Testing Based on Identifiable P Systems Using Cover Automata and X-Machines

Marian Gheorghe[1], Florentin Ipate[2] and Savas Konur[1]

[1] *School of Electrical Engineering and Computer Science*
University of Bradford
Bradford BD7 1DP, UK
{m.gheorghe,s.konur}@bradford.ac.uk

[2] *Department of Computer Science*
Faculty of Mathematics and Computer Science
University of Bucharest
Str. Academiei 14, Bucharest 010014, Romania
florentin.ipate@ifsoft.ro

Abstract

This paper represents a significant advance on the issue of testing for implementations specified by P systems with transformation and communicating rules. Using the X-machine framework and the concept of cover automaton, it devises a testing approach for such systems, that, under well defined conditions, it ensures that the implementation *conforms* to the specification. It also investigates the issue of *identifiability* for P systems, that is an essential prerequisite for testing implementations based on such specifications and establishes a fundamental set of properties for identifiable P systems.

Keywords: P systems, X-machines, cover automata, testing

1. Introduction

Inspired by the structure and functioning of living cells, *membrane computing*, the research field initiated by Gheorghe Păun [25], has been intensively investigated in the last fifteen years. Its main research themes, investigated so far, refer to the computational power of different variants of membrane systems (also *called P systems*), hierarchies of language or multiset classes produced by these devices, their capability to solve hard problems,

decidability and complexity aspects [27]. There have also been significant developments in using the P systems paradigm to model various systems [6]. More recently variants of P systems have been introduced in connection with modelling problems in various areas, e.g., systems and synthetic biology [10], information in biotic systems [29], synchronisation of distributed systems [7], grid algorithms [24], parallel algorithms utilised in 3D graphics [12].

Since P systems have been extensively used in various applications, it is a natural question to ask whether these applications and their implementations are correct and error-free. As *testing* is an essential part of software development and in many cases consumes large percentages of project efforts and budgets (test generation, in particular), it has been recently considered in the context of P systems applications. Some widely used testing approaches, such as mutation testing or transition cover have been considered [18] or adapted (rule coverage [11]) for P systems specifications.

When a formal specification exists, test generation can be automated and, in some cases, it is possible to guarantee that the test suite achieves a certain level of fault detection. The successful application of the test suite to the implementation may even, under some well defined conditions, guarantee that the implementation *conforms* to the specification; this is called *conformance testing*. In particular, conformance testing has been extensively studied for finite state machine specifications and a number of test generation techniques from such specifications exist [22]; these are widely used for testing hardware, embedded systems or network protocols.

As test generation techniques from finite state machine specifications can virtually guarantee (under well defined conditions) the correctness of the implementation, the adaptation and application of such techniques to P systems have been a tempting proposition and some initial ideas were outlined in [17]. The main problem, however, is that finite state machines can only be used to specify the control of a software system; however, non-trivial data is often difficult, even impossible, to specify, using finite state machines. Consequently, a more complex formalism that combines the finite state machines capability to specify the control aspects of a system with suitable data structure is needed.

Such a model is the *X-machine* [8]. Basically, an X-machine is like a finite state machine in which transitions are labelled by processing functions that operate on a data set [8, 14]. This formalism elegantly and effectively combines the control aspects of the system with data structures, while allowing them to be separately described. Conformance testing for X-machines

has been extensively studied and a number of test generation methods have been devised [2, 19, 13]. However, all these have been developed for a particular class of X-machines, called *stream X-machines* [14]. Such X-machines are useful as a specification vehicle for systems whose functionality can be expressed as an input/output transformation, in particular for interactive systems. This is not the case for P systems. In order to use X-machine based techniques for P systems, these techniques would have to be extended to more general classes of X-machines.

In this paper, we present a significant advance on testing implementations based on P systems specifications, by using a general X-machine framework and devising a suitable testing approach. Some essential properties of P systems that make them amenable for this testing approach are identified.

Our contribution has three significant components:

- *A testing method for the general X-machine model.* A non-trivial extension of finite state machine testing based on a particular type of finite state machine called *cover automaton* is developed.

- *An X-machine representation of a P system with transformation and communication rules.* An approximation of the computation of the P system is built using the cover automaton formalism and the previously devised testing method is applied to test an implementation based on a P systems specification. The approach is fairly general, it can be equally applied to cell-like and tissue-like P systems [23] with transformation and communication rules and, under well defined conditions, guarantees the correctness of the implementation under test with regard to the P systems specification.

- *Identifiable P systems.* In order to ensure the correctness of the implementation under test against the P systems specification, the P systems must meet a particular property, called *identifiability*. In this respect, the concept of identifiable P systems is introduced and studied.

These represent major contributions of the paper, as they have not been investigated in the context of P systems research and are essential for testing implementations based on P systems specifications.

The paper is structured as follows. Section 2 provides basic concepts and results: it briefly presents the P system model, finite automata and a variant of the W-method, used for testing from finite cover automata specifications.

Section 3 investigates the issue of identifiability in the context of the P system model. Section 4 presents the X-machine model and the newly developed testing method based on X-machines. Section 5 shows how this method can be applied to test implementations specified by P systems. Finally, conclusions are drawn and further work is outlined in Section 6.

2. Preliminaries

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

2.1. P systems

A basic *cell-like P system* is defined as a hierarchical arrangement of membranes (a *tree structure*) delimiting compartments of the system. The so-called membrane structure resembles the compartmentalisation of biological systems. Each compartment contains a finite multiset of objects and a finite set of rules, which may be empty. Any object (alone or together with other objects) can be transformed into other objects, can diffuse through membranes, and can dissolve the membrane in which it is placed [25]. When the hierarchical structure is replaced by an arbitrary network of compartments (a *graph structure*), we deal with *tissue-like P systems* [23].

One of the most investigated variants of P systems, using either a hierarchical structure of compartments or an arbitrary network of compartments, relies on using transformation and communication rules [26] – generically called processing rules. In this system the multisets of objects from each compartment are processed by the rules associated to it, either transformed into other objects or communicated to neighbouring compartments (those

directly connected to the current one). In some cases an environment is also attached to the system. In this case some compartments can communicate with the environment.

In the sequel we provide the definitions of a cell-like P system with m compartments and of a computation associated with it – for more details related to these concepts we refer to [27]. We will then define the concept of identifiability.

Definition 1. A *cell-like P system* with m compartments is a tuple

$$\Pi = (V, T, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where

- V is a finite set, called *alphabet*.
- T is a subset of V, called the *terminal alphabet*.
- μ is a rooted tree with m nodes, called the *P system structure*; each node, called *compartment*, consists of a unique label denoted by a natural number, $i, 1 \leq i \leq m$, and a multiset over V.
- $w_1, \ldots, w_m \in V^*$ are *multisets of objects* initially present in the m compartments of μ.
- $R_1, \ldots, R_m$ are finite sets of *processing rules*; the rules of R_i are applied in the compartment with label i, $1 \leq i \leq m$.
- $i_0 \in \{1, \ldots, m\}$ is the label of the *output compartment*.

Since now onwards the cell-like P system will be simply called P system. Also a membrane with label i will be called membrane i.

The processing rules of $R_i, 1 \leq i \leq m$, have the form $r : x \to y$, where $x \in V^+$ and $y \in (V \times \{here, out, in\})^*$ and are applied as follows: if z is a multiset from compartment i, then x is removed from z and all objects a that appear as $(a, here)$ in y are added to the multiset from i; all the objects a represented as (a, out) will be sent to the parent of i, and all the objects a represented as (a, in) will be sent to one of the children of i, nondeterministically chosen.

Definition 2. A *configuration* of a P system, Π, with m compartments, is a tuple $c = (c_1, \ldots, c_m)$, where $c_i \in V^*$, $1 \leq i \leq m$, is the multiset from compartment i. The *initial configuration* is $(w_1, \ldots, w_m)$, where $w_i \in V^*$ is the initial multiset of the compartment i, $1 \leq i \leq m$.

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

Definition 3. Given two configurations $c = (c_1, \ldots, c_m)$ and $c' = (c'_1, \ldots, c'_m)$ of a P system, Π, with m compartments, where for any $i, 1 \leq i \leq m$, $u_i \in V^*$, 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, $1 \leq i \leq m$, denoted by $c \Longrightarrow^{(M_1,\ldots,M_m)} c'$, such that for each i, $1 \leq i \leq m$, 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 $(a, here)$ 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 either as (b, out) and i is the parent of j or (b, in) and i is a child of j.

In the theory of P systems, there are various ways of applying the rules, i.e., of selecting the rules of the multisets M_i, $1 \leq i \leq m$, utilised (applied) in any transition. Those referred to in this paper are: *maximal parallelism* (when in any compartment i, $1 \leq i \leq m$, the multiset of rules, M_i, is such that no other multiset containing M_i, which is applicable, exists); *asynchronous* (in any compartment i, $1 \leq i \leq m$, M_i is a multiset consisting of an arbitrary number of rules); *sequential* (in any compartment i, $1 \leq i \leq m$, M_i consists of at most one element). We will denote them by max, $async$ and seq, respectively. When in a transition from c to c' by 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 4. For a P system Π using the transition mode tm, $tm \in \{max, async, seq\}$, by $N_{tm}(\Pi)$ we denote the number of objects from the terminal alphabet, T, appearing in the output compartment in a final configuration.

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

In this paper we will only deal with P 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 P systems does not affect the generality of it due to the fact that there are ways of flattening an arbitrary P system, of the type 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 [9], but similar approaches are also presented in other papers ([28], [1]). Such a P system will be denoted

$$\Pi = (V, T, \mu_1, w_1, R_1, 1),$$

where μ_1 denotes the tree with one node. The rules on the right-hand side will have multisets over V instead of $V \times \{here, out, in\}$ as in the case of one single compartment there is no need to indicate where objects are sent to.

We now introduce the key concept we aim to investigate in this paper, namely *identifiability*. This is introduced in connection with the P system model. The identifiability concept is first introduced for simple rules and then is generalised for multisets of rules.

Definition 5. Two rules $r_1 : x_1 \to y_1$ and $r_2 : x_2 \to y_2$ 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 $r_1 = r_2$.

The rules are not identifiable when the condition from Definition 5 is not satisfied. According to this definition one must have the following computation steps for the rules r_1 and r_2:

$$c = x_1 v_1 \Longrightarrow^{r_1} c' = y_1 v_1; c = x_2 v_2 \Longrightarrow^{r_2} c' = y_2 v_2.$$

Definition 6. The multisets of the 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 $M' = M''$.

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

2.2. Finite automata

In the following two subsections we briefly introduce the basic finite automata concepts and results to be used in the paper. These are largely from [17].

Definition 7. 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*;
- $h : Q \times V \longrightarrow Q$ is the *next-state* function.

As indicated by the above definition, only *deterministic* finite automata will be considered. The next-state (partial) function h can be extended to take sequences in the usual manner, i.e. $h : Q \times V^* \longrightarrow Q$ [8].

Given $q \in Q$, the set L_A^q is defined by $L_A^q = \{s \in V^* \mid h(q, s) \in F\}$. When q is the initial state of A, the set is called the *language accepted (defined) by* A and the simpler notation L_A is used.

A state $q \in Q$ is called *reachable* if there exists $s \in V^*$ such that $h(q_0, s) = q$. A is called *reachable* if all states of A are reachable.

Given $Y \subseteq V^*$, two states $q_1, q_2 \in Q$ are called Y*-equivalent* if $L_A^{q_1} \cap Y = L_A^{q_2} \cap Y$. Otherwise q_1 and q_2 are called Y*-distinguishable*. If $Y = V^*$ then q_1 and q_2 are simply called *equivalent* or *distinguishable*, respectively. Two FAs are called $(Y-)$equivalent or $(Y-)$distinguishable if their initial states are $(Y-)$equivalent or $(Y-)$distinguishable, respectively. A FA A is called *reduced* if every two distinct states of A are distinguishable.

A FA A is called *minimal* if any FA that accepts L_A has at least the same number of states as A. A FA A is minimal if and only if A is reachable and reduced. Furthermore, the minimal FA that accepts the same language as a given FA A is unique (up to a renaming of the state set). These are well-known results; for proofs and other details see for example [15].

2.3. Finite cover automata

A *deterministic finite cover automaton* (*DFCA*) of a finite language U is a FA that accepts all sequences in U and possibly other sequences that are longer than any sequence in U. The concept was introduced by Câmpeanu et al. [3, 4].

Definition 8. 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.

A minimal DFCA for U may have considerably fewer states than the minimal FA that accepts U [16]. Hence, we have the advantage of using a DFCA instead of the precise FA that accepts U.

In the remainder of this subsection we provide the necessary concepts for characterising and constructing a minimal DFCA. These are largely from [3] and [21].

Let $U \subseteq V^*$ be a finite language, l be the length of the longest sequence(s) in U and A be a FA; for simplicity, A is assumed to be reachable. For every state q of A, we define $level(q)$ as the length of the shortest input sequences that reach q, i.e.

$$level(q) = min\{|s| \mid s \in V^*, h(q_0, s) = q\}.$$

Recall that a minimal FA that accepts U is a FA in which all states are reachable and pairwise distinguishable. In a DFCA only sequences of length at most l are considered; thus, every states q_1 and q_2 will have to be distinguished by some input sequence of length at most $l - max\{level(q_1), level(q_2)\}$. If this is the case, we say that q_1 and q_2 are *l-dissimilar*. Unlike state equivalence, similarity is not a transitive relation [3].

Definition 9. Let $A = (V, Q, q_0, F, h)$ be a reachable FA. States q_1 and q_2 are said to be similar, written $q_1 \sim q_2$ if q_1 and q_2 are $V[j]$-equivalent whenever $j = l - max\{level(q_1), level(q_2)\} \geq 0$. Otherwise, q_1 and q_2 are said to be dissimilar, written $q_1 \nsim q_2$.

A minimal DFCA for U can be obtained by decomposing the state set of A based on the similarity criterion.

Definition 10. Let $A = (V, Q, q_0, F, h)$ be a reachable FA. $(Q_i)_{1 \leq i \leq n}$ is called a *state similarity decomposition* (SSD) of Q if

- $\cup_{1 \leq i \leq n} Q_i = Q$;
- for every i and j, $1 \leq i < j \leq n$, $Q_i \cap Q_j = \emptyset$;
- for every i, $1 \leq i \leq n$ and every $q_1, q_2 \in Q_i$, $q_1 \sim q_2$;
- for every i and j, $1 \leq i < j \leq n$, there exist $q_1 \in Q_i$ and $q_2 \in Q_j$ such that $q_1 \nsim q_2$.

For every $q \in Q$ we denote by $[q]$ the set Q_i of the decomposition such that $q \in Q_i$.

In other words, a state similarity decomposition of Q is a partition of Q for which every two elements of the same class are similar and every two distinct classes have at least a pair of dissimilar elements.

Theorem 1. *[21] Let $A = (V, Q, q_0, F, h)$ be a reachable DFCA for U and let $(Q_i)_{1 \leq i \leq n}$ be an SSD of Q. For every i, choose $q_i \in Q_i$ such that $level(q_i) = min\{level(q) \mid q \in Q_i\}$. Define $A' = (V, Q', q_0', F', h')$ by $Q' = \{Q_1, \ldots, Q_n\}$, $q_0' = [q_0]$, $F' = \{[q] \mid q \in F\}$ and $h'(Q_i, a) = [h(q_i, a)]$ for all i, $1 \leq i \leq n$, and $a \in V$. Then A' is a minimal DFCA for U.*

As similarity is not an equivalence relation, the SSD may not be unique and, thus, there may be more than one DFCA for the same finite language U. Several algorithms for constructing a minimal DFCA exist; the best known algorithm [20] requires $O(n \; log \; n)$ time.

2.4. The W-method for testing finite cover automata

In this section we present the W-method for generating test suites from finite cover automata [16].

First, note that the W–method has been developed in the context of Mealy automata; these are finite state machines with outputs and in which all states are terminal. Since the output of these machines is not necessary for our purposes (as it will transpire from Section 4), in the remainder of the paper we will use a form of finite automaton in which all states are terminal, but the next state function may be a partial (rather than total) function. In this case, the words rejected by the automaton will be those for which h is not defined. Such a model can be transformed into an FA that conforms to Definition 7 by adding a new, non-final, ("sink") state that collects all the undefined transitions. However, this addition is not necessary for the testing

theory (the W-method) we will be presenting and so the "sink" state will not be explicitly shown. For simplicity, in what follows, the set of final sets will be omitted from the notation of an automaton (since this coincides with the entire set of states).

In conformance testing we have a formal specification (in our case a FA) and we want 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 [5]), but in this paper we are interested in conformance for bounded sequences. In this case, the problem can be formulated 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. [1]

Suppose the specification A used for test generation is a minimal DFCA for $L_A \cap V[l]$ (if not, this can be minimised using algorithms such that given in [20]) The W–method for bounded sequences, as developed in [16], 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 $|s| = level(q)$.

[1] Naturally, the entire set $V[l]$ can be chosen, but automata theory is used to reduce the size of the test suite.

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:

$$Y_k = SV[k+1](W \cup \{\lambda\}) \cap V[l] \setminus \{\lambda\}.$$

This result is formally proved in [16].

Theorem 2. *[16] For every FA A' which has at most k more states than A, $L_A \cap V[l] = L_{A'} \cap V[l]$ if and only if $L_A \cap Y_k = L_{A'} \cap V_k$.*

3. Identifiable transitions in P systems

In this section we investigate the property of identifiability for P system rules and multisets that will prove to be essential for testing these systems. We establish the necessary and sufficient conditions for rule distinguishability and we show how to construct P systems for which any pair of distinct multisets is distinguishable.

We start by introducing a notation utilised in this section.

Notation 1. Given a multiset $M = r_1^{n_1} \dots r_k^{n_k}$, where $r_i : x_i \to y_i$, $1 \leq i \leq k$, we denote by r_M the rule $x_1^{n_1} \dots x_k^{n_k} \to y_1^{n_1} \dots y_k^{n_k}$, 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 1. For any two rules $r_i : x_i \to y_i$, $1 \leq i \leq 2$, when we check whether they are identifiable or not one can write them as $r_i : uv_i \to wz_i$, $1 \leq i \leq 2$, 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. Obviously, when $v_1 v_2 = \lambda$ and $z_1 z_2 = \lambda$ then the rules are not identifiable, as they represent the same rule, $u \to w$.

We first show that the identifiability of two rules does not depend on the configurations in which they are applicable. For the two rules introduced in Remark 1 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.

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

Proof. Applying the two identifiable rules, r_1 and r_2, to the configuration c, one gets c' and c'' and $c' \neq c''$. If the rules are applicable to another configuration c_1 bigger than c, i.e, $c_1 = ct$, then the results are $c'_1 = c't$ and $c''_1 = c''t$ and obviously $c'_1 \neq c''_1$, hence r_1 and r_2 are identifiable in c_1. □

Corollary 4. *Two rules r_1 and r_2 identifiable in c_{r_1,r_2} are identifiable in any configuration in which they are applicable.*

Proof. The result is an immediate consequence of Lemma 3 and Remark 1. □

One can formulate a similar result for two multisets of rules.

Corollary 5. *Two multisets of rules M_1 and M_2 identifiable in $c_{r_{M_1},r_{M_2}}$ are identifiable in any configuration in which they are applicable.*

Proof. The result is an immediate consequence of Corollary 4 and Notation 1. □

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.

We make some comments regarding the constraint imposed on the rules to be *applicable* as this has different meanings depending on the transition mode.

Remark 2. For the *async* transition mode two multisets of rules (and two rules) applicable in a configuration are also applicable in any other bigger configuration. 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 $r_1 : a \to a; r_2 : ab \to abb; r_3 : bb \to c$ 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 $r_1 r_3$ applicable.

Remark 3. 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.

We now provide a characterisation of the two rules to be (non) identifiable.

Theorem 6. *The rules $r_1 : x_1 \to y_1$ and $r_2 : x_2 \to y_2$, are not identifiable if and only if they have the form $r_1 : uv_1 \to wv_1$ and $r_2 : uv_2 \to wv_2$ and for any $a \in V$, a appears in at most one of v_1 or v_2.*

Proof. Let us start with this implication "$\Longrightarrow$". As we have already discussed one can use the rules as $r_1 : uv_1 \to y_1$ and $r_2 : uv_2 \to y_2$ and for any $a \in V$, a appears in at most one of v_1 or v_2; and one can consider the smallest configuration where they are applicable, $c_{r_1,r_2} = uv_1v_2$. Applying these rules to this configuration, the following computations are obtained:

$$c_{r_1,r_2} \Longrightarrow^{r_1} y_1v_2; c_{r_1,r_2} \Longrightarrow^{r_2} y_2v_1.$$

As these rules are not identifiable it turns out that the results of the two computations are the same, i.e., $y_1v_2 = y_2v_1$. Given that for any $a \in V$, a appears in at most one of v_1 or v_2, it follows that y_1 contains v_1 and y_2 contains v_2, i.e., $y_1 = w_1v_1$ and $y_2 = w_2v_2$. From the equality of the results of the computations it follows that $w_1 = w_2 = w$ and this proves the result.

Let us consider the opposite "$\Longleftarrow$". In this case the rules are $r_1 : uv_1 \to wv_1$, $r_2 : uv_2 \to wv_2$ and for any $a \in V$, a appears in at most one of v_1 or v_2. We consider again the smallest configuration $c_{r_1,r_2} = uv_1v_2$ and apply the two rules; then one can obtain:

$$c_{r_1,r_2} \Longrightarrow^{r_1} wv_1v_2; c_{r_1,r_2} \Longrightarrow^{r_2} v_1wv_2.$$

Hence, r_1 and r_2 are not identifiable.

The above proof assumes that v_1 and v_2 are not empty multisets. The result remains true when one of them or both are empty. In the latter case we have the same rule, which might have the right-hand side λ. □

Based on the result provided by Theorem 6 one can state when two rules are identifiable.

Corollary 7. *The rules $r_1 : uv_1 \to wz_1$ and $r_2 : uv_2 \to wz_2$, such that for any $a \in V$, a appears in at most one of v_1 or v_2, are identifiable if and only if $v_1 \neq z_1$ or $v_2 \neq z_2$.*

One can formulate similar results regarding the identifiability of the multisets of rules by referring to the associated rules as introduced by Notation 1.

With the results obtained so far one can determine, for a P system, whether any two rules are identifiable or not. In various transition modes utilised in P systems – maximal parallelism or asynchronous mode – in any computation step either single rules or multisets of rules are involved. It is then important to determine whether the identifiability of single rules can be lifted to multisets of rules. More precisely, we want to know whether it is true that the identifiability of any pair of simple rules is inherited by the multisets of rules. Unfortunately, this is not true in general, as it is shown by the next example.

Example 1. Let us consider a P system with the following four rules: $r_1 : a \to b$, $r_2 : b \to a$, $r_3 : c \to d$, $r_4 : d \to c$. According to Corollary 7, any two rules are identifiable, but $M_1 = r_1 r_2$ and $M_2 = r_3 r_4$ are not, as $r_{M_1} : ab \to ab$ and $r_{M_2} : cd \to cd$ are identity rules and according to Theorem 6 they are not identifiable.

However, one can show that some particular multisets of rules are identifiable when their components are. More precisely, we have the following result.

Theorem 8. *If r_1 and r_2 are identifiable then r_1^n and r_2^n are identifiable, for any $n \geq 1$.*

Proof. According to Corollary 7 the rules can be written $r_1 : uv_1 \to wz_1$ and $r_2 : uv_2 \to wz_2$, such that for any $a \in V$, a appears in at most one of v_1 or v_2, and $v_1 \neq z_1$ or $v_2 \neq z_2$. This implies that $v_1^n \neq z_1^n$ or $v_2^n \neq z_2^n$, for any $n \geq 1$, i.e., r_1^n and r_2^n are identifiable. □

One can ask whether the more general case of r_1 and r_2 being identifiable leads to r_1^n and r_2^m (with arbitrary $n, m \geq 1$) being identifiable. Unfortunately, this does not prove to be true. Let us consider the identifiable rules

$r_1 : a \to b$ and $r_2 : aa \to bb$; the multisets r_1^2 and r_2 are not identifiable as they are identical.

One can show that identifiability of any two multisets of rules can be achieved in some special circumstances. Assume for a P system $\Pi = (V, T, \mu_1, w_1, R_1, 1)$, as introduced in Definition 1, we build the following P system

$$\Pi_L = (V', T, \mu_1, w_1, R_1', 1)$$

extending each rule on its right-hand side with its label – hence the index L associated with the P system; more precisely, if $Lab(R_1) = \{r \mid r : x \to y \in R_1\}$ then

- $V' = V \cup Lab(R_1)$; and
- $R_1' = \{r' : x \to yr \mid r : x \to y \in R_1\}$.

Also for any multiset of rules M we will denote by $Lab(M)$ the multiset of labels of rules occurring in M.

Theorem 9. *For any P system Π, the P system Π_L is such that (i) $N_{tm}(\Pi) = N_{tm}(\Pi_L)$ and (ii) any two multisets of rules are identifiable, for any of the transition mode tm, $tm \in \{max, async, seq\}$.*

Proof. First in order to show that Π and Π_L are equivalent for any transition mode tm, $tm \in \{max, async, seq\}$, one can observe that for any terminal computation in Π

$$u_0 = w_1 \Longrightarrow_{tm}^{M_1} u_1 \ldots u_{n-1} \Longrightarrow_{tm}^{M_n} u_n$$

there is a terminal computation in Π_L and vice versa

$$u_0' = w_1 \Longrightarrow_{tm}^{M_1'} u_1' = u_1 Lab(M_1) \ldots u_{n-1}' = u_{n-1} Lab(M_1) \ldots Lab(M_{n-1})$$

$$\Longrightarrow_{tm}^{M_n'} u_n' = u_n Lab(M_1) \ldots Lab(M_{n-1}) Lab(M_n),$$

where M_i', $1 \leq i \leq n$, is obtained from M_i, $1 \leq i \leq n$, by replacing each rule $r \in R_1$ by its corresponding $r' \in R_1'$.

Obviously, $|u_n|_T = |u_n'|_T$, which proves the equivalence of the two devices.

Let us consider two multisets of rules, M_1 and M_2, applicable to a configuration in given transition mode. One can use Notation 1 to obtain the

rules, r_{M_1} and r_{M_2}, associated with the multisets of rules, and Remark 1 for providing the following format of them:

$$r_{M_i} : uv_i \to wz_iLab(M_i), 1 \leq i \leq 2.$$

Obviously, $v_i \neq z_iLab(M_i)$, $1 \leq i \leq 2$, and according to Corollary 7 the rules are identifiable and consequently the multisets of rules, M_1 and M_2. □

This result is useful as it shows how one can get P systems with all applicable multisets of rules being identifiable. The downside is that we have to slightly change the rules and collect all the labels, and these increase the size of all the intermediary results. One can partially alleviate some of these constraints, but only in certain circumstances, as it is shown below. Before providing the result one more notation will be introduced. For a P system $\Pi = (V, T, \mu_1, w_1, R_1, 1)$, as introduced in Definition 1, we build the following P system

$$\Pi_{L,\lambda} = (V', T, \mu_1, w_1, R_1'', 1)$$

where V' is defined as for Π_L and R_1'' is obtained based on R_1' by adding label erasing rules, i.e.,

$$R_1'' = R_1' \cup \{r'' : r \to \lambda \mid r : x \to yr \in R_1'\}.$$

With this newly introduced P system we aim to remove the labels generated in the previous steps, hence one can reduce the size of the intermediary results.

Theorem 10. *For any P system* Π*, the P system* $\Pi_{L,\lambda}$ *is such that (i)* $N_{tm}(\Pi) = N_{tm}(\Pi_{L,\lambda})$ *and (ii) any two multisets of rules are identifiable, for any of the transition mode tm,* $tm \in \{max, seq\}$*.*

Proof. In order to prove the equivalence $N_{tm}(\Pi) = N_{tm}(\Pi_{L,\lambda})$, $tm \in \{max, seq\}$, we consider a terminal computation of length n in Π, similar to the one provided in the proof of Theorem 9.

In the case of $tm = max$, the corresponding terminal computation in $\Pi_{L,\lambda}$ consists of $n + 1$ steps: the first is identical with the first step in Π, but is using corresponding rules from R_1'; the rest of the n steps in $\Pi_{L,\lambda}$ use both rules from R_1' and rules $r'' : r \to \lambda$ for all the labels r appearing in each configuration; in the last step, $n+1$, only label erasing rules are utilised and the final configuration is the same in both P systems.

When $tm = seq$ then in each step of a terminal computation in Π only one single rule is used. If we assume its length is n then the corresponding terminal computation in $\Pi_{L,\lambda}$ will consists of $2n$ steps and both will lead to the same configuration.

These prove the equivalence of Π and $\Pi_{L,\lambda}$ for $tm \in \{max, seq\}$.

For proving the identifiability of any two applicable multisets of rules we distinguish between the two transition modes considered. Let us start with $tm = max$ and two multisets M_1 and M_2. They both can consist of a multiset over R_1' and one over $\{r'' : r \to \lambda \mid r : x \to yr \in R_1'\}$. Let us denote them by $M_{i,V}$ and $M_{i,Lab}$, respectively, for $i = 1, 2$. Given that both M_1 and M_2 are applicable in the max mode then it follows that $M_{1,Lab} = M_{2,Lab}$. So, it will be enough just to show that $M_{1,V}$ and $M_{2,V}$ are identifiable and this follows from the argument used in the proof of Theorem 9 for the identifiability of the two multisets of rules.

In the case of $tm = seq$ the two multisets of rules are in fact simple rules, as in each step only a single rule is applied. The two considered rules, r_1 and r_2 can be (i) both from R_1' or $\{r'' : r \to \lambda \mid r : x \to yr \in R_1'\}$; or (ii) one from R_1' and one from $\{r'' : r \to \lambda \mid r : x \to yr \in R_1'\}$. It is obvious that any two rules from R_1'' are identifiable, according to Corollary 7, and consequently the rules above. □

The construction utilised in Theorem 10 does not work in the case of the asynchronous transition mode. Indeed, let us consider a P system with a set of rules including $r_1 : a \to a$ and $r_2 : b \to b$. These are not identifiable according to Theorem 6. If we build the rules from R_1'', $r_1' : a \to ar_1$, $r_1'' : r_1 \to \lambda$, $r_2' : b \to br_2$, $r_2'' : r_2 \to \lambda$, and consider the configuration abr_1, then according to the *async* mode both r_2' and $r_1'r_1''r_2'$ applicable, but are not identifiable ($r_1'r_1''$ is the same as r_1, i.e., $a \to a$).

4. X-machine based testing

An X-machine is a finite automaton in which transitions are labelled by partial functions on a data set X instead of mere symbols [8]. In many previous publications, a particular class, called *stream X-machines*, in which X is composed of input and output streams and an internal storage called memory, is considered, but in this paper we use the general model.

Definition 13. An *X-Machine* (abbreviated *XM*) is a tuple

$$Z = (Q, X, \Phi, H, q_0, x_0),$$

where:

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

It is sometimes helpful to think of an X-machine as a finite automaton with the arcs labelled by functions from the set Φ. The set Φ is often called the *type* of Z. The automaton $A_Z = (\Phi, Q, H, q_0)$ over the alphabet Φ is called *the associated finite automaton* (abbreviated *associated 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.

We now turn our attention to test generation from an X-machine. 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 Φ and initial data value x_0 as the specification.

The idea 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. We will then need a mechanism that translates sequences of processing functions into sequences of data values. This will be called a *test transformation* of Z.

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

In general, consider a sequence of processing functions, $p = \phi_1 \dots \phi_n$. First suppose that p is controllable. If p is contained in the language defined by A_Z then the second rule of the above definition will be applied for every processing function in the sequence, so $t(p)$ will be the associated computation $x_0 \dots x_n$ of Z. Otherwise, the second rule will be applied until a prefix $\phi_1 \dots \phi_{k+1}$ of p is found that is not in the language defined by A_Z, so $t(\phi_1 \dots \phi_k) = x_0 \dots x_k$, where $\phi_1 \dots \phi_k$ denotes the longest prefix of p that is in L_{A_Z}. Then the second rule will be applied once more, so $t(\phi_1 \dots \phi_{k+1}) = x_0 \dots x_{k+1}$. For the remaining processing functions $\phi_{k+2}, \dots, \phi_n$, the third rule will be applied and so $t(p) = t(\phi_1 \dots \phi_{k+1}) = x_0 \dots x_{k+1}$. Finally, if p is not controllable, rules (4) and (5) ensure that $t(p)$ is undefined. All these are summarised by the following lemma.

Lemma 11. *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 k, $0 \leq k \leq n-1$, 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.*

Proof. We prove the results by induction on j, $1 \leq j \leq n$.

We prove the results for $j = 1$. By rule (1) of Definition 16, $t(\lambda) = x_0$. Suppose $\phi(x_0) = x_1$. If $\phi_1 \in L_{A_Z}$, then by rule (2) of Definition 16, $t(\phi_1) = x_0x_1$; else, by rule (3) $t(\phi_1) = x_0$. If $x_0 \notin dom\ \phi_1$ then, by rule (4), $t(\phi_1)$ is undefined.

Assume the results hold for j, $1 \leq j \leq n-1$. Suppose $\phi_1 \dots \phi_{j+1}$ is controllable and let $\phi_i(x_{i-1}) = x_i$, $1 \leq i \leq j$. If $p \in L_{A_Z}$ then, by the induction hypothesis, $t(\phi_1 \dots \phi_j) = x_0 \dots x_j$ and, by rule (2) $t(\phi_1 \dots \phi_{j+1}) = x_0 \dots x_{j+1}$. If $p \notin L_{A_Z}$ let k, $0 \leq k \leq j-1$, be such that $\phi_1 \dots \phi_k \in L_{A_Z}$ and $\phi_1 \dots \phi_k\phi_{k+1} \notin L_{A_Z}$. By the induction hypothesis $t(\phi_1 \dots \phi_j) = x_1 \dots x_{k+1}$ and so, by rule (3), $t(\phi_1 \dots \phi_{j+1}) = x_0 \dots x_{k+1}$. If $\phi_1 \dots \phi_{j+1}$ is not controllable then either $\phi_1 \dots \phi_j$ is not controllable or $\phi_1 \dots \phi_j$ is controllable and $x_n \notin dom\ \phi_{j+1}$. If $\phi_1 \dots \phi_j$ is not controllable then, by the induction hypothesis, $t(\phi_1 \dots \phi_j)$ is not defined and so, by rule (5), $t(\phi_1 \dots \phi_{j+1})$ is not defined. If $\phi_1 \dots \phi_j$ is controllable and $x_n \notin dom\ \phi_{j+1}$, then, by rule (4), $t(\phi_1 \dots \phi_{j+1})$ 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. Informally, Φ is identifiable if, given any two distinct processing functions ϕ_1 and ϕ_2 and any data value x, the two functions cannot produce the same data value x' when applied in x.

Definition 17. Φ 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 Φ 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 12. *Let Z and Z' be XMs with type Φ. Suppose Φ 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 $p \in L_{A_{Z'}}$.*

Proof. "$\Longrightarrow$": Suppose $p \in L_{A_Z}$. Since p is controllable then $t(p) = x_0 \dots x_n$ such that $\phi_i(x_{i-1}) = x_i$, $1 \leq i \leq n$. $x_1 \dots x_n$ is a computation of Z and so $x_0 \dots x_n$ is a computation of Z'. Then there exist $\phi'_1, \dots, \phi'_n \in \Phi$ such that

$\phi'_1 \ldots \phi'_n \in L_{A_{Z'}}$ and $\phi'_i(x_{i-1}) = x_i$, $1 \leq i \leq n$. Since Φ is identifiable it follows that $\phi'_i = \phi_i$, $1 \leq i \leq n$.

"$\Longleftarrow$": Suppose $p \notin L_{A_Z}$. Let k, $0 \leq k \leq n-1$ be such that $\phi_1 \ldots \phi_k \in L_{A_Z}$ and $\phi_1 \ldots \phi_k \phi_{k+1} \notin L_{A_Z}$. Then $t(p) = x_0 \ldots x_{k+1}$ such that $\phi_i(x_{i-1}) = x_i$, $1 \leq i \leq k+1$. We prove by contradiction that $\phi_1 \ldots \phi_{k+1} \notin L_{A_{Z'}}$. Assume $\phi_1 \ldots \phi_{k+1} \in L_{A_{Z'}}$. Then $x_1 \ldots x_{k+1}$ is a computation of Z' and so $x_1 \ldots x_{k+1}$ is a computation of Z. Then there exist $\phi'_1, \ldots, \phi'_{k+1} \in \Phi$ such that $\phi'_1 \ldots \phi'_{k+1} \in L_{A_Z}$ and $\phi'_i(x_{i-1}) = x_i$, $1 \leq i \leq k+1$. Since Φ is identifiable it follows that $\phi'_i = \phi_i$, $1 \leq i \leq n$. Then $\phi_1 \ldots \phi_{k+1} \in L_{A_Z}$, which provides a contradiction as required. □

As for the case of finite automata discussed in Section 2.4, we are interested to show that the implementation Z' conforms to the specification Z for sequences of length up to a given upper bound l. A test suite that will establish this property will be called an l-bounded conformance test suite.

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 13. *Let Z be an XM with type Φ, data set X and initial data value x_0. Suppose Φ is identifiable and $L_{A_Z} \cap \Phi[l]$ 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.*

Proof. Let $Z' \in C$. Assume $t(P) \cap Comp(Z) = t(P) \cap Comp(Z')$. Let $p \in P$. If p is controllable, by Lemma 12, $p \in L_{A_Z}$ if and only if $p \in L_{A_{Z'}}$. If p is not controllable, $p \notin L_{A_Z}$ and $p \notin L_{A_{Z'}}$. As p is arbitrarily chosen, we have $P \cap L_{A_Z} = P \cap L_{A_{Z'}}$. Then $L_{A_Z} \cap \Phi[l] = L_{A_{Z'}} \cap \Phi[l]$. Hence, by Definition 14, it follows that $Comp(Z) \cap X[l] = Comp(Z') \cap X[l]$. □

We can now apply the results presented in Section 2.4 to produce test suites from an X-machine specification Z. Let $l > 0$ be a predefined upper bound. We assume that Φ is identifiable and $L_{A_Z} \cap \Phi[l]$ is controllable (we will discuss the satisfiability of these requirements in the next section, in the context of the XM model of a P system). 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 as discussed earlier[2]). Suppose the fault model C is the set of X-machines Z' with the same data set X, type Φ 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

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

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.

5. Modelling and testing P systems using X-machines

In order to generate test suites for a P system using the method presented above, an appropriate X-machine model needs to be defined first. As discussed in Section 2.1, multi-membrane P systems can be collapsed into one membrane systems [9]. In the following we consider the case of a one membrane P system $\Pi = (V, T, \mu_1, w_1, R_1, 1)$. A configuration of Π can be changed as a result of the application of one or more rules, in parallel. The idea is to express the computation tree of Φ as an X-machine $Z^t = (Q^t, X, \Phi, H^t, q_0^t, x_0)$. In order to keep the computation tree of the P system Π finite, only computations of maximum l steps will be considered, with $l > 0$ a predefined integer. Let $R_1 = \{r_1, \ldots, r_n\}$ be the set of rules of Π. 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;

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

- 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;
- Φ is the set of (partial) functions induced by the application of multisets $r_1^{i_1} \dots r_n^{i_n}$, $0 \leq i_1 \leq N_1, \dots, 0 \leq i_n \leq N_n$, $i_1 + \dots i_n > 0$;
- H^t is the next-state function determined by the computation tree.

Remark 4. Note that, by definition, L_{A_Z} is controllable.

Remark 5. The set of (partial) functions, Φ, 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 6).

Example 2. Let us consider one compartment P system $\Pi_1 = (V, V, \mu_1, w_1, R_1, 1)$, where $V = \{a, b, c\}$, $w_1 = ab$, $R_1 = \{r_1 : a \to b, r_2 : ab \to bc, r_3 : c \to b, r_4 : c \to cc\}$. 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 Π_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 $b^2c^2 \Longrightarrow^{r_4^2} bc^4$; $bc^4 \Longrightarrow^{r_3^4} b^5$, $bc^4 \Longrightarrow^{r_3^3 r_4} b^4c^2$, $bc^4 \Longrightarrow^{r_3^2 r_4^2} b^3c^4$, $bc^4 \Longrightarrow^{r_3 r_4^3} b^2c^6$ and $bc^4 \Longrightarrow^{r_4^4} bc^8$. The configurations produced by these eight rules represent the fourth level of the tree.

One can observe that any two of the above multisets of rules are identifiable, according to Corollary 7, 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 $l = 4$. It can be observed that 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 Φ of the X-machine Z^t

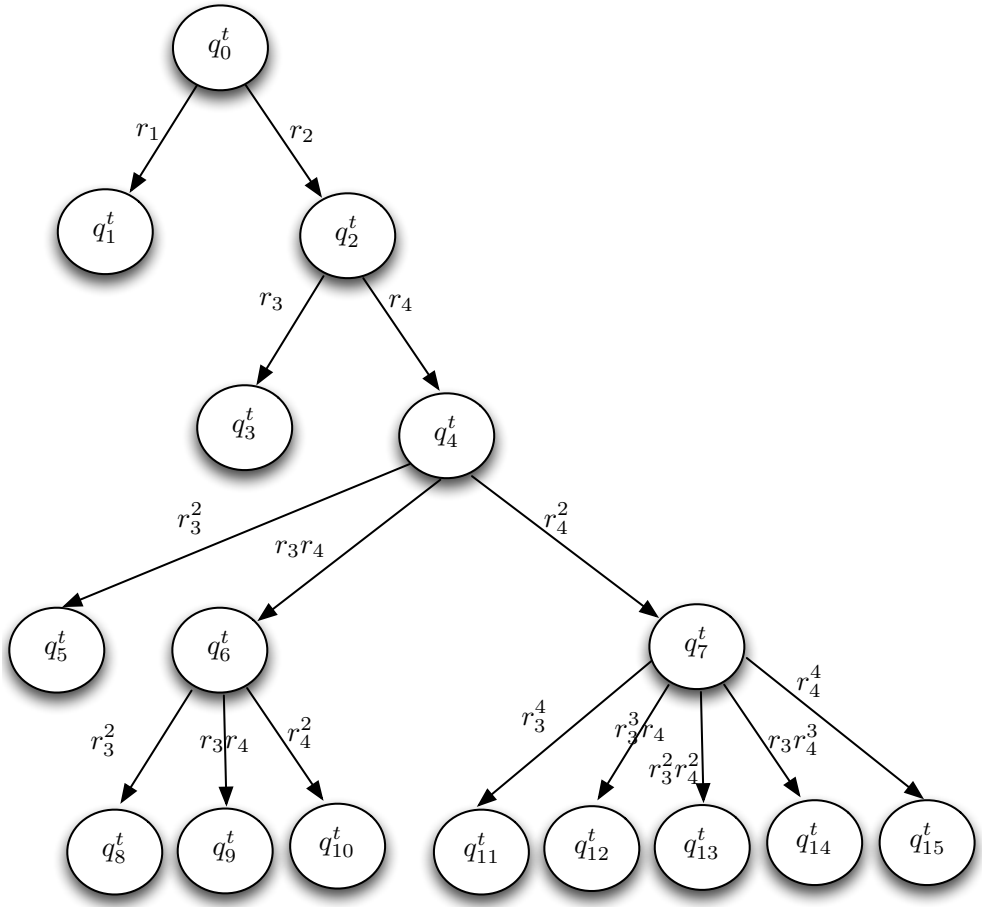

Figure 1: The associated automaton A_{Z^t} corresponding to the computation tree for Π_1 and $l = 4$

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 4, an X-machine Z whose associated automaton A_Z is a DFCA for $L_{A_{Z^t}}$ needs to be constructed first. In what follows we show how Theorem 1 can be used to define such an X-machine. 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.

Theorem 14. *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 $L_{A_{Z^t}}$.*

Proof. Let $q \in P^t$, $q_1, q_2 \in [q]$. Since $q_1 \sim q$, $q_2 \sim q$, $level(q) \leq level(q_1)$ and $level(q) \leq level(q_2)$, $q_1 \sim q_2$ [16, 17]. Thus, since the other conditions of Definition 10 follow directly from the definition of $[q]$, $([q])_{q \in P^t}$ is an SSD of Q^t. Then, by Theorem 1, A_Z is a minimal DFCA for $L_{A_{Z^t}}$. □

Example 3. 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).
We illustrate this step for Z as in Example 3. As λ, r_1, r_2, $r_2\ r_4$, $r_2\ r_4\ r_4^2$ are the sequences of minimum length that reach $[q_0^t], [q_1^t], [q_2^t], [q_4^t]$ and $[q_7^t]$, respectively, $S = \{r_1, r_2, r_2\ r_4, r_2\ r_4\ r_4^2\}$ 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. (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 followed by the application of rule r' in the following step; the second notation is also used for multisets of rules.)

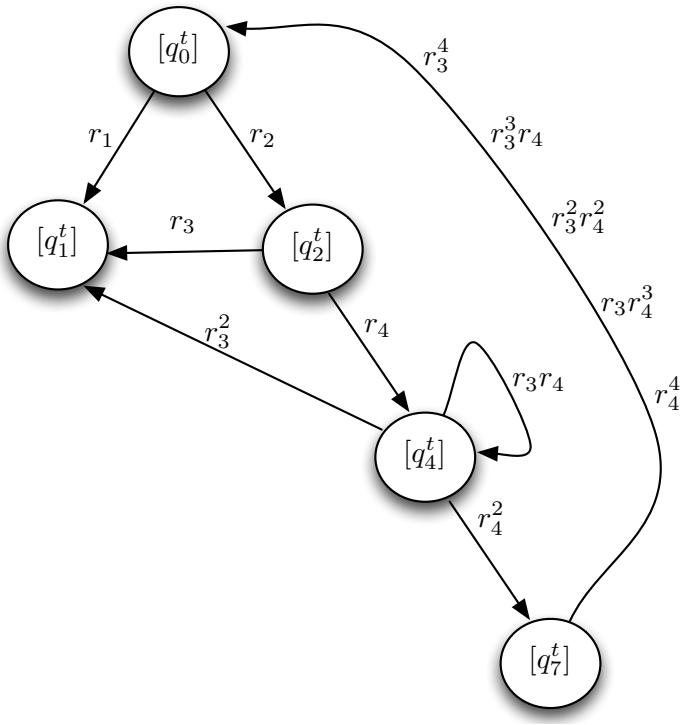

Figure 2: The DFCA for $L_{A_{Z^t}}$

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 19. 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.

Example 4. Consider again the P system Π_1 as in Example 2. 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 Π_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 [5], 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 Φ. 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.

Example 5. We illustrate the construction of the test transformation t with an example. Consider again Φ_1 as in Example 2 and Z as in Example 3. The rule application mode is maximal parallelism for Φ 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.

Note that the test suite may contain both positive and negative test sequences. A *positive* test sequence is a sequence of configurations that is a computation of Z, whereas a negative test sequence is a sequence of configurations that is not a computation of Z. A successful application of a

positive test sequence on the IUT will produce a computation of the IUT, whereas the successful application of a negative test sequence will produce no corresponding computation on the IUT.

Note that the application of the previously constructed test suite is guaranteed (when successfully applied to the implementation under test) to ensure that the IUT conforms to the specification (for sequences up to length l) if the processing functions Φ are identifiable.

6. Conclusions

In this paper we have presented a testing approach for P systems that, under well defined conditions, ensures that the implementation conforms to the specification. We have also investigated the concept of identifiable P systems, which is essential for testing these systems. This concept has been discussed for one-compartment systems, but can be naturally extended for an arbitrary P system with m compartments.

Indeed, the two distinct multisets of rules $M' = (M'_1, \ldots, M'_m)$ and $M'' = (M''_1, \ldots, M''_m)$ are identifiable, if there is a configuration $c = (c_1, \ldots, c_m)$ where M' and M'' are applicable and if $c \Longrightarrow^{M'} c'$ and $c \Longrightarrow^{M''} c'$ then $M' = M''$. One can notice that M' and M'' might have only certain components distinct in order to be distinct multisets of rules. One can also introduce the concept of *strong identifiability* which means that any two components of the two multisets of rules are identifiable. This concept will be considered in our future work together with specific classes of P systems using various types of rules and constraints in applying them.

Acknowledgements. The authors would like to thank the anonymous reviewers for their comments and suggestions that have contributed to the improvement of the paper. Marian Gheorghe and Savas Konur acknowledge the support from EPSRC (EP/I031812/1). Marian Gheorghe's and Florentin Ipate's work is partially supported by CNCS-UEFISCDI (PN-II-ID-PCE-2011-3-0688).

[1] Agrigoroaiei, O., Ciobanu, G., 2010. Flattening the transition P systems with dissolution. In: Proceedings of the 11th International Conference on Membrane Computing. CMC'10. Springer-Verlag, pp. 53–64.

[2] Bogdanov, K., Holcombe, M., Ipate, F., Seed, L., Vanak, S., 2006. Testing methods for X-machines: A review. Form. Asp. Comput. 18 (1), 3–30.

[3] Câmpeanu, C., Sântean, N., Yu, S., 1999. Minimal cover-automata for finite languages. In: Revised Papers from the Third International Workshop on Automata Implementation. WIA '98. Springer-Verlag, pp. 43–56.

[4] Câmpeanu, C., Sântean, N., Yu, S., 2001. Minimal cover-automata for finite languages. Theor. Comput. Sci. 267 (1-2), 3–16.

[5] Chow, T. S., 1978. Testing software design modeled by finite-state machines. IEEE Trans. Softw. Eng. 4 (3), 178–187.

[6] Ciobanu, G., Pérez-Jiménez, M. J., Păun, Gh., 2005. Applications of Membrane Computing (Natural Computing Series). Springer-Verlag.

[7] Dinneen, M. J., Yun-Bum, K., Nicolescu, R., 2012. Faster synchronization in P systems. Natural Computing 11 (4), 637–651.

[8] Eilenberg, S., 1974. Automata, Languages, and Machines. Academic Press, Inc.

[9] Freund, R., Leporati, A., Mauri, G., Porreca, A. E., Verlan, S., Zandron, C., 2014. Flattening in (tissue) P systems. In: Revised Selected Papers of the 14th International Conference on Membrane Computing - Volume 8340. CMC 2013. Springer-Verlag, pp. 173–188.

[10] Frisco, P., Gheorghe, M., Pérez-Jiménez, M. J. (Eds.), 2014. Applications of Membrane Computing in Systems and Synthetic Biology. Springer-Verlag; Emergence, Complexity and Computation Series, Vol. 7.

[11] Gheorghe, M., Ipate, F., 2009. On testing P systems. In: Membrane Computing. Springer-Verlag, pp. 204–216.

[12] Gimel'farb, G. L., Nicolescu, R., Ragavan, S., 2013. P system implementation of dynamic programming stereo. Journal of Mathematical Imaging and Vision 47 (1–2), 13–26.

[13] Hierons, R. M., 2010. Checking experiments for stream X-machines. Theor. Comput. Sci. 411 (37), 3372–3385.

[14] Holcombe, M., Ipate, F., 1998. Correct Systems. Building a Business Process Solution. Springer, Applied Computing Series.

[15] Hopcroft, J. E., Motwani, R., Ullman, J. D., 2006. Introduction to Automata Theory, Languages, and Computation (3rd Edition). Addison-Wesley Longman Publishing Co., Inc.

[16] Ipate, F., 2010. Bounded sequence testing from deterministic finite state machines. Theor. Comput. Sci. 411 (16-18), 1770–1784.

[17] Ipate, F., Gheorghe, M., 2009. Finite state based testing of P systems. Natural Computing 8 (4), 833–846.

[18] Ipate, F., Gheorghe, M., 2009. Mutation based testing of P systems. International Journal of Computers, Communication and Control 4 (3), 253–262.

[19] Ipate, F., Holcombe, M., 2008. Testing data processing-oriented systems from stream X-machine models. Theor. Comput. Sci. 403 (2-3), 176–191.

[20] Körner, H., 2003. On minimizing cover automata for finite languages in O(n log n) time. In: Proceedings of the 7th International Conference on Implementation and Application of Automata. CIAA'02. Springer-Verlag, pp. 117–127.

[21] Körner, H., 2003. A time and space efficient algorithm for minimizing cover automata for finite languages. Int. J. Found. Comput. Sci. 14 (06), 1071–1086.

[22] Lee, D., Yannakakis, M., 1996. Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE 84 (8), 1090–1123.

[23] Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A., 2003. Tissue P systems. Theor. Comput. Sci. 296 (2), 295–326.

[24] Nicolescu, R., 2014. Structured grid algorithms modelled with complex objects. In: Revised Selected Papers of the 16th International Conference on Membrane Computing - Volume 8340. CMC 2013. Springer-Verlag, pp. 56–79.

[25] Păun, Gh., 2000. Computing with membranes. Journal of Computer and System Sciences 61 (1), 108–143.

[26] Păun, Gh., 2002. Membrane Computing: An Introduction. Springer-Verlag.

[27] Păun, Gh., Rozenberg, G., Salomaa, A., 2010. The Oxford Handbook of Membrane Computing. Oxford University Press, Inc.

[28] Verlan, S., 2015. Using the formal framework for P systems. In: Revised Selected Papers of the 14th International Conference on Membrane Computing - Volume 9504. CMC 2015. Springer-Verlag, pp. 321–337.

[29] Vincenzo, M., 2013. Infobiotics. Springer-Verlag; Emergence, Complexity and Computation Series, Vol. 3.