $\bar{A} = \{\bar{a} | a \in A\}$. For a multiset w over $A \cup \bar{A}$ and an element a from the same set we denote by $|w|_a$ the number of a's occurring in w. We also consider the set of well-known relational operators $Rel = \{<, \leq, =, \neq, \geq, >\}$. For a multiset $w = a_1^{n_1} \ldots a_k^{n_k}$, $a_j \in A \cup \bar{A}$, $1 \leq j \leq k$, and $\alpha_j \in Rel$, $1 \leq j \leq k$, we introduce the following notation $w' = \alpha_1 a_1^{n_1} \ldots \alpha_k a_k^{n_k}$; a_j is not necessarily unique in w or w'; w' is called *multiset* over $A \cup \bar{A}$ with *relational operators* over Rel.

Each rule r has the form r $\{g\}$, denoting that r is applicable when g is evaluated to true. The guards are constructed according to the following criteria (let g be a guard and pr a predicate over the set of guards):

- $g = \epsilon$ means $pr(\epsilon)$ is always *true*, i.e., no condition is associated with the rule r; this guard is almost always ignored from the syntax of the rule;
- g is a *multiset* over $A \cup \bar{A}$ with *relational operators* over Rel, i.e., $g = \alpha_1 a_1^{n_1} \ldots \alpha_k a_k^{n_k}$, then $pr(w)$ is *true* iff for z, the current multiset of C_i, we have, for every $1 \leq j \leq k$, either (i) if $a_j \in A$ then $|z|_{a_j}$ α_j n_j holds, or (ii) if $a_j \in \bar{A}$, i.e., $a_j = \bar{a}$, $a \in A$, then $|z|_{a_j}$ α_j n_j does not hold;
- $g = w_1 | \ldots | w_p$, i.e., g is a *finite disjunction* of *multisets* over $A \cup \bar{A}$ with *relational operators* over Rel, then $pr(w_1 | \ldots | w_p)$ is *true* iff there exists j, $1 \leq j \leq p$, such that $pr(w_j)$ is *true*.

We denote by $FE(A \cup \bar{A})$, from Finite regular Expressions over $A \cup \bar{A}$ with relational operators, the set of expressions defined above. When a compound guard, cg, referring to compartments l_i and l_j is used, its generic format is $cg = l_i.g_1$ *op* $l_j.g_2$, where g_1, g_2 are finite expressions referring to compartments l_i and l_j, respectively; then, obviously, $pr(cg) = pr(g_1)$ *op* $pr(g_2)$, $op \in \{\&, |\}$, where & stands for *and* and | for *or*, meaning that either both guards are true or at least one is true. Simpler forms, where one of the operands is missing, are also allowed as well as $cg = \epsilon$. A compound guard defines a Boolean condition

defined across the two compartments.

A rule can have one the following types:

- (a) **rewriting and communication** rule: $x \to y\ \{g\}$,
  where $x \in A^+$, $y \in A^*$, $g \in FE(A \cup \bar{A})$; the right hand side, y, has the form $y = (a_1, t_1) \dots (a_h, t_h)$, where $a_j \in A$ and $t_j \in L$, $1 \leq j \leq h$, is an object and a target, i.e., the label of a compartment, respectively; the target, t_j, must be either the label of the current compartment, l_i, (more often ignored) or of an existing neighbour of it ($(l_i, t_j) \in E$) or an unspecified one, $*$; otherwise the rule is not applicable; if a target, t_j, refers to a label that appears more than once then one of the involved compartments will be non-deterministically chosen; if t_j is $*$ then the object a_j is sent to a neighbouring compartment arbitrarily chosen;
- (b) **input-output** rule, is a form of symport/antiport rule: $(x/y)\ \{g\}$,
  where $x, y \in A^*$, $g \in FE(A \cup \bar{A})$; x from the current region, l_i, is sent to the environment and y from the environment is brought into the current region;
- (c) **system structure rules**; the following types are considered:
  - (c1) **membrane division** rule: $[]_{l_i} \to []_{l_{i_1}} \dots []_{l_{i_h}}\ \{g\}$,
    where $g \in FE(A \cup \bar{A})$; the compartment l_i will be replaced by h compartments obtained from l_i, i.e., the content of them will coincide with that of l_i; their labels are $l_{i_1}, \dots, l_{i_h}$, respectively; all the links of l_i are inherited by each of the newly created compartments;
  - (c2) **membrane dissolution** rule: $[]_{l_i} \to \lambda\ \{g\}$;
    the compartment l_i will be destroyed together with its links;
  - (c3) **link creation** rule: $[]_{l_i}; []_{l_j} \to []_{l_i} - []_{l_j}\ \{cg\}$;
    the current compartment, l_i, is linked to l_j and if more than one l_j exists then one of them will be non-deterministically picked up; cg, called compound guard, describes an expression $l_i.g_1\ op\ l_j.g_2$ as defined above;
  - (c4) **link destruction** rule: $[]_{l_i} - []_{l_j} \to []_{l_i}; []_{l_j}\ \{cg\}$;
    is the opposite of link creation and means that compartments l_i, l_j are disconnected; as usual, when more than a link, $(l_i, l_j) \in E$, exists then only one is considered by this rule; cg is a compound guard.

Further details and examples of kP system computations can be found in [5].

2.2 Regular Expressions and their Interpretation for kP Systems

In kP systems the rule execution strategy is described using regular expressions over the sets of labels of rules.

First consider the set of labels of the rules from the set R in a given compartment, denoted by $Lab(R)$. The set of regular expressions over this set is denoted by $REG(Lab(R))$. A regular expression $\sigma \in REG(Lab(R))$ is interpreted as follows:

- $\sigma = \epsilon$ means no rule from the current compartment will be executed;
- $\sigma = r$, $r \in Lab(R)$, means the rule r is executed;

- $\sigma = \alpha\beta$ means first are executed rules designed by α and then those in β, where $\alpha, \beta \subseteq Lab(R)$;
- $\sigma = \alpha|\beta$ means either the rules designed by α or those by β are executed, $\alpha, \beta \subseteq Lab(R)$; often we use the notation defining sets where | is replaced by ,;
- $\sigma = \gamma^*$ means rules designed by γ are executed in a maximal parallel way, $\gamma \subseteq Lab(R)$.

The use of regular expressions allows the usual behaviour of P systems - requiring the rewriting and communication, and input-output rules to be applied in a maximal parallel way and structural rules (e.g. membrane division and dissolution, creation and destruction of links) to be executed one per membrane - as well as other alternative or additional features to be expressed in a consistent and elegant manner [5]. For example, for a one-compartment kP system with object processing rules R_1 and structural rules R_2 the *maximal parallelism* mode can be expressed by $R_1^* R_2$. Furthermore, if a certain order relationship on the object processing rules exists, e.g. $r_1, r_2 > r_3, r_4$ (i.e. when weak priority is applied, the first two rules are executed first, if possible, then the next two), this can be described by $\{r_1, r_2\}^* \{r_3, r_4\}^*$. Considering a hyperdag P system [11], having two rules:

1. $r_1 : x \rightarrow_{min} y$
2. $r_2 : x' \rightarrow_{max} y'$

meaning that r_1 is executed before r_2, such that r_1 is applied only once, while r_2 is applied in a maximal way, the corresponding regular expression for a kP system with this type of execution strategy is: $\{r_1\}^1\{r_2\}^*$. Further details are given in [5].

3 3-Col Problem Specified with kP Systems

In order to illustrate the modelling power of kP systems we consider the 3-Col problem [3]. In general, the k-colouring problem is formulated as follows: given an undirected graph $G = (V, E)$, decide whether or not G is k-colourable; that is, if there exists a k-colouring of G for which every edge $\{u, v\} \in E$ the colours of u and v are different.

As shown in [3], the 3-colouring problem can be solved in linear time by a recognizer tissue P system with cell division and symport/antiport rules [3]. In what follows, we present a kP system model of the same problem and compare the two approaches.

A kP system which solves the 3-Col problem for a graph with $n \geq 2$ vertices is $k\Pi = (A, L, IO, \mu, C_1,\ C_2, 0)$, where

- $A = \{A_1, \ldots, A_n\} \cup \{A_{i,j} | 1 \leq i < j \leq n\} \cup \{B_1, \ldots, B_n\} \cup \{R_1, \ldots, R_n\} \cup \{G_1, \ldots, G_n\} \cup \{a, S, X, Y, T, yes, no\} \cup \{X_1, \ldots, X_{n+3}\}$, where A_i, $1 \leq i \leq n$, stand for the n vertices, $A_{i,j}$, $1 \leq i < j \leq n$ are for all possible edges between the n vertices, B_i, R_i, G_i, $1 \leq i \leq n$, are for the three colours, blue,

red and green, respectively, that can be associated with the n vertices; a is used only in cell 1; S is a flag, X is used to mark that cell division has been completed, Y is used after cell division and T, F at the end; yes, no are the two possible answers - one of them being sent to the environment at the end of the computation; $X_1, \ldots, X_{n+3}$ are used to count the maximum number of steps, $2n + 2$, requested for the last possible input from C_2;

- $L = \{1, 2\}$;
- IO is not relevant in this case; at the end one of two possible answers will be sent out;
- $C_1 = (1, w_{1,0}, R_1^\sigma), C_2 = (2, w_{2,0}, R_2^\sigma)$, where $w_{1,0} = aX_1$, $w_{2,0} = A_1 S\, code(n)$, with $code(n)$ being the multiset of edges of the graph to be coloured;
- μ is given by the graph with edge $(1, 2)$
- R_1^σ and R_2^σ are given by
  - R_1 contains only *rewriting and communication rules*:
    - $X_i \to X_{i+1}, 1 \leq i \leq n + 2$; these rules are used for counting the first $n + 2$ steps;
    - $aT \to (yes, 0)$; in the first $n + 2$ steps, for each solution found, an object T will be sent from C_2 to C_1; when one or more T's are received from compartments C_2, i.e., there is at least one solution, then this rule is used to release yes into the environment;
    - $aX_{n+3} \to (no, 0)\ \{\geq \bar{T}\}$; when no T's are received after $n + 3$ steps, a no is sent out into the environment.
  - R_2 contains
    - *membrane division rules:* $[A_i]_2 \to [B_i A_{i+1}]_2 [G_i A_{i+1}]_2 [R_i A_{i+1}]_2 \{= S\}$, $1 \leq i \leq n - 1$ and $[A_n]_2 \to [B_n X]_2 [G_n X]_2 [R_n X]_2 \{= S\}$; these are applied in n steps and all the possible combinations of colouring n vertices with three colours are obtained;
    - *rewriting and communication rules*
      - $S \to \lambda \{= A_{1,2} = B_1 = B_2 | = A_{1,2} = G_1 = G_2 | = A_{1,2} = R_1 = R_2 | \ldots | = A_{n-1,n} = B_{n-1} = B_n | = A_{n-1,n} = G_{n-1} = G_n | = A_{n-1,n} = R_{n-1} = R_n\}$ (one rule but with a condition containing $3 * n * (n-1)/2$ terms) and $X \to Y$; the first rule checks, for any pair $1 \leq i < j \leq n$, that the colour of i and j is the same and S is available; if so, S is erased; when S disappears, no further verifications are performed in the corresponding cell; when all the verifications are completed, X is transformed into Y by the second rule $X \to Y$;
      - $YS \to (T, 1)$; this rule is applied in the $(n + 2)$th step: if S is available, i.e, there is a solution in the current cell 2, T is sent to cell 1.

The rule execution strategy for compartment 1 is the well-known maximal parallelism mode, i.e. the associated regular expression is R_1^*. Similarly, the rules are applied in a maximal parallel manner in compartment 2, with the constraint that cell division rules may only be applied once per computation step, at the end of the step. That is, if the cell division rules of compartment 2 are denoted by

$R_{2,1}$ and the rewriting and communication rules by $R_{2,2}$, the regular expression associated with compartment 2 is $R_{2,2}^* R_{2,1}$.This is possible as long as the symbols used in division rules are not used by any of the rewriting and communication rules.

The following table summarizes some comparative data concerning the specification of the 3-Col problem with the model from [3] using symport/antiport rules and the one above. This shows that the number of symbols and rules is significantly reduced in comparison to the tissue P system model. Naturally, to some extent, the reduction in the number of rules is achieved at the expense of their complexity: for example, the kP system has one rule with a condition containing $3 * n * (n-1)/2$ terms, also the cell division rules contain 7 symbols plus one used by their guard. The actual number of cells labelled 2 produced by the kP system may in general be (significantly) less than 3^n since cell divisions are only performed when S is available (i.e. when the cell content may lead to a solution); otherwise, when S is no longer available, the cell division stops, as illustrated by the example below. On the other hand, in the case of the tissue P system, 3^n cells labelled 2 are always produced.

Consider, for example, the following configurations for $n \geq 4$:

- $[aX_1]_1[A_1 Scode(n)]_2$ - only the division rule will be applied
- $[aX_2]_1[B_1 A_2 Scode(n)]_2$ - only the division rule will be applied
- $[aX_3]_1[B_1 B_2 A_3 Scode(n)]_2$ - suppose that $A_{1,2}$ is in $code(n)$; then S will disappear at this step (the rewriting rule will be applied) after which the cell will be divided;
- $[aX_4]_1[B_1 B_2 B_3 A_4 code(n)]_2$ - this will no longer evolve as S is no longer available.

It can be observed that an edge (i, j), $1 \leq i < j \leq n$, is checked at the $(j+1)$th step. Therefore, at the $(j+1)$th, S will disappear from all cells for which there exists an edge (i, j), $1 \leq i < j$, and i and j a coloured identically; such cells will no longer divide. Thus, the maximum possible number of cells will only be attained for graphs in which all edges (if any) are of the form (i, n), $i < n$.

Type/Specification	**Tissue P systems**	**kP systems**
Alphabet	$6n^2 + 12n + +2m + 2[log\ m] + 29$	$n(n+1)/2 + 5n + 10$
Rules	$2n$ & $6n(n+1)/2 + 8n + 2m + 3[log\ m] + 25$	n & $n + 7$
Max number of cells	$3^n + 1$	$3^n + 1$
Number of steps	$2n + m + [log\ m] + 11$	$n + 3$

The number of steps for the kP system specification will increase if we consider division rules with no object rewriting features associated with. Indeed, in

this case the rules $[A_i]_2 \to [B_iA_{i+1}]_2[G_iA_{i+1}]_2[R_iA_{i+1}]_2\{= S\}$, $1 \leq i \leq n-1$ and $[A_n]_2 \to [B_nX]_2[G_nX]_2[R_nX]_2\{= S\}$ must be replaced with division rules that do not use object rewriting. Hence we need some additional cells, three more types, namely 21, 22, 23 and new rules to replace the membrane above division rules. The new rules used in the membrane division process are the following:

- membrane division in R_2: $[]_2 \to []_{21}[]_{22}[]_{23}\ \{= A_1 = S| \ldots | = A_n = S\}$; whenever an A_i and a S are present in cell 2, this is divided into three cells of types 21, 22 and 23;
- rewriting rules: $A_i \to B_iA_{i+1} \in R_{21}, 1 \leq i \leq n-1$, $A_n \to B_nX \in R_{21}$, $A_i \to G_iA_{i+1} \in R_{22}, 1 \leq i \leq n-1$, $A_n \to G_nX \in R_{22}$ and $A_i \to R_iA_{i+1} \in R_{23}, 1 \leq i \leq n-1$, $A_n \to R_nX \in R_{23}$;
- membrane division (in fact a label change): $[]_{21} \to []_2\ \{= B_1 = A_2| \ldots | = B_{n-1} = A_n| = B_n = X\} \in R_{21}$; $[]_{22} \to []_2\ \{= G_1 = A_2| \ldots | = G_{n-1} = A_n| = G_n = X\} \in R_{22}$; $[]_{23} \to []_2\ \{= R_1 = A_2| \ldots | = R_{n-1} = A_n| = R_n = X\} \in R_{23}$; cell 21, 22 and 23 will be relabelled 2 when the guards are true;

In this case, $1 + 3n + 3$ rules will replace the n division rules; the maximum number of cells will remain the same, i.e., $3^n + 1$ and the number of steps will become $2n + 3$ (provided that the rewriting and division rules in cells 21, 22 and 23 are executed in the same step. These values are presented in the table below.

Type/Specification	**kP systems**
Alphabet	$n(n+1)/2 + 4n + 8$
Rules	4 & $4n + 7$
Max number of cells	$3^n + 1$
Max number of steps	$2n + 3$

4 Promela Implementation of the kP System

To further demonstrate our solution to the 3-Col problem using kP systems, we will implement, execute and formally verify the proposed model in the Spin verification tool [1]. Originally designed for verifying communications protocols, Spin is particularly suited for modelling concurrent and distributed systems that are based upon interleaving of atomic instructions. Systems are described in a high level language called *Promela* (a process meta language). Promela allows the use of embedded C code as part of the model specifications, facilitating the verification of high level, implementation dependent properties. It also allows for a concise specification of logical correctness requirements, including, but not restricted to requirements expressed in LTL (linear temporal logic). Our goal is twofold: firstly, we aim to simulate the kP system for several instances of the problem and secondly, we attempt to verify the correctness of our model by

asserting the validity of a set of properties/invariants. In this section we address the implementation of the kP system model defined earlier, into Spin's Promela.

There are two essential antinomic aspects usually considered when implementing a specification. On the one hand, an implementation must target coherence, structural clarity and a granularity that confers a certain flexibility for potential alterations and/or extensions. On the other hand, the developer is challenged by the performance requirements, the demand for a reasonably fast and efficient execution. The importance of the latter is particularly augmented in the context of NP complete problems. There is an evident trade-off between the level of abstraction and an optimal representation given by exploiting the particularities of the problem. Additionally, we must take into consideration the issue of property specification in our implementation; that is, we must be able to relate to variables denoting P system elements (i.e. membranes, objects, links).

In accordance with these considerations, we set our primary focus on the development of a model specific implementation, taking into account particularities such as the existence of only two types of membranes, the absence of link creation/destruction rules, the absence of a composite execution strategy, the use of indexed object symbols.

The Spin model checker was successfully used on various other case studies employing membrane systems [8, 7, 10]. We can identify certain (re-usable) patterns of correspondence between P system features and Promela statements and instruction blocks. For example, the guarded parallel rewriting of P systems can be translated using Promela's non-deterministic conditional statements (i.e *if*, *do*); the projection of a membrane to a Promela data type definition. However, our approach differs from existing implementations due to the direct mapping of a membrane's set of rules (the instruction set) to an individual process in Promela. Spin executes processes asynchronously (the programs are interleaved behind the scene, simulating a parallel execution) which makes it a natural support for membrane rules. Hence, we have an expandable set of membrane instances which store a multiset of objects - the *program data*; and a finite set of instructions encapsulated in independent processes - the parallel *instruction set*. Each process will run for each membrane instance it is associated with, performing rewriting and communication but also creating new instances (membrane division). In order to distinguish a computational step in a P system and prevent a rather chaotic process execution, an auxiliary process is introduced as a coordinator: for each step the *Scheduler* process will launch each set of instructions on the instances and will block until all processes have completed (for all membrane instances). The scheduler plays the role of a clock which synchronises execution into steps. This task is repeated until *the environment* is flagged with either "yes" or "no", signalling the computation has halted and a result was generated, or until it reaches a maximum number of steps specified by the user. The input is also defined through a global parameter - the number of vertices and an initial configuration of A_{ij} objects which denote the edges of the graph.

In Table 1 we illustrate the mapping of kP system features to Promela elements. Figure 1 describes the algorithm devised for this kP system solution in

(commented) pseudo-code. A code sample for each of the P system's distinctive components is also provided in the Appendix.

kP system component	Element in Promela	Sample
Membrane type	Data type definition	Fig. 2
Membrane instances	Instances of defined type (organised in an array)	Fig. 3
Objects	Variables/arrays of variables of type int	Fig. 2
Rules (instruction set)	Promela processes	Fig. 4

Table 1. Mapping of kP system features to Promela elements

5 Formal Verification with Spin

In this section we describe the formal verification of some kP systems properties, by extending the approach introduced in [6] for P systems with static structure. This approach was initially used in conjunction with the NuSMV model checker and further developed in [8, 10] for verification of static P systems with the Spin model checker, later adapted in [7, 9] for P systems with active membranes and tissue P systems, respectively.

The main idea behind this formal verification is the transformation of the P system into an associated Kripke structure, for which expected properties are defined. The equivalence between these properties, expressed in terms of P system and the corresponding Kripke structure is formally proved in [6]. However, the executable model of a P or kP system written in Promela is normally implemented as a sequence of transitions (as described in Section 4) and consequently additional (intermediary) states are introduced into the model.

One important assumption is that the intermediary states do not form infinite loops and consequently every possible (infinite) path in the Promela model will contain infinitely often states corresponding to the kP system configurations. This request ensures that every path in the kP system has at least one corresponding path in the Promela model and vice versa.

The properties to be verified in the kP system must be reformulated as equivalent formulas for the associated Promela model. In Table 2 we summarize the transformations of different types of LTL formulas for the Promela specification. The first set of rows describe the transformations of basic LTL formulas, involving temporal logic operators such as Globally (G), Until (U), neXt (X), Finally (F), Release (R), for which the equivalent Promela transformations have been formally proved in [8]. The second set of formulas from Table 2 can be easily derived using the basic LTL operators, for which the transformations are previously defined. In corresponding LTL Promela specifications, $pInS$ represents a flag which is true in the original (non-intermediary) states of the kP system. For example, a property such as 'Eventually, there is YES in the environment', will

```
/* Array of instances for membranes of type Compartment1 */
Compartment1 c1Cells[ ];
/* Array of instances for membranes of type Compartment2 */
Compartment2 c2Cells[ ];

int stepCount = 0;
while(environment is empty AND stepCount < MaxNumberOfSteps) {
    foreach(membraneInstance m in c1Cells) {
        /* Execute process1 which contains all rules
            defined for a membrane of type "Compartment1"
        */
        run process1(m);
    }

    foreach(membraneInstance m in c2Cells) {
        /* Execute process2 which contains all rules
            defined for a membrane of type "Compartment2"
        */
        run process2(m);
    }

    /* Wait until all processes complete
        i.e. when all applicable rules were executed.
    */
    wait();
    /* Computational step complete */
    stepCount++;
}
```

Fig. 1. Pseudo code illustrating the kP system model implementation into Promela

Property	LTL specification
$\mathbf{G}\,p$	`[] (p || !pInS)`
$\mathbf{F}\,p$	`<> (p && pInS)`
$p\,\mathbf{U}\,q$	`(p || !pInS) U (q && pInS)`
$\mathbf{X}\,p$	`X (!pInS U (p && pInS))`
$p\,\mathbf{R}\,q$	`(p && pInS) V (q || !pInS)`
$\mathbf{G}\,(p \rightarrow q)$	`[] (!p || q || !pInS)`
$\mathbf{G}\,(p \rightarrow \mathbf{F}\,q)$	`[] ((p -> <>(q && pInS)) || !pInS)`
$\mathbf{G}\,(p \rightarrow \mathbf{X}q)$	`[] (!p || X(!pInS U (q && pInS)) || !pInS))`

Table 2. Reformulating the P system properties for the Promela implementation

become, for the Promela model, <> (*environment.yes* == 1 && *pInS*), i.e. we expect the number of *yes* objects in the environment to become 1 only for configurations corresponding to the kP system, but not for all the intermediary states.

In the following we present some properties of the kP system which were verified:

- `ltl solYes { ((environment.yes == 0 && environment.no == 0) || !pInS) U (environment.yes == 1 && environment.no == 0 && pInS)}` is a property stating that the number of *yes* and *no* objects in the environment is zero until a *yes* is sent to the environment.
- `ltl solNo { ((environment.yes == 0 && environment.no == 0) || !pInS) U (environment.yes == 0 && environment.no == 1 && pInS)}` is a property stating that the number of *yes* and *no* objects in the environment is zero until a *no* is sent to the environment. This property, checked for a 3-Col model having the number of vertices $n = 3$ and consequently having solution was falsified by the model checker and a counterexample was received.
- `ltl g1 {[] (!(c1Cells[0].a == 1 && c1Cells[0].T >= 1) || X(!pInS U ((environment.yes == 1) && pInS)) || !pInS) }` states that: globally, if in the membrane labelled with 1 there are present the objects a, T, then, in the next state of the kP system, an *yes* object will be sent to the environment.
- `ltl g2 {[] (!(schedulerStep == stepIndex && ((c2Cells[c2CellIdx].B[vtx1] && c2Cells[c2CellIdx].B[vtx2]) || (c2Cells[c2CellIdx].G[vtx1] && c2Cells[c2CellIdx].G[vtx2]) || (c2Cells[c2CellIdx].R[vtx1] && c2Cells[c2CellIdx].R[vtx2])) && c2Cells[c2CellIdx].Aij[vtx1*N + vtx2]) || X(!pInS U (c2Cells[c2CellIdx].S == 0 && pInS)) || !pInS) }` expresses that: if at the computation step given by $c2CellIdx$, in a certain membrane with the label 2, given by the index $c2CellIdx$, there exists an edge between $(vtx1, vtx2)$, and the two vertices are both coloured in blue,

green or red, then, in the next kP system configuration, the S symbol from this membrane will disappear.

The properties expressed before were checked for several kP systems, having different graph structures, which implied obtaining different results for these formulas, $true$ or $false$ plus a corresponding counterexample. For higher values of n the kP system simulation is realised in a few seconds, but the property verification faces the well-known 'state explosion problem'.

6 Conclusions

Kernel P systems offer an unitary and elegant way of integrating established features of existing P system variants with new elements, valuable for formal modelling. This paper presents a case study illustrating the expressive power and efficiency of kernel P systems on the 3-Col problem. It presents a kP system that models the problem and compares it with a tissue P system model available in the literature; the comparison proves the efficiency and expressiveness of kP systems. The paper also makes a first step towards formal verification of kP systems: the kP system model for the 3-Col problem is implemented in Promela and a number of rules for converting a kP system into a Promela implementation are identified. Furthermore, using this implementation, a number of interesting properties are formally verified using Spin.

In future work we will continue to asses the modelling power and efficiency of kP systems in other case studies. Another priority is the development of a platform for simulation and formal verification of kP systems.

Acknowledgement. The work of FI and RL was supported by a grant of the Romanian National Authority for Scientific Research, CNCS–UEFISCDI, project number PN-II-ID-PCE-2011-3-0688. The author MPJ acknowledges the support of the project TIN2009–13192 of the Ministerio de Ciencia e Innovación of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with Investigador de Reconocida Valía of the Junta de Andalucía, grant P08–TIC–04200.

References

1. Ben-Ari, M.: Principles of the Spin Model Checker. Springer-Verlag, London (2008)
2. Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.): Applications of Membrane Computing. Natural Computing Series, Springer (2006)
3. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A uniform family of tissue P systems with cell division solving 3-COL in a linear time. Theoretical Computer Science 404(1-2), 76–87 (2008)
4. Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A P-Lingua programming environment for membrane computing. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 9th International Workshop, WMC 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 5391, pp. 187–203. Springer (2009)

5. Gheorghe, M., Ipate, F., Dragomir, C.: A Kernel P system. Tenth Brainstorming Week on Membrane Computing vol. I, 153–170 (2012)
6. Ipate, F., Gheorghe, M., Lefticaru, R.: Test generation from P systems using model checking. Journal of Logic and Algebraic Programming 79(6), 350–362 (2010)
7. Ipate, F., Lefticaru, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Tudose, C.: Formal verification of P systems with active membranes through model checking. In: Gheorghe, M., Păun, G., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) Int. Conf. on Membrane Computing. Lecture Notes in Computer Science, vol. 7184, pp. 215–225. Springer (2011)
8. Ipate, F., Lefticaru, R., Tudose, C.: Formal verification of P systems using Spin. International Journal of Foundations of Computer Science 22(1), 133–142 (2011)
9. Lefticaru, R., Ipate, F., Valencia-Cabrera, L., Turcanu, A., Tudose, C., Gheorghe, M., Jiménez, M.J.P., Niculescu, I.M., Dragomir, C.: Towards an integrated approach for model simulation, property extraction and verification of P systems. Tenth Brainstorming Week on Membrane Computing vol. I, 291–318 (2012)
10. Lefticaru, R., Tudose, C., Ipate, F.: Towards automated verification of P systems using Spin. International Journal of Natural Computing Research 2(3), 1–12 (2011)
11. Nicolescu, R., Dinneen, M.J., Kim, Y.B.: Structured modelling with Hyperdag P systems. In: Gutiérrez-Escudero, R., Gutiérrez-Naranjo, M.A., Păun, G., Pérez-Hurtado, I. (eds.) Membrane Computing, Seventh Brainstorming Week, BWMC 2009. vol. Part A, pp. 85–17. Universidad de Sevilla (2009)
12. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
13. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)

Appendix

```
/* Membrane type and object definition */
typedef Compartment2 {
int A[N] = 0;
int Aij[NN] = 0;
int B[N] = 0;
int G[N] = 0;
int R[N] = 0;
int X = 0;
int S = 0;
int Y = 0;
int isDisolved;
int isComputing;
}
```

Fig. 2. Sample of membrane type and object specification

```
M1 instM1[MAX_M1_COUNT]; /* global array of membrane instances */
```

Fig. 3. Sample of membrane instances global array

```
proctype M1Rules(int instIndex) {
    instM1[instIndex].isComputing = true;

    /* rewriting rules */
    /* non-deterministic rules application */
    do
    ::  instM1[instIndex].A && instM1[instIndex].B &&
        instM1[instIndex].D >= 1 ->   /* guard */
            instM1[instIndex].A--; instM1[instIndex].B--;
            instM1[instIndex].D++;
    ::  c1Cells[cellIndex].A && c1Cells[cellIndex].C &&
        instM1[instIndex].D < 2 ->   /* guard */
            instM1[instIndex].A--; instM1[instIndex].C--;
    ::  else -> break;
    od;

    /* communication rules */
    if
    :: instM1[instIndex].A && instM1[instIndex].B ->
            instM1[instIndex].A--; instM1[instIndex].B--;
            /* outgoing symbols are stored global buffers */
            /* until the end of the current step */
            goingToM2.S++;
    :: else -> skip;
    fi;

    /* membrane division rules */
    if
    ::  instM1[instIndex].D == 3 ->    /* guard */
        d_step {
            /* copyCell(srcM1,destM1): inline macro defined in Promela */
            copyCell(instM1[instIndex], instM1[totalM1Count]);
            totalM1Count++;
            copyCell(instM1[instIndex], instM1[totalM1Count]);
            totalM1Count++;
        }
        instM1[instIndex].isDisolved = true;
    ::  else -> skip;
    fi;

    instM1[instIndex].isComputing = false;
}
```

Fig. 4. Sample of Promela encoding for the following set of rules: $AB \to D\{\geq D\}$, $AC \to \lambda\{< D^2\}$, $AB \to (S, 2)$, $[]_1 \to []_{11}[]_{12}\{= D^3\}$