# Testing Identifiable Kernel P Systems Using an X-machine Approach

Marian Gheorghe[1], Florentin Ipate[2], Raluca Lefticaru[1,2], Ana Ţurlea[2]

[1] School of Electrical Engineering and Computer Science,
University of Bradford, West Yorkshire, Bradford BD7 1DP, UK
{m.gheorghe,r.lefticaru}@bradford.ac.uk

[2] Department of Computer Science,
Faculty of Mathematics and Computer Science and ICUB
University of Bucharest,
Str. Academiei nr. 14, 010014, Bucharest, Romania
florentin.ipate@ifsoft.ro,ana.turlea@fmi.unibuc.com

**Summary.** This paper presents a testing approach for kernel P systems (*kP systems*), based on the X-machine testing framework and the concept of cover automaton. The testing methodology ensures that the implementation conforms the specifications, under certain conditions, such as the *identifiably* concept in the context of kernel P systems.

**Keywords:** membrane computing; kernel P systems; X-machines; cover automata; testing.

## 1 Introduction

Membrane computing [19] is a research field initiated twenty years ago [17, 18] by Gheorghe Păun. Initially inspired by the structure and functioning of the living cells, the field has known a fast development, different types of membrane systems (or *P systems*) being investigated.

Having so many computational models (cell-like, tissue-like P systems, P colonies, kernel P systems) and also different software implementations for these models, it is important to devise testing methodologies that ensure that the implementation conforms with the specification. The testing task is not trivial, given the fact that the models are parallel and non-deterministic. Previous works on P systems testing include testing cell-like P systems with methods like finite state-based inspired [13], stream X-machine based testing [14], mutation testing for evaluating the efficiency of the test sets [16], model-checking based testing [15].

In this paper we will present a testing approach for *kernel P systems*, which is based on the X-machine testing approach and has as core concept the *identifiability* of multisets of rules. Kernel P systems are a model introduced in [9], which can be simulated using a software framework, called kPWorkbench [5] or some earlier variants (so called *simple kP systems*) using P-Lingua and the MeCoSim simulator [11].

This paper is structured as follows: Section 2 presents the preliminaries regarding kP systems and theoretical background regarding automata and X-machine based testing. Section 3 introduces the concept of *identifiable* kernel P systems, while Section 4 illustrates our testing approach for kP systems. Finally, conclusions are presented in Section 5.

## 2 Preliminaries

This section briefly presents the notations used, then gives the basic definitions regarding kernel P systems [9] and presents the previous testing approaches for automata and X-machines, that have been applied also for testing simple cell-like P systems.

In the following we introduce the notations used in the paper. For a finite alphabet $A = \{a_1, ..., a_p\}$, $A^*$ represents the set of all strings (sequences) over $A$. The empty string is denoted by $\lambda$ and $A^+ = A^* \setminus \{\lambda\}$ denotes the set of non-empty strings. $A^n$ denotes the set of all strings of length $n$, $n \geq 0$, with members in the alphabet $A$, and $A[n] = \bigcup_{0 \leq i \leq n} A^i$ denotes the set of all strings of length at most $n$.

For a string $u \in A^*$, $|u|_a$ denotes the number of occurrences of $a$ in $u$, where $a \in A$. For a subset $S \subseteq A$, $|u|_S$ denotes the number of occurrences of the symbols from $S$ in $u$. The length of a string $u$ is given by $\sum_{a_i \in A} |u|_{a_i}$. The length of the empty string is 0, i.e. $|\lambda| = 0$.

A multiset over $A$ is a mapping $f : A \to \mathbb{N}$. Considering only the elements from the support of $f$ (where $f(a_{i_j}) > 0$, for some $j$, $1 \leq j \leq p$), the multiset is represented as a string $a_{i_1}^{f(a_{i_1})} \ldots a_{i_p}^{f(a_{i_p})}$, where the order is not important. In the sequel multisets will be represented by such strings.

### 2.1 Kernel P systems

In the following we will give a formal definition of kernel P systems (or kP systems) [9]. We start by introducing the concept of a *compartment type* utilised later in defining the compartments of a kernel P system (kP system).

**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, $\sigma_i$, defined over $Lab(R_i)$, the labels of the rules of $R_i$.*

Kernel P systems have features inspired by object-oriented programming, for example one *compartment type* can have one or more *instances*. These instances share the same set of rules and execution strategies (so will deliver the same functionality), but they may contain different multisets of objects and different neighbours according to the graph relation specified.

**Definition 2.** *A* kP system *of degree $n$ is a tuple $k\Pi = (A, \mu, C_1, \ldots, C_n, i_0)$, where*

- *$A$ is a finite set of elements called objects;*
- *$\mu$ defines the membrane structure, which is a graph, $(V, E)$, where $V$ is a set of vertices representing components (compartments), and $E$ is a set of edges, i. e., links between components;*
- *$C_i = (t_i, w_{i,0})$, $1 \le i \le n$, is a compartment of the system consisting of a* compartment type*, $t_i$, from a set $T$ and an initial multiset, $w_{i,0}$ over $A$; the type $t_i = (R_i, \sigma_i)$ consists of a set of evolution rules, $R_i$, and an execution strategy, $\sigma_i$;*
- *$i_0$ is the output compartment where the result is obtained.*

In this paper we will only deal with a simplified version of kP systems having *one single compartment* as this does not affect the general method introduced here and makes the presentation easier to follow. For details regarding the ways of flattening an arbitrary P system, including the kP system discussed in this paper, we refer mainly to [7], but similar approaches are also presented in other papers ([20], [1]). The kP system will be denoted $k\Pi = (A, \mu_1, C_1, 1)$, where $\mu_1$ denotes the graph with one node.

Within the general kP systems framework, the following types of evolution rules have been considered so far:

- *rewriting and communication* rule: $x \to y\{g\}$, where $g$ represents a **guard** (will be formally explained in Def. 4), $x \in A^+$ and $y \in A^*$, where $y$ is a multiset with potential different compartment type targets (each symbol from the right side of the rule can be sent to a different compartment, specified by its type; if multiple compartments of the same type are linked to the current compartment, then one is randomly chosen to be the target). Unlike cell-like P systems, the targets in kP systems indicate only the types of compartments to which the objects will be sent, not particular instances (for example, $y = (a_1, t_1) \ldots (a_h, t_h)$, where $h \ge 0$, and for each $1 \le j \le h$, $a_j \in A$ and $t_j$ indicates a compartment type from $T$).
- *structure changing* rules: membrane division, membrane dissolution, link creation and link destruction rules, which all may also incorporate complex guards and that are covered in detail in [9]. However, this type of rules will not be considered in the following discussion.

*Remark 1.* In the context of *one compartment kP systems*, there will be no need to specify the target compartment, so the rules will be simple communication rules, which in addition can have *guards*. Each rule occurring in the following discussion has the form $r : x \to y\{g\}$, where $r$ identifies the rule and is called **label**, $x \to y$ is the rule itself and $g$ is its **guard**. The part $x \to y$ is also called the **body** of the rule, denoted also $b(r)$. The guards are constructed using multisets over $A$, as operands, and relational or Boolean operators. The definition of the guards is now introduced. We start with some notations.

For a multiset $w$ over $A$ and an element $a \in A$, we denote by $|w|_a$ the number of objects $a$ occurring in $w$. Let us denote $Rel = \{<, \leq, =, \neq, \geq, >\}$, the set of relational operators, $\gamma \in Rel$, a relational operator, and $a^n$ a multiset, consisting of $n$ copies of $a$. We first introduce an *abstract relational expression.*

**Definition 3.** *If $g$ is the* abstract relational expression *denoting $\gamma a^n$ and $w$ a multiset, then the guard $g$ applied to $w$ denotes the* relational expression *$|w|_a \gamma n$.*

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

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

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

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

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

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

*Example 1.* If $g$ is the guard defined by the abstract Boolean expression $\geq a^4 \wedge < b^2 \vee \neg > c$ and $w$ a multiset, then $g$ applied to $w$ is true if it has at least 4 $a'$s and less than 2 $b'$s or no more than one $c$.

In addition to its evolution rules, each compartment type in a kP system has an associated *execution strategy.* The rules corresponding to a compartment can be grouped in blocks, each having one of the following strategies:

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

**Definition 5.** *For a compartment type $t = (R, \sigma)$ from $T$ and $r \in Lab(R)$, $r_1, \ldots, r_s \in Lab(R)$, the* execution strategy*, $\sigma$, 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 non-deterministically chosen and executed; if none is applicable then nothing 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 the* maximal parallelism *strategy;*
- $\sigma = \sigma_1 \& \ldots \& \sigma_s$, *means executing sequentially* $\sigma_1, \ldots, \sigma_s$, *where* $\sigma_i$, $1 \leq i \leq s$, *describes any of the above cases; if one of* $\sigma_i$ *fails to be executed then the rest is no longer executed.*

These execution strategies and the fact that in any compartment several blocks with different strategies can be composed and executed offer a lot of flexibility to the kP system designer, similarly to procedural programming.

**Definition 6.** *A* configuration *of a kP system, $k\Pi$, with $n$ compartments, is a tuple $c = (c_1, \ldots, c_n)$, where $c_i \in A^*$, $1 \leq i \leq n$, is the multiset from compartment $i$. The* initial configuration *is $(w_1, \ldots, w_n)$, where $w_i \in A^*$ is the initial multiset of the compartment $i$, $1 \leq i \leq n$.*

A *transition* (or *computation step*), introduced by the next definition, is the process of passing from one configuration to another.

**Definition 7.** *Given two configurations $c = (c_1, \ldots, c_n)$ and $c' = (c'_1, \ldots, c'_n)$ of a kP system, $k\Pi$, with $n$ compartments, where for any $i, 1 \leq i \leq n$, $u_i \in A^*$, and a multiset of rules $M_i = r_{1,i}^{n_{1,i}} \ldots r_{k_i,i}^{n_{k_i,i}}$, $n_{j,i} \geq 0$, $1 \leq j \leq k_i, k_i \geq 0$, a* transition *or a* computation step *is the process of obtaining $c'$ from $c$ by using the multisets of rules $M_i$, $1 \leq i \leq n$, denoted by $c \Longrightarrow^{(M_1,\ldots,M_n)} c'$, such that for each $i$, $1 \leq i \leq n$, $c'_i$ is the multiset obtained from $c_i$ by first extracting all the objects that are in the left-hand side of each rule of $M_i$ from $c_i$ and then adding all the objects $a$ that are in the right-hand side of each rule of $M_i$ represented as $(a, t_i)$ and all the objects $b$ that are in the right-hand side of each rule of $M_j$, $j \neq i$, such that $b$ is represented as $(b, t_i)$.*

In the theory of kP systems, each compartment might have its own execution strategy. In the sequel we focus on three such execution strategies, namely maximal parallelism, arbitrary parallelism (also called asynchronous execution) and sequential execution. These will be denoted by *max*, *async* and *seq*, respectively. When in a transition from $c$ to $c'$ using $(M_1, \ldots, M_m)$, we intend to refer to a specific transition mode $tm$, $tm \in \{max, async, seq\}$, then this will be denoted by $c \Longrightarrow_{tm}^{(M_1,\ldots,M_m)} c'$.

A *computation* in a P system is a sequence of transitions (computation steps).

A configuration is called *final configuration*, if no rule can be applied to it. In a final configuration the computation stops.

As usual in P systems, we only consider terminal computations, i.e., those arriving in a final configuration and using one of the above mentioned transition modes. We are now ready to define the result of a computation.

**Definition 8.** *For a kP system $k\Pi$ using the transition mode $tm$, $tm \in \{max, async, seq\}$, in each compartment, we denote by $N_{tm}(\Pi)$ the number of objects appearing in the output compartment of a final configuration.*

Two kP systems $k\Pi$ and $k\Pi'$ are called *equivalent* with respect to the transition mode $tm$, $tm \in \{max, async, seq\}$, if $N_{tm}(k\Pi) = N_{tm}(k\Pi')$.

In this paper we will only deal with kP systems having *one single compartment* as this does not affect the general method introduced here and makes the presentation easier to follow. Indeed, limiting the investigation to one compartment kP systems does not affect the generality of it due to the fact that there are ways of flattening an arbitrary P system, including the kP system discussed in this paper, into a P system with one single compartment. For details regarding the flattening of a P system we refer mainly to [7], but similar approaches are also presented in other papers ([20], [1]). Such a kP system will be denoted $k\Pi = (A, \mu_1, C_1, 1)$, where $\mu_1$ denotes the graph with one node. The rules on the right-hand side will have multisets over $A$, as in the case of one single compartment there is no need to indicate where objects are sent to.

### 2.2 The *W*-method for testing finite cover automata

In the following subsection we introduce the basic finite cover automata concepts [3, 12] and the $W$-method for generating test suites from finite cover automata [13]. We will consider only deterministic finite automata.

#### Finite Cover Automata

**Definition 9.** *A finite automaton (abbreviated FA) is a tuple* $A = (V, Q, q_0, F, h)$, *where:*

- $V$ *is the finite input alphabet;*
- $Q$ *is the finite set of states;*
- $q_0 \in Q$ *is the initial state;*
- $F \subseteq Q$ *is the set of final states;*
- $h : Q \times V \to Q$ *is the next-state function.*

**Definition 10.** *Let* $A = (V, Q, q_0, F, h)$ *be a FA,* $U \subseteq V^*$ *a finite language and* $l$ *the length of the longest sequence(s) in* $U$*. Then* $A$ *is called a* deterministic finite cover automaton *(*DFCA*) of* $U$ *if* $L_A \cap V[l] = U$*. A* minimal *DFCA for* $U$ *is a DFCA for* $U$ *having the least number of states.*

The concept of DFCA was introduced by Câmpeanu et al. [2], [3]. A minimal DFCA have considerably fewer states than the minimal FA that accepts $U$.

#### The *W*-method

In *conformance testing* there is a formal specification of the system (for example a FA) and the aim is to generate a test suite such that whenever the implementation under test (IUT) passes all tests, it is guaranteed to conform to the specification. The IUT is unknown but it is assumed to behave like some element from a set of models, called *fault model*. In the case of the $W$-method, the fault model consists of all FAs $A'$ with the same input alphabet $V$ as the specification $A$, whose number of states $m'$ does not exceed the number of states $m$ of $A$ by more than $k$ ($m' - m \leq k$), where $k \geq 0$ is a predetermined integer that must be estimated by the tester.

The $W$-method was originally devised for when the conformance relation is automata equivalence [4], but in this paper we are interested in conformance for bounded sequences. This problem is described in [10] as follows: given an FA specification $A$ and an integer $l \geq 1$ (the upper bound) such that $L_A$ contains at least one sequence of length $l$, we want to construct a set of sequences of length less than or equal to $l$ that can establish whether the implementation behaves as specified for all sequences in $V[l]$. Since $L_A$ contains at least one sequence of length $l$, $A$ is a $DFCA$ for $L_A \cap V[l]$ and so the test suite will check whether the IUT model $A'$ is also a DFCA for $L_A \cap V[l]$.

A *test suite* will be a finite set $Y_k \subseteq V[l]$ of input sequences that, for every $A'$ in the fault model that is not $V[l]$-equivalent to $A$, will produce at least one erroneous output. That is, $A$ and $A'$ are $V[l]$-equivalent whenever $A$ and $A'$ are $Y_k$-equivalent.

Suppose the specification $A$ used for test generation is a minimal DFCA for $L_A \cap V[l]$. The $W$-method for bounded sequences, as developed in [12], involves the selection of two sets of input sequences, $S$ and $W$, as follows:

**Definition 11.** *$S \subseteq V^*$ is called a proper state cover of $A$ if for every state $q$ of $A$ there exists $s \in S$ such that $h(q_0, s) = q$ and $|s| = level(q)$.*

**Definition 12.** *$W \subseteq V^*$ is called a strong characterisation set of $A$ if for every two states $q_1$ and $q_2$ of $A$ and every $j \geq 0$, if $q_1$ and $q_2$ are $V[j]$-distinguishable then $q_1$ and $q_2$ are $(W \cap V[j])$-distinguishable.*

Naturally, in the above definition, it is sufficient for $q_1$ and $q_2$ to be $(W \cap V[j])$-distinguishable when $j$ is the length of the shortest sequences that distinguish between $q_1$ and $q_2$.

Once $S$ and $W$ have been selected, the test suite is obtained using the formula: $Y_k = SV[k+1](W \cup \{\lambda\}) \cap V[l] \setminus \{\lambda\}$ [12].

## 2.3 X-machine based testing

This subsection presents the X-machine based testing methodology, giving the formal definitions for X-machines, the test transformation of an X-machine and $l$-bounded conformance test suites. For more details and complete proofs [10] can be consulted, here only the main results are given.

An X-machine is a finite automaton in which transitions are labelled by partial functions on a data set X instead of mere symbols [6].

**Definition 13.** *An* X-machine *(XM) is a tuple $Z = (Q, X, \Phi, H, q_0, x_0)$ where:*

- *$Q$ is a finite set of states;*

- *$X$ is the (possible infinite) data set;*
- *$\Phi$ is a finite set of distinct* processing functions*; a processing function is a non-empty (partial) function of type $X \to X$;*
- *$H$ is the (partial)* next-state *function, $H : Q \times \Phi \to Q$;*
- *$q_0 \in Q$ is the initial state;*
- *$x_0 \in X$ is the initial data value.*

We regard an X-machine as a finite automaton with the arcs labelled by functions from the set $\Phi$, which is often called the *type* of $Z$. The automaton $A_Z = (\Phi, Q, H, q_0)$ over the alphabet $\Phi$ is called the *associated finite automaton* (FA) of Z. The language accepted by the automaton is denoted by $L_{A_Z}$.

**Definition 14.** *A* computation *of $Z$ is a sequence $x_0, \dots x_n$, with $x_i \in X, 1 \leq i \leq n$, such that there exist $\phi_1, \dots, \phi_n \in \Phi$ with $\phi_i(x_{i-1}) = x_i, 1 \leq i \leq n$ and $\phi_1 \dots \phi_n \in L_{A_Z}$. The set of computations of $Z$ is denoted by $Comp(Z)$.*

A sequence of processing functions that can be applied in the initial data value $x_0$ is said to be controllable.

**Definition 15.** *A sequence $\phi_1, \dots, \phi_n \in \Phi^*$, with $\phi_i \in \Phi, 1 \leq i \leq n$, is said to be controllable if there exist $x_1, \dots x_n \in X$ such that $\phi_i(x_{i-1}) = x_i, 1 \leq i \leq n$. A set $P \subseteq \Phi^*$ is called controllable if for every $p \in P$, $p$ is controllable.*

Let us assume we have an X-machine specification $Z$ and an (unknown) IUT that behaves like an element $Z'$ of a fault model. In this case, the fault model will be a set of X-machines with the same data set $X$, type $\Phi$ and initial data value $x_0$ as the specification. The idea of test generation from an X-machine is to reduce checking that the IUT $Z'$ conforms to the specification $Z$ to checking that the associated automaton of the IUT conforms to the associated automaton of the X-machine specification.

**Definition 16.** *The test transformation of $Z$ is the (partial) function $t : \Phi^* \to X^*$ defined by:*

- *$t(\lambda) = x_0$.* (1)
- *Let $p \in \Phi^*$ and $\phi \in \Phi$.*
  - *Suppose $t(p)$ is defined. Let $t(p) = x_0 \dots x_n$.*
    - *If $x_n \in dom\phi$ then:*
      - *If $p \in L_{A_Z}$ then $t(p\phi) = t(p)\phi(x_n)$.* (2)
      - *Else $t(p\phi) = t(p)$.* (3)
    - *Else $t(p\phi)$ is undefined.* (4)
  - *Otherwise, $t(p\phi)$ is undefined.* (5)

**Lemma 1.** *Let $t$ be a test transformation of $Z$ and $p = \phi_1 \dots \phi_n$, with $\phi_1, \dots, \phi_n \in \Phi$.*

- *Suppose $p$ is controllable and let $x_1, \dots, x_n \in X$ such that $\phi_i(x_{i-1}) = x_i, 1 \leq i \leq n$.*

  - *If $p \in L_{A_Z}$, then $t(p) = x_0 \dots x_n$.*
  - *If $p \notin L_{A_Z}$, then $t(p) = x_0 \dots x_{k+1}$, where $0 \leq k \leq n-1$, is such that $\phi_1 \dots \phi_k \in L_{A_Z}$ and $\phi_1 \dots \phi_k \phi_{k+1} \notin L_{A_Z}$.*
- *If $p$ is not controllable, then $t(p)$ is not defined.*

In order to establish that the associated automaton of the IUT $Z'$ conforms to the associated automaton of the X-machine specification $Z$, we have to be able to identify the processing functions that are applied when the computations of $Z$ and $Z'$ are examined.

**Definition 17.** *$\Phi$ is called* identifiable *if for all $\phi_1, \phi_2 \in \Phi$, whenever there exists $x \in X$ such that $\phi_1(x) = \phi_2(x)$, $\phi_1 = \phi_2$.*

If $\Phi$ is identifiable, then we are able to establish if a controllable sequence of processing functions is correctly implemented by examining the computations of the specification $Z$ and the implementation $Z'$, as shown by the following lemma.

**Lemma 2.** *Let $Z$ and $Z'$ be XMs with type $\Phi$. Suppose $\Phi$ is identifiable. Let $p = \phi_1 \dots \phi_n \in \Phi^*$, with $\phi_i \in \Phi$, $1 \leq i \leq n$, be a controllable sequence. Suppose $t(p)$ is a computation of $Z$ if and only if $t(p)$ is a computation of $Z'$. Then $p \in L_{A_Z}$ if and only if $p \in L_{A'_Z}$ .*

**Definition 18.** *Let $Z$ be an X-machine and $C$ a fault model for $Z$. An* l-bounded conformance test suite *for $Z$ w.r.t. $C$, $l > 0$, is a set $T \subseteq X[l+1]$ such that for every $Z' \in C$ the following holds: if $T \cap Comp(Z) = T \cap Comp(Z')$ then $Comp(Z) \cap X[l+1] = Comp(Z') \cap X[l+1]$.*

That is, whenever any element of $T$ is a computation of $Z$ if and only if it is a computation of $Z'$, $Z'$ conforms to $Z$ for sequences of length up to $l$. The following theorem shows that the test transformation defined earlier provides a mechanism for converting test suites for finite automata into set suites for X-machines.

**Theorem 1.** *Let $Z$ be an XM with type $\Phi$, data set $X$ and initial data value $x_0$. Suppose $\Phi$ is identifiable and $L_{A_Z} \cup \Phi[l]$ is controllable. Let $C$ be a set of XMs such that for every $Z' \in C$, $L_{A'_Z} \cap \Phi[l]$ is controllable. Let $P \subseteq \Phi[l]$, such that, for every $Z' \in C$, whenever $P \cap L_{A_Z} = P \cap L_{A'_Z}$ we have $L_{A_Z} \cap \Phi[l] = L_{A'_Z} \cap \Phi[l]$. Then $t(P)$ is an* l-bounded conformance test suite *for $Z$ w.r.t. $C$.*

Let $l > 0$ be a predefined upper bound. We assume that $\Phi$ is identifiable and $L_{A_Z} \cap \Phi[l]$ is controllable. We assume that $A_Z$, the associated automaton of $Z$, is a minimal DFCA for $L_{A_Z} \cup \Phi[l]$ (if not, this is minimised [3]). Suppose the fault model $C$ is the set of X-machines $Z'$ with the same data set $X$, type $\Phi$ and initial data value $x_0$ as $Z$ such that $L_{A_{Z'}} \cap \Phi[l]$ is controllable, whose number of states $m'$ does not exceed the number of states $m$ of $Z$ by more than $k$ ($m' - m \leq k$), $k \leq 0$. Then an $l$-bounded conformance test suite for $Z$ w.r.t. $C$ is

[3] The minimisation preserves the controlability requirements as the set $L_{A_Z} \cap \Phi[l]$ remains unchanged.

$$T_k = t(S\Phi[k+1](W \cup \{\lambda\}) \cap \Phi[l] \setminus \{\lambda\}),$$

where $S$ is a proper state cover of $A_Z$, $W$ is a strong characterisation set of $A_Z$ and $t$ is a test transformation of $Z$.

## 3 Identifiable transitions in kernel P systems

The concept of identifiable transitions in cell-like P systems was first introduced in [10] and then extended to kernel P systems in [8]. We now aim to present the *identifiability* concept in the context of kP systems and then illustrate how it is used as basis for kP systems testing. The identifiability concept is first introduced for simple rules and then is generalised for multisets of rules.

**Definition 19.** *Two rules $r_1 : x_1 \to y_1\{g_1\}$ and $r_2 : x_2 \to y_2\{g_2\}$ from $R_1$, are said to be* identifiable *in configuration $c$, if they are applicable to $c$ and if $c \Longrightarrow^{r_1} c'$ and $c \Longrightarrow^{r_2} c'$ then $b(r_1) = b(r_2)$.*

According to the above definition the rules $r_1$ and $r_2$ are identifiable in $c$ if when the result of applying them to $c$ is the same then their bodies, $x_1 \to y_1$ and $x_2 \to y_2$, are identical. The rules are not identifiable when the condition from Definition 19 is not satisfied.

A multiset or rules $M = r_1^{n_1} \dots r_k^{n_k}, M \in R_1^*$, where $r_i : x_i \to y_i\{g_i\}$, $1 \le i \le k$, is applicable to the multiset $c$ iff $x_1^{n_1} \dots x_k^{n_k} \subseteq c$ and $g_i$ is true in $c$ for $1 \le i \le k$.

**Definition 20.** *The multisets of rules $M', M'' \in R_1^*$, are said to be* identifiable*, if there is a configuration $c$ where $M'$ and $M''$ are applicable and if $c \Longrightarrow^{M'} c'$ and $c \Longrightarrow^{M''} c'$ then $M' = M''$.*

*Example 2.* Considering the rules $r_1 : a \to x\{\ge a\}$, $r_2 : b \to y\{\ge b\}$, $r_3 : a \to y\{\ge a\}$, $r_4 : b \to x\{\ge b\}$, and the configuration $ab$ it is clear that the multisets of rules $M' = r_1 r_2$ and $M'' = r_3 r_4$ are not identifiable in the configuration $c = ab$, as $c = ab \Longrightarrow^{M'} c' = xy$ and $c = ab \Longrightarrow^{M''} c' = xy$, but $M' \ne M''$.

A kP system $k\Pi$ has its *rules identifiable* if any two multisets of rules, $M', M'' \in R_1^*$, are identifiable.

Given a multiset of rules $M = r_1^{n_1} \dots r_k^{n_k}$, where $r_i : x_i \to y_i\{g_i\}$, $1 \le i \le k$, we denote by $r_M$ the rule $x_1^{n_1} \dots x_k^{n_k} \to y_1^{n_1} \dots y_k^{n_k}\{g_1 \wedge \dots \wedge g_k\}$, i.e., the concatenation of all the rules in $M$. One can observe that the applicability of the multiset of rules $M$ to a certain configuration is equivalent to the applicability of the rule $r_M$ to that configuration. It follows that one can study first the usage of simple rules.

*Remark 2.* For any two rules $r_i : x_i \to y_i$, $1 \le i \le 2$, when we check whether they are identifiable or not one can write them as $r_i : uv_i \to wz_i\{g_i\}$, $1 \le i \le 2$, where for any $a \in V$, $a$ appears in at most one of the $v_1$ or $v_2$, i.e., all the common symbols on the left-hand side of the rules are in $u$. Let us denote by $c_{r_1,r_2}$, the configuration $uv_1v_2$. Obviously this is the smallest configuration in which $r_1$ and $r_2$ are applicable, given that $g_1$ and $g_2$ are true in $uv_1v_2$.

*Remark 3.* If $r_i : x_i \to y_i \ \{g_i\}$, $1 \leq i \leq 2$, are applicable in a configuration $c$ and $c \subseteq c'$ then they are not always applicable to $c'$. They are applicable to $c'$ when all $g_i$, $1 \leq i \leq 2$, are true in $c'$.

*Remark 4.* If the rules $r_1, r_2$ are not applicable to $c_{r_1,r_2}$ then there must be minimal configurations $c$ where the rules are applicable and they are minimal, i.e., there is no $c_1$, $c_1 \subset c$ where the rules are applicable. Such minimal configurations where $r_1, r_2$ are applicable are of the form $tc_{r_1,r_2}$, where $t \in A^*$, $t \neq \lambda$.

In the following we introduce some theoretical results, characterising the identifiability or non-identifiability or rules and multisets of rules under certain conditions. The complete proofs for these the results are given in [8].

**Lemma 3.** *Two rules which are identifiable in a configuration $c$ are identifiable in any configuration containing $c$ in which they are applicable.*

**Lemma 4.** *Two rules which are identifiable in a minimal configuration $c$ are identifiable in any other minimal configuration $c'$ where they are applicable.*

**Corollary 1.** *Two rules $r_1$ and $r_2$ identifiable in a minimal configuration $tc_{r_1,r_2}$, $t \in A^*$, are identifiable in any configuration in which they are applicable.*

**Corollary 2.** *Two multisets of rules $M_1$ and $M_2$ identifiable in $tc_{r_{M_1},r_{M_2}}$, $t \in A^*$, are identifiable in any configuration in which they are applicable.*

From now on, we will always verify the identifiability (or non identifiability) only for the smallest configurations associated with rules or multisets of rules and will not mention these configurations anymore in the results to follow.

The applicability of two rules (multisets of rules) to a certain configuration depends not only on the fact that there left hand sides (the concatenation of the left hand sides) must be contained in the configuration and the guards must be true, but takes into account the execution strategy.

*Remark 5.* For the *async* transition mode two multisets of rules (and two rules) applicable in a configuration are also applicable in any other bigger configuration, when the corresponding guards are true. For the *seq* mode this is true only for multisets with one single element and obviously for simple rules. In the case of the *max* mode the applicability of the multisets of rules (or rules) to various configurations depends on the contents of the configurations and other available rules. For instance if we consider a P system containing the rules $r_1 : a \to a \ \{\geq a\}; r_2 : ab \to abb \ \{\leq b^{100}\}; r_3 : bb \to c \ \{\geq b^2\}$ and the configuration $c = ab$ then in $c$ only $r_1$ and $r_2$ are applicable and identifiable, but in $c_1 = abb$, containing $c$, $r_1$ is no longer applicable, but instead we have $r_2$ and the multiset $r_1r_3$ applicable. In $ab^{101}$ $r_2$ and any multiset containing it are not applicable due to the guard being false; also $r_1$ is no longer applicable, but $r_1r_3^{55}$ is now applicable, due to maximal parallelism.

*Remark 6.* In the following results whenever we refer to arbitrary rules or multisets of rules they are always meant to be applicable with respect to the transition mode.

**Theorem 2.** *The rules $r_1 : x_1 \to y_1 \ \{g_1\}$ and $r_2 : x_2 \to y_2 \ \{g_2\}$, are not identifiable if and only if they have the form $r_1 : uv_1 \to wv_1 \ \{g_1\}$ and $r_2 : uv_2 \to wv_2 \ \{g_2\}$ and for any $a \in A$, $a$ appears in at most one of $v_1$ or $v_2$.*

**Corollary 3.** *The rules $r_1 : uv_1 \to wz_1 \ \{g_1\}$ and $r_2 : uv_2 \to wz_2 \ \{g_2\}$, such that for any $a \in A$, $a$ appears in at most one of $v_1$ or $v_2$, are identifiable if and only if $v_1 \neq z_1$ or $v_2 \neq z_2$.*

**Theorem 3.** *If $r_1$ and $r_2$ are identifiable then $r_1^n$ and $r_2^n$ are identifiable, for any $n \geq 1$.*

## 4 Testing identifiable kernel P systems

In order to generate test suites for a kernel P system using the X-machine testing method, first a corresponding X-machine model needs to be constructed. As discussed in Section 2, multi-compartment P systems can be flattened into one membrane P systems and there are different ways to realise this [1, 7, 20]. Consequently, we will illustrate the testing approach using an one-membrane kP system model $k\Pi = (V, T, \mu_1, w_1, R_1, 1)$. The main idea is to construct an X-machine $Z^t = (Q^t, X, \Phi, H^t, q_0^t, x_0)$, corresponding to the computation tree of $k\Pi$. As the computation tree of the kP system might be infinite, we will consider only computations of maximum $l$ steps, where $l > 0$ is a predefined integer. Let $R_1 = \{r_1, \ldots, r_n\}$ be the set of rules of $k\Pi$. As only finite computations are considered, for every rule $r_i \in R_1$ there will be some $N_i$ such that, in any step, $r_i$ can be applied at most $N_i$ times, $1 \leq i \leq n$. Thus the X-machine $Z^t = (Q^t, X, \Phi, H^t, q_0^t, x_0)$ is defined as follows:

- $Q^t$ is the set of nodes of the computation tree of maximum $l$ steps;
- $q_0^t$ is the root node;
- $X$ is the set of multisets with elements in $V$;
- $x_0$ is the initial multiset $w_1$;
- $\Phi$ is the set of (partial) functions induced by the application of multisets of rules $r_1^{i_1} \ldots r_n^{i_n}$, $0 \leq i_1 \leq N_1, \ldots, 0 \leq i_n \leq N_n$, $i_1 + \ldots i_n > 0$;
- $H^t$ is the next-state function determined by the computation tree.

*Remark 7.* Note that, by definition, $L_{A_Z}$ is controllable, i.e. any sequence of processing functions from the associated automaton $A_Z$ can be applied in the initial data $x_0$ (corresponding to the initial multiset $w_1$). Intuitively a path in the DFCA corresponds to a path in the computation tree of the kP system.

*Remark 8.* The set of (partial) functions, $\Phi$, from the above definition is identifiable (according to Definition 17) if and only if the corresponding multisets of rules are pairwise identifiable (according to Definition 20).

*Example 3.* Let us consider one compartment kP system $k\Pi_1 = (V, V, \mu_1,\ w_1, R_1, 1)$, where $V = \{a, b, c\}$, $w_1 = ab$, and

$$R_1 = \left\{ \begin{array}{ll} r_1 : a \to b\{\geq a \wedge \geq b\} & r_2 : ab \to bc\{\leq a \wedge \geq b\} \\ r_3 : c \to b\{\geq b \wedge \leq c^{100}\} & r_4 : c \to cc\{\leq c^{100}\} \end{array} \right\}$$

Let us build the computation tree considering that rules are applied in the maximally parallel mode. The initial configuration $w_1 = ab$ is at the root of the tree (level 0 of the tree). Two computation steps are possible from the root: $ab \Longrightarrow^{r_1} b^2$ and $ab \Longrightarrow^{r_2} bc$. Then the configurations $b^2$ and $bc$ are at the first level of the tree. No rule can be applied in $b^2$ (this is a terminal configuration of $k\Pi_1$), but two computation steps exist form $bc$: $bc \Longrightarrow^{r_3} b^2$ and $bc \Longrightarrow^{r_4} bc^2$. The new configurations produced represent the second level of the tree. Again, no rule can be applied in $b^2$, but there are three computation steps from $bc^2$: $bc^2 \Longrightarrow^{r_3^2} b^3$, $bc^2 \Longrightarrow^{r_3 r_4} b^2c^2$ and $bc^2 \Longrightarrow^{r_4^2} bc^4$. No rule can be applied in $b^3$, but there are three computation steps from $b^2c^2$ and five from $bc^4$: $b^2c^2 \Longrightarrow^{r_3^2} b^4$, $b^2c^2 \Longrightarrow^{r_3 r_4} b^3c^2$ and $b^2c^2 \Longrightarrow^{r_4^2} b^2c^4$; $bc^4 \Longrightarrow^{r_3^4} b^5$, $bc^4 \Longrightarrow^{r_3^3 r_4} b^4c^2$, $bc^4 \Longrightarrow^{r_3^2 r_4^2} b^3c^4$, $bc^4 \Longrightarrow^{r_3 r_4^3} b^2c^6$ and $bc^4 \Longrightarrow^{r_4^4} bc^8$. The configurations produced by these eight multisets of rules represent the fourth level of the tree.

It can be easily checked that any two of the above multisets of rules are identifiable, according to Corollary 3, and consequently they produce different results when applied to the same configuration – see above.

Let the upper bound on the number of computation steps considered be $l = 4$. For this value of $l$, the rules $r_1$ and $r_2$ have been applied at most once, so $N_1 = 1$ and $N_2 = 1$, whereas rules $r_3$ and $r_4$ have been applied at most four times, so $N_3 = 4$ and $N_4 = 4$. Therefore the type $\Phi$ of the X-machine $Z^t$ corresponding to the computation tree is the set of partial functions induced by the multisets $r_1^{i_1} r_2^{i_2} r_3^{i_3} r_4^{i_4}$, $0 \leq i_1 \leq 1, 0 \leq i_2 \leq 2, 0 \leq i_3 \leq 4, 0 \leq i_4 \leq 4$, $i_1 + i_2 + i_3 + i_4 > 0$. The associated automaton $A_{Z^t}$ is as represented in Figure 1.

Let $L_{A_{Z^t}} \subseteq \Phi^*$ be the language accepted by the associated automaton $A_{Z^t}$. In order to apply the test generation method presented in Section 2.3, an X-machine $Z$ whose associated automaton $A_Z$ is a DFCA for $L_{A_{Z^t}}$ needs to be constructed first.

Let $\leq$ be a total order on $Q^t$ such that $q_1 \leq q_2$ whenever $level(q_1) \leq level(q_2)$ and denote $q_1 < q_2$ if $q_1 \leq q_2$ and $q_1 \neq q_2$. In other words, the node at the superior level in the tree is before the node at the inferior level; if the nodes are at the same level then their order is arbitrarily chosen. Define $P^t = \{q \in Q^t \mid \neg\exists q' \in Q^t \cdot q' \sim q, q' < q\}$ and $[q] = \{q' \in Q^t \mid q' \sim q \wedge \neg\exists q'' \in P^t \cdot q'' \sim q', q'' < q\}$ for every $q \in P^t$ (i.e. $[q]$ denotes the set of all states $q'$ for which $q$ is the minimum state similar to $q'$). Then we have the following result (the proof is given in [10]).

**Theorem 4.** *Let $Z = (Q, X, \Phi, H, q_0, x_0)$, where $Q = \{[q] \mid q \in P^t\}$, $q_0 = [q_0^t]$, $H([q], \phi) = [H^t(q, \phi)]$ for all $q \in P^t$ and $\phi \in \Phi$. Then $A_Z$ is a minimal DFCA for $L_{A_{Z^t}}$.*

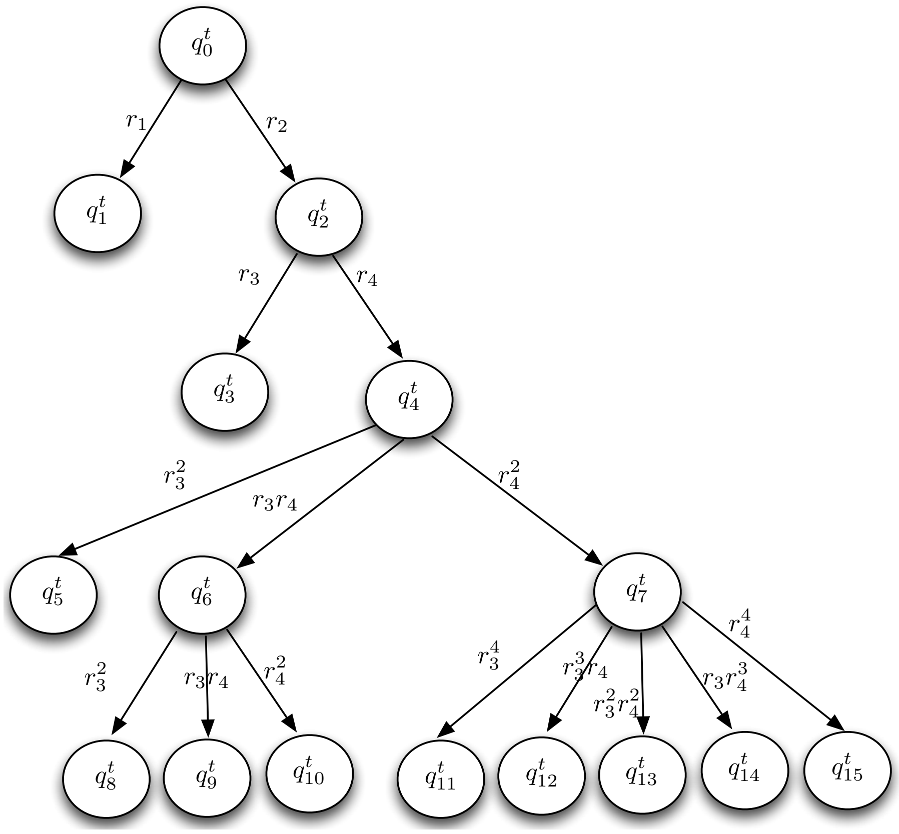

**Fig. 1.** The associated automaton $A_{Z^t}$ corresponding to the computation tree for $k\Pi_1$ and $l = 4$

*Remark 9.* Consider $Z^t$ as in the previous example. $P^t = \{q_0^t, q_1^t, q_2^t, q_4^t, q_7^t\}$; $[q_0^t] = \{q_0^t, q_8^t, q_9^t, q_{10}^t, q_{11}^t, q_{12}^t, q_{13}^t, q_{14}^t, q_{15}^t\}$, $[q_1^t] = \{q_1^t, q_3^t, q_5^t\}$, $[q_2^t] = \{q_2^t\}$, $[q_4^t] = \{q_4^t, q_6^t\}$, $[q_7^t] = \{q_7^t\}$. Then $Z = (Q, X, \Phi, H, q_0, x_0)$, where $Q = \{[q_0^t], [q_1^t], [q_2^t], [q_4^t], [q_7^t]\}$ and $q_0 = [q_0^t]$. The associated automaton of $Z$ is a minimal DFCA for $L_{A_{Z^t}}$ and is as represented in Figure 2.

Once the X-machine $Z$ has been constructed the test generation process entails the following steps:

1. **Construct the sets $S$ and $W$ (proper state cover and characterisation sets, respectively).**
   It can be easily remarked from Fig. 2. that $\lambda$, $r_1$, $r_2$, $r_2$ $r_4$, $r_2$ $r_4$ $r_4^2$ are the sequences of minimum length[4] that reach $[q_0^t]$, $[q_1^t]$, $[q_2^t]$, $[q_4^t]$ and $[q_7^t]$, respectively.

[4] Notation: for rules $r$ and $r'$, $rr'$ denotes the application of rules $r$ and $r'$ in one single step, whereas $r$ $r'$ (separated by space) denotes the application of rule $r$ in one step

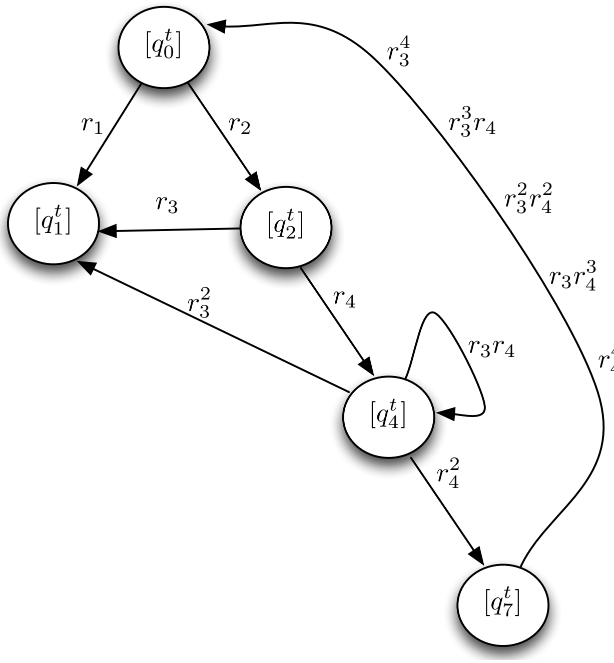

**Fig. 2.** The DFCA for $L_{A_{Z^t}}$

Consequently $S = \{r_1, r_2, r_2\ r_4, r_2\ r_4\ r_4^2\}$ is a proper state cover of $Z$. Furthermore, since $r_1$ distinguishes $[q_0^t]$ from all remaining states and $r_3$, $r_3 r_4$ and $r_3^4$ hold the same property for $[q_2^t]$, $[q_4^t]$ and $[q_7^t]$, respectively, $W = \{r_1, r_3, r_3 r_4, r_3^4\}$ is a strong characterisation set of $Z$.

2. **Determine the fault model of the IUT.**
This entails establishing the transitions that a (possibly faulty) implementation is capable to perform. For example, when the correct application of rules (in the P system specification) is in the maximally parallel mode *max*, one fault that we may consider is when the rules are applied in a less restrictive mode such that the asynchronous mode *async*. Hence the notion of controllability for P systems is defined by considering this, less restrictive, application mode.

**Definition 21.** *A sequence of multisets of rules $p = M_1 ... M_m$, with $M_i \in R_1^*$, $1 \leq i \leq m$, is said to be* controllable *if there exist configurations $u_0 = w_1, u_1, \ldots, u_m$, $u_i \in V^*$, $0 \leq i \leq m$, such that $u_{i-1} \Longrightarrow_{FM}^{M_i} u_i$, $1 \leq i \leq m$, where $u \Longrightarrow_{FM}^{M} u'$ denotes a computation step in the* fault model *from configuration $u$ to configuration $u'$ by applying the multiset of rules $M$.*

followed by the application of rule $r'$ in the following step; the second notation is also used for multisets of rules.

Consider again the P system $k\Pi_1$ as in Example 3. Then $ab \Longrightarrow^{r_2} bc$ and $bc \Longrightarrow^{r_4} bc^2$, but $bc^2 \Longrightarrow^{r_4} bc^3$ does not hold since the rules of $k\Pi_1$ must be applied in the maximally parallel mode. However, if we consider that in the fault model of the IUT rules may be applied in the asynchronous mode, the sequence $r_2\ r_4\ r_4$ is controllable. The fault model is also determined by the maximum number of states $m + k$ that the IUT may have, where $m$ is the number of states of the X-machine $Z$ and $k \geq 0$ is a non-negative integer estimated by the tester.

3. **Construct an $l$-bounded conformance test suite.**
This is $T_k = t(Y_k)$, where $Y_k = S\Phi[k+1](W \cup \{\lambda\}) \cap \Phi[l] \setminus \{\lambda\}$ and $t$ is a test transformation of $Z$.
According to [4], the upper bound for the number of sequences in $S\Phi[k+1]W$ is $m^2 \cdot r^{k+1}$ and the total length of all sequences is not greater that $m^2 \cdot (m + k) \cdot r^{k+1}$, where $r$ is the number of elements of $\Phi$. In particular, for $k = 0$, the respective bounds are $m^2 \cdot r$ and $m^3 \cdot r$. The increase in size produced by replacing $W$ with $W \cup \{\lambda\}$ in the above formula is negligible. Note that these bounds refer to the worst case; in an average case, the size of $Y_k$ is much lower. Furthermore, the size of $t(Y_k)$ is normally significantly lower than the size of $Y_k$ since only the controllable sequences are in the domain of $t$.
The construction of $Y_k$ is straightforward, so we illustrate only the construction of the test transformation $t$ with an example. Consider again rule application mode is maximal parallelism for $k\Phi$ and the asynchronous mode for the fault model. Consider the sequences $s_0 = \lambda$, $s_1 = r_2$, $s_2 = s_1\ r_4$, $s_3 = s_2\ r_4$, $s_4 = s_3\ r_4$, $s_5 = s_4\ r_1$ and $s_6 = s_5\ r_1$. By rule (1) of Definition 16, $t(s_0) = x_0 = ab$. As $ab \Longrightarrow^{r_2} bc$, by rule (2) $t(s_1) = ab\ bc$. Similarly, as $bc \Longrightarrow^{r_4} bc^2$, by rule (2) $t(s_2) = ab\ bc\ bc^2$. On the other hand $r_4$ cannot be applied in configuration $bc^2$ in the maximally parallel mode, but $bc^2 \Longrightarrow^{r_4}_{FM} bc^3$ (in the asynchronous mode) and so, by rule (2), $t(s_3) = ab\ bc\ bc^2\ bc^3$. Furthermore, $bc^3 \Longrightarrow^{r_4}_{FM} bc^4$ and so, by rule (3) of Definition 16, $t(s_4) = t(s_3) = ab\ bc\ bc^2\ bc^3$. As $r_1$ cannot be applied in $bc^4$, by rule (4) $t(s_5)$ is undefined. Furthermore, by rule (5), $t(s_6)$ is also undefined, so no test sequences will be generated for $s_5$ and $s_6$.

## 5 Conclusions

This paper presents a testing approach for kernel P systems that, under certain conditions, ensures that the implementation conforms to the specification. The methodology is based on the *identifiable kernel P systems* concept, which is essential for testing, and has been introduced for one-compartment kP systems with rewriting rules, but could be extended.

## Acknowledgements

This work is supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-III-P4-ID-PCE-2016-0210.

## References

1. Agrigoroaiei, O., Ciobanu, G.: Flattening the transition P systems with dissolution. In: Gheorghe, M., Hinze, T., Paun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing - 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers. Lecture Notes in Computer Science, vol. 6501, pp. 53–64. Springer (2010), https://doi.org/10.1007/978-3-642-18123-8_7
2. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. In: International Workshop on Implementing Automata. pp. 43–56. Springer (1998), https://doi.org/10.1007/3-540-48057-9_4
3. Câmpeanu, C., Santean, N., Yu, S.: Minimal cover-automata for finite languages. Theoretical Computer Science 267(1-2), 3–16 (2001), https://doi.org/10.1016/S0304-3975(00)00292-9
4. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Transactions on Software Engineering 4(3), 178–187 (1978), https://doi.org/10.1109/TSE.1978.231496
5. Dragomir, C., Ipate, F., Konur, S., Lefticaru, R., Mierla, L.: Model checking kernel p systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing. Lecture Notes in Computer Science, vol. 8340, pp. 151–172. Springer Berlin Heidelberg (2014), https://doi.org/10.1007/978-3-642-54239-8_12
6. Eilenberg, S.: Automata, languages, and machines. Academic press (1974)
7. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C.: Flattening in (tissue) P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing. Lecture Notes in Computer Science, vol. 8340, pp. 173–188. Springer Berlin Heidelberg (2014), https://doi.org/10.1007/978-3-642-54239-8_13
8. Gheorghe, M., Ipate, F.: Identifiable kernel P systems. Submitted (2018)
9. Gheorghe, M., Ipate, F., Dragomir, C., Mierla, L., Valencia-Cabrera, L., García-Quismondo, M., Pérez-Jiménez, M.J.: Kernel P Systems - Version I. Eleventh Brainstorming Week on Membrane Computing (11BWMC) pp. 97–124 (2013), http://www.gcn.us.es/files/11bwmc/097_gheorghe_ipate.pdf
10. Gheorghe, M., Ipate, F., Konur, S.: Testing based on identifiable P systems using cover automata and X-machines. Information Sciences 372, 565–578 (2016), https://doi.org/10.1016/j.ins.2016.08.028
11. Gheorghe, M., Ipate, F., Lefticaru, R., Pérez-Jiménez, M.J., Turcanu, A., Valencia-Cabrera, L., García-Quismondo, M., Mierla, L.: 3-col problem modelling using simple kernel P systems. International Journal of Computer Mathematics 90(4), 816–830 (2013), https://doi.org/10.1080/00207160.2012.743712
12. Ipate, F.: Bounded sequence testing from deterministic finite state machines. Theoretical Computer Science 411(16-18), 1770–1784 (2010), https://doi.org/10.1016/j.tcs.2010.01.030

13. Ipate, F., Gheorghe, M.: Finite state based testing of P systems. Natural Computing 8(4), 833 (2009), https://doi.org/10.1007/s11047-008-9099-3
14. Ipate, F., Gheorghe, M.: Testing non-deterministic stream X-machine models and P systems. Electronic Notes in Theoretical Computer Science 227, 113–126 (2009), https://doi.org/10.1016/j.entcs.2008.12.107
15. 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), https://doi.org/10.1016/j.jlap.2010.03.007
16. Lefticaru, R., Gheorghe, M., Ipate, F.: An empirical evaluation of P system testing techniques. Natural Computing 10(1), 151–165 (2011), https://doi.org/10.1007/s11047-010-9188-y
17. Păun, G.: Computing with membranes. Tech. rep., Turku Centre for Computer Science (1998), http://tucs.fi/publications/view/?pub_id=tPaun98a
18. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000), https://doi.org/10.1006/jcss.1999.1693
19. The P systems website. http://ppage.psystems.eu, [Online; accessed 12/05/2018]
20. Verlan, S.: Using the formal framework for P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing. Lecture Notes in Computer Science, vol. 8340, pp. 56–79. Springer Berlin Heidelberg (2014), https://doi.org/10.1007/978-3-642-54239-8_6