

# An empirical evaluation of P system testing techniques

Raluca Lefticaru · Marian Gheorghe · Florentin Ipate

Published online: 28 March 2010  
© Springer Science+Business Media B.V. 2010

**Abstract** This paper presents the existing techniques for P system testing and performs an empirical evaluation of their fault-detection efficiency. The comparison is performed using mutation testing and, based on the results obtained, some improved testing methodologies are proposed.

**Keywords** P system testing · Coverage criteria · FSM-based testing · Mutation testing

## 1 Introduction

*Membrane computing* represents a branch of *natural computing* (Kari and Rozenberg 2008), and the basic models were introduced by Gheorghe Păun in 1998, in a seminal research report (Păun 1998), further published as a journal paper (Păun 2000). The computational model, called *P system*, is inspired by the structure of the living cells and the way they process chemical substances in different compartments. The key element of the model is the *membrane*, which structures the system into compartments arranged in accordance with a certain topology. The membranes define compartments (or regions). These regions can contain multisets of objects and other regions that evolve according to given rules, which are applied in a nondeterministic, maximally parallel manner.

---

R. Lefticaru (✉) · M. Gheorghe · F. Ipate  
Department of Computer Science, University of Pitesti, Str Targu din Vale 1,  
110040 Pitesti, Romania  
e-mail: raluca.lefticaru@gmail.com

F. Ipate  
e-mail: florentin.ipate@ifsoft.ro

M. Gheorghe  
Department of Computer Science, University of Sheffield, Regent Court,  
Portobello Street, Sheffield S1 4DP, UK  
e-mail: M.Gheorghe@dcs.shef.ac.uk

Many variants of P systems have been proposed and investigated in this *fast breaking area of computer science* (The P Systems Web Site 2009): cell P systems, tissue P systems, neural P systems, population P systems, etc. The differences between them are given by their structure, the ways this structure is changing, the format of the rules applied and, ultimately, by the type of computation they perform (for more details, see Păun 2002; Păun et al. 2009).

Many classes of P systems have been introduced and studied, many of them being computationally universal. Other aspects of these systems have been investigated: complexity issues, hierarchies of families of sets of numbers or languages associated with them, formal semantics, the power of solving NP-complete problems. A series of applications of P systems in biology, linguistics, computer science, economy have been developed (Păun 2002; Păun and Rozenberg 2002; Păun et al. 2009). In these circumstances, the need to formally verify properties of such systems as well as of testing their implementations emerged in the last years (Gheorghe and Ipate 2008).

A recent paper (Gheorghe and Ipate 2008) suggests some initial steps towards building a P system testing theory, based on rule coverage. Another approach to P system testing relies on conformance testing techniques for finite state machines and involves the construction of an associated automaton, which is used as a basis for test set derivation (Ipate and Gheorghe 2009a). Other recent approaches use mutation testing (Ipate and Gheorghe 2009b) and nondeterministic stream X-machines (Ipate and Gheorghe 2009c).

Although theoretical aspects of P system testing have been discussed in previous works (Gheorghe and Ipate 2008; Ipate and Gheorghe 2009a, b, c), this paper is the first one, to the best of our knowledge, which realizes an experimental comparison between some testing strategies for P systems. In the following:

1. Different testing techniques are applied to several implementations of transitional, non-deterministic, cell-like P systems.
2. Mutation analysis is used to evaluate the efficiency of different methods or test criteria. In particular, we evaluate how good a test set is depending on how many mutants it detects.
3. Some practical methodologies, for using the test sets obtained from P system specifications, are proposed and evaluated.

## 2 Background

The previous work on P systems testing was developed for a particular class of *cell-like* P systems (Păun and Rozenberg 2002), involving a non-deterministic use of evolution and communication rules. Some features like membrane division, creation or dissolution are difficult to be covered in the current approach, whereas others, like charges or polarisation, will be considered in the future work.

In any P system membranes play a key role. Each membrane defines a region. A region without any membrane inside is called an elementary one. Each region contains, apart from zero or many membranes, a multiset of objects and a set of rules. More precisely, a cell-like P system consists of:

- a hierarchical arrangement of *membranes*, embedded in the *skin membrane* (the one which separates the system from its environment);
- *objects* occurring inside the regions delimited by membranes, coding simple molecules or complex chemical compounds;

- *rules* assigned to the regions of the membrane structure, acting in each region upon the objects contained inside.

A *configuration* of a P system is represented by the current membrane structure and the multisets of objects occurring in each region. The system will go from one configuration to a new one by applying the rules in a non-deterministic and maximally parallel manner, i.e., at each step, in each membrane it is applied a maximal multiset of rules. A *computation* is defined by a set of steps, when the system moves from one configuration to another one. The system will halt when no more rules are available to be applied—the corresponding configuration is called *terminal configuration* and the computation is called *terminal computation*. Usually, the result of the computation is obtained in a specified component of the system, called the *output region*. All the computations starting from the same initial multiset define a *derivation tree*.

In what follows, a basic P system using transformation and communication rules is formally defined (Păun 2002; Păun and Rozenberg 2002).

**Definition 1** A P system is a tuple

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

where

- $V$  is an *alphabet*; its elements are called *objects*;
- $\mu$  is a *membrane structure* consisting of  $m$  membranes, with the membranes and the regions labelled in a one-to-one manner with elements of a given set  $\Lambda$ , usually, the set  $\{1, \dots, m\}$ ;  $m$  is called the *degree* of  $\Pi$ ;
- $w_i, 1 \leq i \leq m$ , are strings which represent *initial multisets* over  $V$  associated with the regions of  $\mu$ ;
- $R_i, 1 \leq i \leq m$ , are *transformation–communication rules* associated with the regions of  $\mu$ ; each rule of  $R_i$  has the form  $x \rightarrow y$ , where  $x \in V$ , and  $y$  defines a multiset over  $\{a_j | a \in V, j \in \{here, out, 1, \dots, m\}\}$  ( $a_{here}$  means  $a$  remains in the current region,  $i$ ; subsequently *here* will be ignored;  $a_{out}$  indicates that  $a$  has to go out of  $i$  to the outer region;  $a_j, 1 \leq j \leq m$ , shows that  $a$  goes to the region  $j$  that must be directly contained by the current membrane); applying a rule means replacing  $x$  by  $y$  and following the target indications;
- $i_0$  is a number between 1 and  $m$  which specifies the *output membrane* of  $\Pi$ .

## 2.1 Grammar-like testing for P systems

A set of coverage criteria for P system rules, inspired from grammar testing, is presented by Gheorghe and Ipate (2008). Test sets should be further designed to satisfy each coverage criterion. In the following, we summarize the main coverage criteria, but for simplicity, we will provide the definitions only for one membrane P systems (for a detailed discussion, in the case of more membranes, see Gheorghe and Ipate 2008). It is known that a P system with several membranes can be transformed into a one membrane P system. In the examples presented in Sect. 4, we will discuss only the case of one membrane P systems.

P systems  $\Pi_i = (V_i, \mu, w_i, R_i, 1), i = 1, 2$  will be considered in the following, where

- $\mu = [1]_1$
- $V_1 = \{s, a, b, c\}, w_1 = s, R_1 = \{r_1: s \rightarrow ab, r_2: a \rightarrow c, r_3: b \rightarrow bc, r_4: b \rightarrow c\}$ ;
- $V_2 = \{a, b, c\}, w_2 = a, R_2 = \{r_1: a \rightarrow bc, r_2: b \rightarrow a, r_3: b \rightarrow b, r_4: b \rightarrow c\}$ .

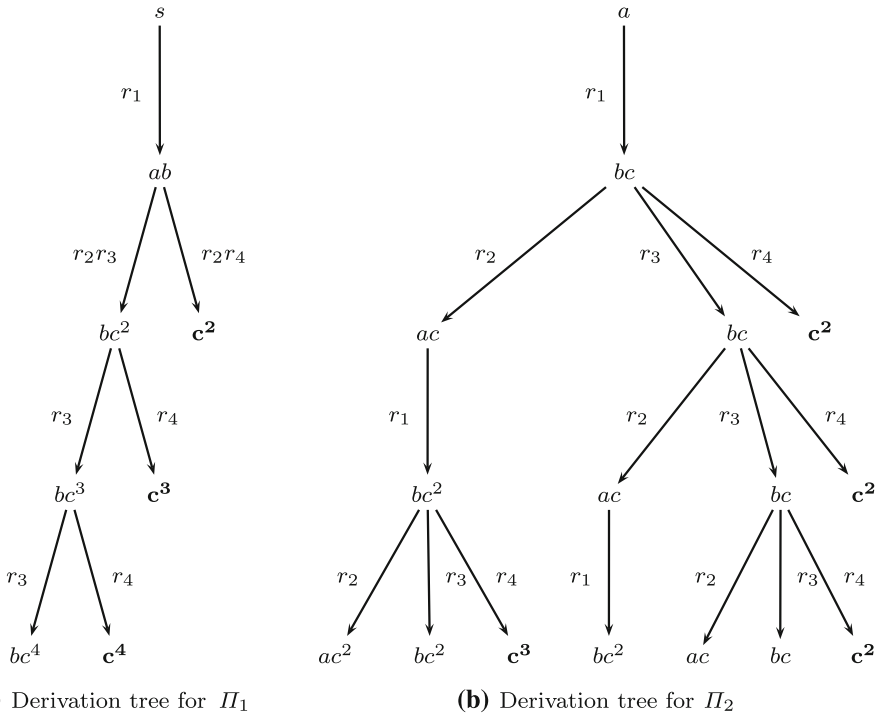


Fig. 1 Derivation trees for  $k = 4$

Figure 1 depicts the derivation trees for  $\Pi_1$  and  $\Pi_2$ , consisting of all the computations of maximum  $k = 4$  steps. The nodes that represent terminal configurations are in bold.

**Definition 2** A multiset denoted by  $u \in V^*$ , covers a rule  $r: a \rightarrow v \in R$ , if there is a derivation  $w \xRightarrow{*} xay \xRightarrow{*} x'vy' \xRightarrow{*} u; x, y, x', y', v, u \in V^*, a \in V, w \in V^*$  is the initial multiset. If there is no further derivation from  $u$ , then this is called a *terminal coverage*.

Considering the  $\Pi_1$  system described before, the multiset  $ab$  covers the rule  $r_1$ , the multiset  $bc^2$  covers the rules  $r_1, r_2, r_3$  and  $c^2$  covers  $r_1, r_2, r_4$ .

**Definition 3** A set  $T \subseteq V^*$ , is called a *test set* that satisfies the *rule coverage (RC)* criterion if for each rule  $r \in R$  there is  $u \in T$  which covers  $r$ . If every  $u \in T$  provides a terminal coverage then  $T$  is called a test set that satisfies the *rule terminal coverage (RTC)* criterion.

The test sets  $T_{1,rc} = \{bc^2, c^2\}$  and  $T_{2,rc} = \{ac, bc, c^2\}$  satisfy the RC criterion for  $\Pi_1$  and  $\Pi_2$ , respectively. The RTC criterion is accomplished by  $T_{rtc} = \{c^2, c^3\}$  for both P systems,  $\Pi_1$  and  $\Pi_2$ . Obviously, each test set which satisfies RTC criterion is also a test set according to the RC criterion.

**Definition 4** A rule  $r \in R, r: a \rightarrow ubv, u, v \in V^*, a, b \in V$ , is called a *direct occurrence* of  $b$ . For every symbol  $b \in V$ , we denote by  $Occs(\Pi, b)$ , the set of all direct occurrences of  $b$ .

For the P systems  $\Pi_1$  and  $\Pi_2$ , the sets of direct occurrences are:  $Occs(\Pi_1, s) = \emptyset$ ;  $Occs(\Pi_1, a) = \{r_1: s \rightarrow ab\}$ ;  $Occs(\Pi_1, b) = \{r_1: s \rightarrow ab, r_3: b \rightarrow bc\}$ ;  $Occs(\Pi_1, c) = \{r_2: a \rightarrow c, r_3: b \rightarrow bc, r_4: b \rightarrow c\}$ ;  $Occs(\Pi_2, a) = \{r_2: b \rightarrow a\}$ ;  $Occs(\Pi_2, b) = \{r_1: a \rightarrow bc, r_3: b \rightarrow b\}$ ;  $Occs(\Pi_2, c) = \{r_1: a \rightarrow bc, r_4: b \rightarrow c\}$ .

**Definition 5** A multiset  $z \in V^*$  covers the rule  $r: b \rightarrow y \in R$  for the *direct occurrence* of  $b$ ,  $a \rightarrow ubv \in R$ , if there is a derivation  $w \Rightarrow^* u_1av_1 \Rightarrow^* u'_1ubvv'_1 \Rightarrow^* u''_1u'yv'v''_1 \Rightarrow^* z$ ;  $u, v, u', v', u_1, v_1, u'_1, v'_1, u''_1, v''_1, y \in V^*$ ,  $a, b \in V$ . A set  $T_r$  is said to cover  $r: b \rightarrow y$  for all direct occurrences of  $b$  if for any occurrence  $o \in Occs(\Pi, b)$  there is  $t \in T_r$  such that  $t$  covers  $r$  for  $o$ .

**Definition 6** A set  $T$  is said to achieve *context-dependent rule coverage* (CDRC) for  $\Pi$  if it covers all  $r \in R$  for all their direct occurrences. If every  $z \in T$  provides a terminal coverage then  $T$  is called a test set that satisfies the context-dependent rule terminal coverage (CDRTC) criterion.

The CDRC criterion can be achieved for  $\Pi_1$  using  $T_{1,cdrc} = \{c^2, c^3, bc^3\}$  and the CDRTC criterion using  $T_{1,cdrtc} = \{c^2, c^3, c^4\}$ .

### 2.2 Finite state machine-based testing for P systems

The testing technique introduced by Gheorghe and Ipate (2008) and Ipate and Gheorghe (2009) is based on the construction of an automata which generates the test set associated to a P system under investigation.

First, a derivation tree of a P system will be built, considering all computations of maximum length  $k$ . From the derivation tree, the minimal deterministic finite cover automaton (DFCA) will be constructed. A DFCA is a finite automaton that accepts all words of a given finite language, but can also accept words that are longer than any word in the language. This DFCA, having the transitions labelled with the multiset of rules that can be applied in one computation step, will provide a (finite) approximation for the computations of the P system and will be the basis for test cases generation.

For  $\Pi_1$  and  $\Pi_2$  defined previously, Fig. 2 shows the DFCA's over the alphabet  $A = \{r_1^{i_1} \dots r_m^{i_m} \mid 0 \leq i_1 \leq N_1, \dots, 0 \leq i_m \leq N_m\}$ , where  $r_1^{i_1} \dots r_m^{i_m}$  describes the multiset with  $i_j$  occurrences of the rule  $r_j$ ,  $1 \leq j \leq m$ .

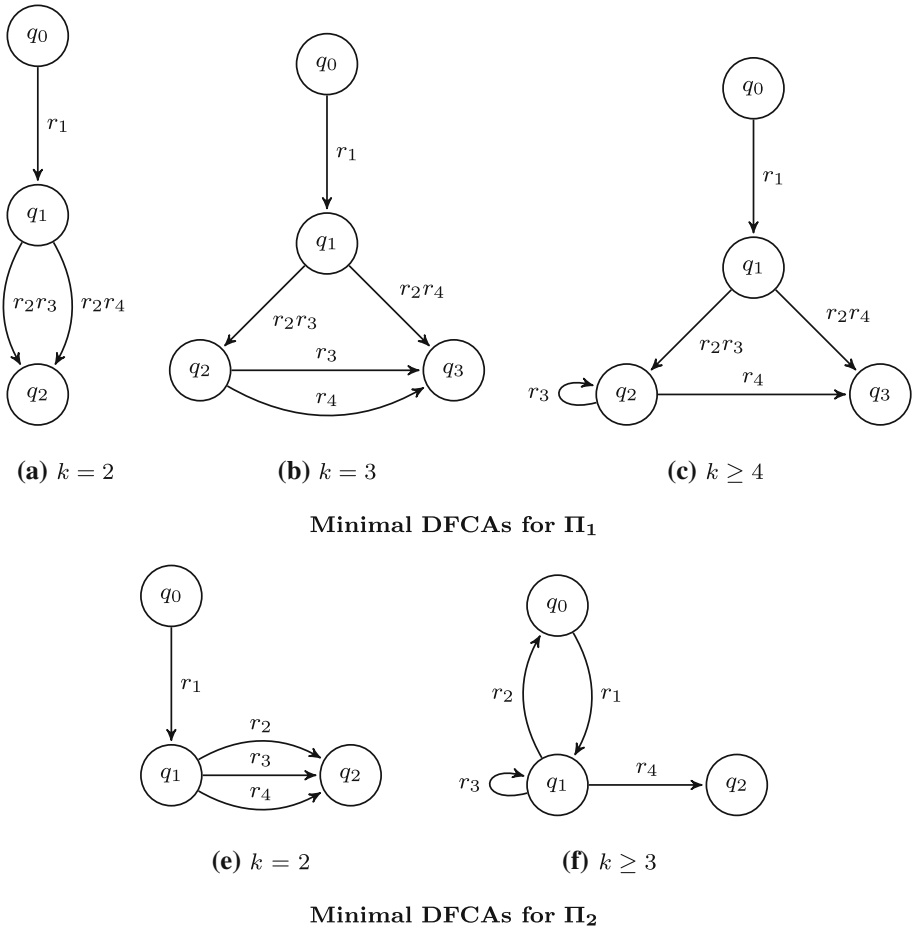
Figure 2 is self-explanatory; however, for the construction of DFCA in the case of more complex P systems, Gheorghe and Ipate (2008) provides further technical and theoretical details. Further, an improved procedure for generating test sets directly from the P system specification (without explicitly constructing the minimal DFCA model) is presented by Ipate and Gheorghe (2009a).

*FSM-based coverage criteria.* Once the minimal DFCA  $M = (A, Q, q_0, F, h)$  (where  $A$  is the input alphabet,  $Q$  is the set of states,  $q_0 \in Q$  is the initial state,  $F \subseteq Q$  is the set of final states and  $h: Q \times A \rightarrow Q$  is the next-state function) has been constructed for the P system  $\Pi$ , various coverage levels for finite state automata can be used (Gheorghe and Ipate 2008), such as *state coverage* and *transition coverage*.

**Definition 7** A set  $T \subseteq V^*$ , is called a *test set* that satisfies the *state coverage* (SC) criterion if for each state  $q$  of  $M$  there exist  $u \in T$  and a path  $s \in A^*$  that reaches  $q$  ( $h(q_0, s) = q$ ) such that  $u$  is derived from  $w$  through the computation defined by  $s$ .

**Definition 8** A set  $T \subseteq V^*$ , is called a *test set* that satisfies the *transition coverage* (TC) criterion if for each state  $q$  of  $M$  and each  $a \in A$  such that  $a$  labels a valid transition from  $q$  ( $h(q, a)$  is defined), there exist  $u, u' \in T$  and a path  $s \in A^*$  that reaches  $q$  such that  $u$  and  $u'$  are derived from  $w$  through the computation defined by  $s$  and  $sa$ , respectively.

Considering the state cover set  $SCS = \{\varepsilon, r_1, r_1 \cdot r_2r_3, r_1 \cdot r_2r_4\}$  and the transition cover set  $TCS = \{\varepsilon, r_1, r_1 \cdot r_2r_3, r_1 \cdot r_2r_4, r_1 \cdot r_2r_3 \cdot r_3, r_1 \cdot r_2r_3 \cdot r_4\}$  for the associated automaton of  $\Pi_1$  (for clarity  $\cdot$  is used as concatenation operator), the corresponding test



**Fig. 2** Deterministic finite state automata

sets for the P system are  $T_{1,sc} = \{s, ab, bc^2, c^2\}$  and  $T_{1,tc} = \{s, ab, bc^2, c^2, bc^3, c^3\}$ , respectively.

Depending on the maximum number of steps  $k$ , chosen for the derivation tree, different associated DFCA can be obtained and consequently the test sets obtained will have different degrees of efficiency.

*The W-method for P systems.* A stronger approach to P system testing is based on a finite state machine conformance technique, namely the *W*-method. Originally, the *W*-method was designed to establish equivalence between a finite state machine specification and the (unknown) implementation under test. It was later extended to establish the conformance of the implementation against a DFCA specification (Ipate 2006) and the new variant was applied to P system testing (Ipate and Gheorghe 2009a). In what follows we provide a brief description of the *W*-method for P systems. Suppose we have a finite automaton  $M = (A, Q, q_0, F, h)$ ; this is the minimal DFCA constructed from the computations of length less than or equal to  $k$  of the P system, as shown above.

**Definition 9**  $S \subseteq A^*$  is called a *proper state cover* of  $M$  if for every state  $q$  of  $M$  there exists  $s \in S$  such that  $s$  reaches  $q$  and  $s$  is the shortest sequence with this property (i.e., for every  $t \in A^*$ , if  $t$  reaches  $q$  then the length of  $s$  is less than or equal to the length of  $t$ ).

**Definition 10**  $W \subseteq A^*$  is called a *strong characterization set* of  $M$  if for every two states  $q_1$  and  $q_2$  of  $M$  and every  $j \geq 0$ , if  $q_1$  and  $q_2$  can be distinguished by a sequence of length  $j$  then  $q_1$  and  $q_2$  can be distinguished by a sequence  $s \in W$  of length  $j$ .

Naturally, in the above definition, it is sufficient, for any  $q_1$  and  $q_2$ , to consider  $j$  to be the length of the shortest sequences that distinguish between  $q_1$  and  $q_2$ . Note that, for a non-trivial finite automaton, which contains both final and non-final states, the empty sequence  $\varepsilon$  is the shortest sequence that distinguishes final from non-final states. Thus  $\varepsilon \in W$ . In our examples (Fig. 2), only the final states are explicitly given; however, the missing transitions can be represented as transitions to an (implicit) non-final state and so  $\varepsilon \in W$ .

For the minimal DFCA for  $\Pi_1$  and  $k \geq 4$ , as represented in Fig. 2 (c), a proper state cover is  $S = \{\varepsilon, r_1, r_1 \cdot r_2r_3, r_1 \cdot r_2r_4\}$  and a strong characterization set is  $W = \{\varepsilon, r_1, r_2r_3, r_3\}$ .

Consider now an implementation under test that can be modelled by some unknown finite automaton  $M' = (A, Q', q_0', F', h')$ . We assume that the difference between the number of states of  $M'$  and the number of states of  $M$  cannot exceed a known non-negative integer  $\sigma$  (in our experiments we will consider  $\sigma = 0$ ). We denote by  $L_M$  and  $L_{M'}$  the languages accepted by  $M$  and  $M'$ , respectively, and by  $A[j]$  the set of all sequences of length at most  $j$  with elements in  $A, j \geq 0$ . Then a test suite is generated using the formula

$$U_\sigma = SA[\sigma + 1]W \cap A[l].$$

The above test suite establishes that the implementation under test conforms to the DFCA specification:

**Theorem 1** (Ipatе 2006) *For every DFA  $M'$  which has at most  $\sigma$  more states than  $M, L_M \cap A[l] = L_{M'} \cap A[k]$  if and only if  $L_M \cap V_\sigma = L_{M'} \cap V_\sigma$ .*

The above results can be used in the context of P system testing, giving rise to the following definition:

**Definition 11** Given  $\sigma \leq 0$ , a set  $T \subseteq V^*$ , is called a *test set* that satisfies the *W-method* (WM) criterion for  $\sigma$  if there exist  $S$  a proper state cover of  $M, W$  a strong characterization set of  $M$  and  $U_\sigma = SA[\sigma + 1]W \cap A[l]$  such that for each sequence  $s \in U_\sigma$  there exists  $u \in T$  which is derived from  $w$  through the computation defined by  $s$ .

### 3 Testing methodologies

Although each of the previously designed coverage-based testing techniques for P systems produces one or several test sets, none of them defines how they will be applied to test an implementation, and what principles are employed to build these test sets. In this section, we define three strategies for generating test sets that will be combined with each of the above discussed testing technique, and will provide a generic procedure for testing an implementation against a given test set. Three main cases will be considered for generating the test set from a computation, by taking into account: (a) the last configurations, (b) the union of the configurations occurring during the computation and (c) the sequence of configurations occurring alongside the computation. (Note that, for simplicity, in Sect. 2, the test sets associated with various testing strategies were only defined for the case (a).)

The results will be evaluated in the next section using mutants of the P systems and measuring the mutation score (number of detected mutants, for each test case).

The *testing procedure* should consider the *non-deterministic* nature of these systems. Let us consider one ‘item’  $c$  in the test set  $T$ , that represents a configuration of the P system, obtained after applying some rules. One common way to test a ‘black-box’ implementation of a non-deterministic system (e.g., a finite state automaton or a P system) is to provide the necessary input to the system (if any), let it evolve and check if the expected output is produced (in our case if the expected configuration is reached). Due to the system non-determinism, several correct answers could be produced, so the test should be repeated  $n$  times,  $n$  large enough, chosen to ensure that all the non-deterministic choices have the chance to be observed. If in  $n$  tries we obtain at least once the expected response, then the test is successful. Otherwise, the implementation contains a fault. For example, considering the derivation trees from Fig. 1, there are four possible paths for  $\Pi_1$  and nine paths for  $\Pi_2$ . If  $n = 10$ , there is a high probability that one particular computation of the P system is not observed in  $n = 10$  tries. This is the reason why, in the experiments performed in Sect. 4, the number of tries  $n = 1000$  was considered to be large enough to reveal all the possible computations of the two P systems, when  $k \leq 5$ .

In the sequel, we will consider that the P system implementation has a reliable *reset* operation, that brings the P system into the initial state, i.e., initial multiset, and the P system implementation can be run using a ‘step-by-step’ approach or run until a terminal configuration is reached. A pseudo-code description of a function which verifies if the P system `psys` reaches a given expected configuration `expConf`, is given below. It checks at most the first  $k$  configurations of the P system; the procedure is repeated  $n$  times, that means that  $n$  random computations of the P system are analysed.

---

```
function is_pSystem_conform_to_expected_configuration(psys, expConf)
begin
  for iTry = 1 to n // n chosen large enough
    begin
      psys.reset();
      if psys.crtConf = expConf then
        return true;
      end if;
      iStep = 0;
      repeat
        if not psys.crtConf.isTerminal() then
          psys.step();
          iStep := iStep + 1;
        end if;
        if psys.crtConf = expConf then
          return true;
        end if;
      until (iStep > k OR psys.crtConf.isTerminal())
    end
  return false
end
```

---

As previously mentioned, several test sets can be built to satisfy the same coverage criterion. For  $\Pi_1$  the computations  $s \Rightarrow ab \Rightarrow bc^2$  and  $s \Rightarrow ab \Rightarrow c^2$  provide the necessary steps for RC. We can consider  $T_a = \{bc^2, c^2\}$ , which takes into account only the last configurations of the above computations (case (a) above),  $T_b = \{ab, bc^2, c^2\}$ , the union of the configurations involved in these computations (case (b)) and  $T_c = \{ab \cdot bc^2, ab \cdot c^2\}$  (where  $\cdot$  means multiset concatenation), which provides the sequences of configurations occurring alongside the two computations (case (c)). Of course, a richer test set is expected to have a higher testing efficiency.

### 3.1 Configuration-based testing methodology

Let us consider the set of expected configurations  $Conf = \{C_1, \dots, C_m\}$  and a constant  $k$ , representing the maximum number of steps needed to obtain all the configurations  $C_1, \dots, C_m$ , starting from the initial multiset. For example, for testing a P system according to the RC criterion, a derivation tree with maximum  $k$  levels can be obtained, and this constant  $k$  can be used as the maximum number of steps needed to obtain the configurations of the derivation tree.

*Example* The set  $Conf = \{ab, bc^2, c^2\}$ , satisfies the RC criterion for the P system  $\Pi_1$  when  $k \geq 2$ .

*Approach* We introduce a *configuration-based testing* methodology as follows: for each configuration  $C_i \in Conf$ ,  $1 \leq i \leq m$ , check if the P system can arrive (in  $n$  tries,  $n$  large enough) to the given configuration  $C_i$ , after at most  $k$  steps (it can get in 0 steps, if  $C_i = w$ , the initial multiset, or in  $k \geq 1$  steps). This methodology covers cases (a) and (b) above, whereas the last case, (c), will be discussed in the next section.

These configurations can be obtained using any of the above mentioned test set generation principles and any of the three ways of selecting configurations corresponding to a computation. Due to the non-deterministic and parallel nature of the P systems, the same configuration  $C_i$  can be obtained using different computations (following various rules or having different number of steps), e.g., the configuration  $c^2$  can be generated for  $\Pi_2$  using the computations:  $a \Rightarrow bc \Rightarrow c^2$  or  $a \Rightarrow bc \Rightarrow bc \Rightarrow c^2$ . Using the configuration-based testing methodology, it is enough to obtain only once each of the given configurations. Of course, a more thorough methodology for checking multiple computations of the same configuration within the limits of at most  $k$  steps can be devised, but this is not considered here.

An additional aspect is taken into account when RTC or CDRTC is used: a test for a configuration  $C_i$  is considered as being successful if this configuration is obtained in maximum  $k$  steps and the final step is also a *terminal* one.

If in  $n$  tries ( $n$  large enough) a configuration  $C_i \in Conf$  is not obtained, in at most  $k$  steps at each try (or it is obtained, but in the case of RTC, CDRTC, it is not a terminal configuration), then a difference in behaviour is noticed between the specified P system and the implemented one. We say that the implementation contains a *fault* (the configuration  $C_i$  that is obtained in the specification, cannot be obtained in the implementation) and, in the case of the mutation testing, *the mutant is killed*. If the implementation computes all the configurations  $Conf = \{C_1, \dots, C_m\}$  in at most  $k$  steps, then we say that it *conforms* to all the given configurations of the set  $Conf$  and we cannot differentiate between the specification and the implementation using the given set  $Conf$ . This corresponds to the case when the mutant is *alive*, i.e., it is not identified as a fault.

### 3.2 Sequence of configurations based testing methodology

Let us consider a derivation tree with all the computations of length at most  $k$  involved, satisfying a certain testing principle. The set of all these computations is  $Deriv = \{Comp_1, \dots, Comp_m\}$ , where each computation  $Comp_i$ ,  $1 \leq i \leq m$ , is represented by,  $Comp_i = \langle w \Rightarrow C_{i_1} \Rightarrow C_{i_2} \Rightarrow \dots \Rightarrow C_{i_p} \rangle$ , with the meaning that the P system is going from the initial multiset  $w$  to  $C_{i_1}, \dots$ , until  $C_{i_p}$ .

We introduce a *sequence of configurations based testing* methodology as follows: the set of all sequences of configurations,  $Conf = \{SC_1, \dots, SC_m\}$ , consists of sequences  $SC_i = C_{i_1} \cdot C_{i_2} \cdot \dots \cdot C_{i_p}$ , where the corresponding configurations,  $C_{i_j}, 1 \leq j \leq p$ , occur, in this order, in the computation  $Comp_i = \langle w \Rightarrow C_{i_1} \Rightarrow C_{i_2} \Rightarrow \dots \Rightarrow C_{i_p} \rangle$ . In this case it is also checked that the P system can arrive (in  $n$  tries,  $n$  large enough) to the given sequence of configurations  $SC_i$ , after at most  $k$  steps.

*Example* For the  $\Pi_1$  P system the set of all computations involved by the RC criterion is given by  $Deriv = \{s \Rightarrow ab \Rightarrow bc^2, s \Rightarrow ab \Rightarrow c^2\}$  and the set of sequences of configurations is  $Conf = \{ab \cdot bc^2, ab \cdot c^2\}$ .

*Approach.* The testing methodology will remain the same as the one mentioned in the previous section, but instead of considering individual configurations, we will select sequences of configurations from the set  $Conf$ . The fault in this case, if identified, will mean that the sequence of configurations is not obtained, which should be a stronger testing principle than the above mentioned two.

## 4 Empirical evaluation

In order to investigate the fault detection capabilities of the previously mentioned testing methodologies, we performed the following experiment: mutants were generated for the P system and all the tests were run against the mutants to check their fault detection efficiency.

### 4.1 Mutation testing for P systems

Traditionally, mutation testing consists in generating mutant versions of a program, that differ from the original source by small modifications. Usually only first-order mutants, that have only one difference from the original program, are employed for experimentation. This is based on the basic assumption of mutation testing: a test set that detects simple faults (such as those introduced by first order mutation) will detect complex faults—the combination of several simple faults. Experiments have shown that mutation testing is an appropriate approach for testing: the generated mutants are similar to the real faults (Andrews et al. 2005), a fact that supports the theory of mutation testing, the assumption that faults made by programmers will be detected by test suites that kill mutants (Andrews et al. 2005).

Although mutation testing is employed mainly for programs, mutation operators can be defined for other artifacts, including program or XML specifications and input languages (Offutt et al. 2006).

For P systems, a very recent paper (Ipate and Gheorghe 2009b) presents a strategy for mutation testing, based on context-free grammars defining Kripke structures associated to a P system. In the followings, we will present a different approach for generating first-order

mutants of a P system. For mutant generation, we will use a set of mutant operators, that are defined in the context of one membrane P systems  $\Pi = (V, [ ]_1, w, R, 1)$ . We will consider that the generated mutants  $\Pi'$  have the same alphabet  $V$ , the same membrane structure  $[ ]_1$  and consider modifications of the initial multiset and of the set of rules  $R$ , defined as follows:

1. *Initial Multiset* modification obtained by using a literal:
  - (a) *Deletion* (IMD): one symbol is deleted from the initial multiset. For example, if the original P system has the initial multiset  $w = ab$ , two IMD mutants can be generated, having  $w = a$  and  $w = b$ , respectively.
  - (b) *Replacement* (IMR): one symbol of the initial multiset  $w$  is replaced by other symbols from the input alphabet  $V$ . For a P system having  $w = s$  and  $V = \{s, a, b, c\}$ , three IMR mutants exist, having:  $w = a$ ,  $w = b$  and  $w = c$ , respectively.
  - (c) *Insertion* (IMI): one symbol from  $V$  is added to the initial multiset. For  $w = s$  and  $V = \{s, a, b, c\}$ , four IMI mutants exist, having:  $w = ss$ ,  $w = sa$ ,  $w = sb$  and  $w = sc$ , respectively.
2. *Rule Deletion* (RD): the mutant P system has the set of rules  $R \setminus \{r_i\}$ , where  $r_i \in R$ . For each of  $\Pi_1$  and  $\Pi_2$  defined above four RD mutants can be generated.
3. *Rule Modification*, by performing one of the following operations:
  - (a) *Literal Deletion* (RMD): one symbol is deleted from the rule  $r: u \rightarrow v$ . If the original P system has a rule  $r: b \rightarrow bc$ , two RMD mutants can be generated, having  $r^{(1)}: b \rightarrow b$  and  $r^{(2)}: b \rightarrow c$ , respectively, because the left hand side of a rule cannot be empty. over  $V$ .
  - (b) *Literal Replacement* (RMR): one symbol of the rule  $r: u \rightarrow v$  is replaced by other symbols from the input alphabet  $V$ . For the rule  $r: a \rightarrow c$ ,  $V = \{s, a, b, c\}$ , six mutants can be defined, having the modified rules:  $r^{(1)}: s \rightarrow c$ ,  $r^{(2)}: b \rightarrow c$ ,  $r^{(3)}: c \rightarrow c$ ,  $r^{(4)}: a \rightarrow s$ ,  $r^{(5)}: a \rightarrow a$  and  $r^{(6)}: a \rightarrow b$ , respectively.
  - (c) *Literal Insertion* (RMI): one symbol from  $V$  is added to the rule  $r: u \rightarrow v$ . For the rule  $r: a \rightarrow c$ ,  $V = \{s, a, b, c\}$ , eight mutants can be defined, having the modified rules:  $r^{(1)}: as \rightarrow c$ ,  $r^{(2)}: aa \rightarrow c$ ,  $r^{(3)}: ab \rightarrow c$ ,  $r^{(4)}: ac \rightarrow c$ ,  $r^{(5)}: a \rightarrow cs$ ,  $r^{(6)}: a \rightarrow ca$ ,  $r^{(7)}: a \rightarrow cb$  and  $r^{(8)}: a \rightarrow cc$ , respectively.

Considering the P systems  $\Pi_1$  and  $\Pi_2$ , the distribution of generated mutants is

P sys.	IMD	IMR	IMI	RD	RMD	RMI	RMR	Total
$\Pi_1$	1	3	4	4	6	32	30	80
$\Pi_2$	1	2	3	4	5	24	18	57

It can be observed that we have not considered the *rule insertion* for two reasons:

A new rule  $r: u \rightarrow v$ ,  $u, v \in V^*$  has an infinite space. This can be reduced only if length constraints are added, such as  $|u| = 1$ , or  $|v| \leq 3$ . Having these constrains the number of possible rules to be inserted is finite,  $4 \cdot (1 + 4 + 10 + 20) = 140$ , but the number of generated mutants is still large for experimentation purposes.

Also, the testing techniques described previously check that the implementation can generate the expected configurations or computations. Consequently, additional

configurations or computations, that are induced by a new inserted rule, cannot be detected with a methodology that checks only the *incompleteness* of the implementation with respect to the specification. For example, a new inserted rule could give birth to additional computations in the P system (additional paths in the derivation tree). But the testing methodology checks only if the previous configurations (computations) can be observed; it does not verify if there are extra configurations (computations).

## 4.2 Experimental results

In our experiments we have used P-lingua to specify the P systems and their mutants, to simulate their computations and to run the test sets against them. P-lingua (Díaz-Pernil et al. 2008; The P-Lingua Web Site 2009) is a programming language for membrane computing, developed as a free software framework for cell-like P systems. Its main component is a Java library, `pLinguaCore`, that accepts as input text files (either in XML or in P-Lingua format) describing the P system model (García-Quismondo et al. 2009).

The library includes several built-in simulators for each supported model: active membrane P systems with membrane division rules or membrane creation rules, transition P systems, symport/antiport P systems, stochastic P systems and probabilistic P systems. The software package P-Lingua 2.0 contains the `pLinguaCore` library and a user interface called `pLinguaPlugin`. The P-lingua software was used to simulate processes with a family of P systems solving the SAT problem (Díaz-Pernil et al. 2008) and to describe and simulate ecosystems by means of P systems (García-Quismondo et al. 2009).

For each P system a set of mutants was generated, following the operators described in Sect. 4.1. Also, test sets were generated, following the methodologies from Sects. 2.1, 2.2 and 3. In each case  $n$  was considered 1000. Tables 1 and 2 present the mutation scores obtained in each case. The first column in the tables shows the testing technique (RC, CDRC, RTC, CDRTC, SC, TC, WM), the second column specifies the testing methodology:

- takes into account only the *last* configurations of the computations,
- the *union* of the configurations occurring during the computation, and
- the *sequence* of configurations occurring alongside the computation.

The number  $k$  of steps considered for maximum length of derivation is given next, as it has influence on the DFCA derived and the test set obtained (see Sect. 2.2). The test ID is given in the fourth column, followed by the test length (number of configurations checked) and the mutation score (percentage of detected mutants).

*Test cases for  $\Pi_1$ .* For the first P system,  $\Pi_1$ , the following test sets, generated using rule-coverage techniques, were used:  $T_{rc,a} = \{bc^2, c^2\}$ ,  $T_{rc,b} = \{ab, bc^2, c^2\}$ ,  $T_{rc,c} = \{ab \cdot bc^2, ab \cdot c^2\}$ ,  $T_{rtc,a} = \{c^3\}$ ,  $T_{rtc,c} = \{ab \cdot bc^2 \cdot c^3\}$ ,  $T_{cdrc,a} = \{c^3, c^2, bc^3\}$ ,  $T_{cdrc,b} = \{ab, bc^2, c^3, c^2, bc^3\}$ ,  $T_{cdrc,c} = \{ab \cdot bc^2 \cdot c^3, ab \cdot c^2, ab \cdot bc^2 \cdot bc^3\}$ ,  $T_{cdrtc,a} = \{c^2, c^3, c^4\}$ ,  $T_{cdrtc,c} = \{ab \cdot c^2, ab \cdot bc^2 \cdot c^3, ab \cdot bc^2 \cdot bc^3 \cdot c^4\}$ .

Depending on the value  $k$ , the maximum number of steps considered for derivation, different DFCA's can be produced for  $\Pi_1$ . These are given in Fig. 2 for (i)  $k = 2$ , (ii)  $k = 3$ , (iii)  $k \geq 4$ . For each automaton, different test cases were obtained.

Due to space constraints, we will enumerate only the test cases for  $k = 2$  and  $k \geq 4$ :  $T_{sc,a,k=2} = \{bc^2\}$ ,  $T_{sc,b,k=2} = \{s, ab, bc^2\}$ ,  $T_{sc,c,k=2} = \{ab \cdot bc^2\}$ ,  $T_{tc,a,k=2} = T_{wm,a,k=2} = \{bc^2, c^2\}$ ,  $T_{tc,b,k=2} = T_{wm,b,k=2} = \{s, ab, bc^2, c^2\}$ ,  $T_{tc,c,k=2} = T_{wm,c,k=2} = \{ab \cdot bc^2, ab \cdot c^2\}$ ,  $T_{sc,a,k \geq 4} = \{bc^2, c^2\}$ ,  $T_{sc,b,k \geq 4} = \{s, ab, bc^2, c^2\}$ ,  $T_{sc,c,k \geq 4} = \{ab \cdot bc^2, ab \cdot c^2\}$ ,

**Table 1** Mutation scores obtained using different testing approaches for  $\Pi_1$

Testing technique	Testing methodologies	$k$ steps	Test ID	Test length	Mutation score, %
RC	a	2	$T_{rc,a}$	2	95.00
RC	b	2	$T_{rc,b}$	3	98.75
RC	c	2	$T_{rc,c}$	4	98.75
RTC	a	3	$T_{rtc,a}$	1	76.25
RTC	c	3	$T_{rtc,c}$	3	97.50
CDRC	a	3	$T_{cdrc,a}$	3	95.00
CDRC	b	3	$T_{cdrc,b}$	5	98.75
CDRC	c	3	$T_{cdrc,c}$	8	100.00
CDTRC	a	4	$T_{cdtrc,a}$	3	92.50
CDTRC	c	4	$T_{cdtrc,c}$	9	100.00
SC	a	2	$T_{sc,a,k=2}$	1	73.75
SC	b	2	$T_{sc,b,k=2}$	3	77.50
SC	c	2	$T_{sc,c,k=2}$	2	77.50
TC, WM	a	2	$T_{tc,a,k=2} = T_{wm,a,k=2}$	2	86.25
TC, WM	b	2	$T_{tc,b,k=2} = T_{wm,b,k=2}$	4	91.25
TC, WM	c	2	$T_{tc,c,k=2} = T_{wm,c,k=2}$	4	98.75
SC	a	3	$T_{sc,a,k=3}$	2	95.00
SC	b	3	$T_{sc,b,k=3}$	4	98.75
SC	c	3	$T_{sc,c,k=3}$	4	98.75
TC, WM	a	3	$T_{tc,a,k=3} = T_{wm,a,k=3}$	3	98.75
TC, WM	b	3	$T_{tc,b,k=3} = T_{wm,b,k=3}$	6	98.75
TC, WM	c	3	$T_{tc,c,k=3} = T_{wm,c,k=3}$	8	100.00
SC	a	$\geq 4$	$T_{sc,a,k \geq 4}$	2	95.00
SC	b	$\geq 4$	$T_{sc,b,k \geq 4}$	4	98.75
SC	c	$\geq 4$	$T_{sc,c,k \geq 4}$	4	98.75
TC	a	$\geq 4$	$T_{tc,a,k \geq 4}$	3	95.00
TC	b	$\geq 4$	$T_{tc,b,k \geq 4}$	6	98.75
TC	c	$\geq 4$	$T_{tc,c,k \geq 4}$	8	100.00
WM	a	$\geq 4$	$T_{wm,a,k \geq 4}$	3	91.25
WM	b	$\geq 4$	$T_{wm,b,k \geq 4}$	7	95.00
WM	c	$\geq 4$	$T_{wm,c,k \geq 4}$	9	100.00

$T_{tc,a,k \geq 4} = \{c^2, bc^3, c^3\}$ ,  $T_{tc,b,k \geq 4} = \{s, ab, c^2, bc^2, bc^3, c^3\}$ ,  $T_{tc,c,k \geq 4} = \{ab \cdot c^2, ab \cdot bc^2 \cdot bc^3, ab \cdot bc^2 \cdot c^3\}$ ,  $T_{wm,a,k \geq 4} = \{c^2, c^3, bc^4\}$ ,  $T_{wm,b,k \geq 4} = \{s, ab, c^2, bc^2, bc^3, c^3, bc^4\}$ ,  $T_{wm,c,k \geq 4} = \{ab \cdot c^2, ab \cdot bc^2 \cdot c^3, ab \cdot bc^2 \cdot bc^3 \cdot bc^4\}$ .

Test cases for  $\Pi_2$ . We similarly define for the P system  $\Pi_2$  test sets according to the rule-coverage principle:  $T'_{rc,a} = \{ac, bc, c^2\}$ ,  $T'_{rc,b} = \{ac, bc, c^2\}$ ,  $T'_{rc,c} = \{bc \cdot ac, bc \cdot bc, bc \cdot c^2\}$ ,  $T'_{rtc,a} = \{c^2, c^3\}$ ,  $T'_{rtc,c} = \{bc \cdot bc \cdot c^2, bc \cdot ac \cdot bc^2 \cdot c^3\}$ ,  $T'_{cdrc,a} = \{ac, bc, bc^2, c^2\}$ ,  $T'_{cdrc,b} = \{ac, bc, bc^2, c^2\}$ ,  $T'_{cdrc,c} = \{bc \cdot bc \cdot ac, bc \cdot bc \cdot bc, bc \cdot ac \cdot bc^2, bc \cdot c^2, bc \cdot bc \cdot c^2\}$ ,  $T'_{cdtrc,a} = \{c^2, c^3\}$ ,  $T'_{cdtrc,c} = \{bc \cdot c^2, bc \cdot bc \cdot c^2, bc \cdot bc \cdot bc \cdot c^2, bc \cdot ac \cdot bc^2 \cdot c^3, bc \cdot ac \cdot bc^2 \cdot bc^2 \cdot c^3\}$ .

Two different DFCAs are defined for (d)  $k = 2$ , (e)  $k \geq 3$  and the following test cases produced:  $T'_{sc,a,k=2} = \{ac\}$ ,  $T'_{sc,b,k=2} = \{a, bc, ac\}$ ,  $T'_{sc,c,k=2} = \{bc \cdot ac\}$ ,  $T'_{tc,a,k=2} =$

**Table 2** Mutation scores obtained using different testing approaches for  $\Pi_2$

Testing technique	Testing methodologies	$k$ steps	Test ID	Test length	Mutation score, %
RC	a	2	$T'_{rc,a}$	3	78.94
RC	b	2	$T'_{rc,b}$	3	78.94
RC	c	2	$T'_{rc,c}$	6	100.00
RTC	a	4	$T'_{rtc,a}$	2	78.94
RTC	c	4	$T'_{rtc,c}$	7	100.00
CDRC	a	3	$T'_{cdr,c,a}$	4	78.94
CDRC	b	3	$T'_{cdr,c,b}$	4	78.94
CDRC	c	3	$T'_{cdr,c,c}$	14	100.00
CDTRC	a	5	$T'_{cdrtc,a}$	2	75.43
CDTRC	c	5	$T'_{cdrtc,c}$	18	100.00
SC	a	2	$T'_{sc,a,k=2}$	1	52.63
SC	b	2	$T'_{sc,b,k=2}$	3	56.14
SC	c	2	$T'_{sc,c,k=2}$	2	57.89
TC, WM	a	2	$T'_{tc,a,k=2} = T'_{wm,a,k=2}$	3	78.94
TC, WM	b	2	$T'_{tc,b,k=2} = T'_{wm,b,k=2}$	4	78.94
TC, WM	c	2	$T'_{tc,c,k=2} = T'_{wm,c,k=2}$	6	100.00
SC	a	$k \geq 3$	$T'_{sc,a,k \geq 3}$	1	57.89
SC	b	$k \geq 3$	$T'_{sc,b,k \geq 3}$	3	59.64
SC	c	$k \geq 3$	$T'_{sc,c,k \geq 3}$	2	61.40
TC	a	$k \geq 3$	$T'_{tc,a,k \geq 3}$	3	78.94
TC	b	$k \geq 3$	$T'_{tc,b,k \geq 3}$	4	78.94
TC	c	$k \geq 3$	$T'_{tc,c,k \geq 3}$	6	100.00
WM	a	$k \geq 3$	$T'_{wm,a,k \geq 3}$	3	78.94
WM	b	$k \geq 3$	$T'_{wm,b,k \geq 3}$	5	78.94
WM	c	$k \geq 3$	$T'_{wm,c,k \geq 3}$	8	100.00

$$T'_{wm,a,k=2} = \{ac, bc, c^2\}, T'_{tc,b,k=2} = T'_{wm,b,k=2} = \{a, ac, bc, c^2\}, T'_{tc,c,k=2} = T'_{wm,c,k=2} = \{bc \cdot ac, bc \cdot bc, bc \cdot c^2\}, T'_{sc,a,k \geq 3} = \{c^2\}, T'_{sc,b,k \geq 3} = \{a, bc, c^2\}, T'_{sc,c,k \geq 3} = \{bc \cdot c^2\}, T'_{tc,a,k \geq 3} = \{ac, bc, c^2\}, T'_{tc,b,k \geq 3} = \{a, ac, bc, c^2\}, T'_{tc,c,k \geq 3} = \{bc \cdot ac, bc \cdot bc, bc \cdot c^2\}, T'_{wm,a,k \geq 3} = \{ac, c^2, bc^2\}, T'_{wm,b,k \geq 3} = \{a, bc, ac, c^2, bc^2\}, T'_{wm,c,k \geq 3} = \{bc \cdot bc \cdot ac, bc \cdot c^2, bc \cdot ac \cdot bc^2\}.$$

A number of conclusions can be drawn from the results obtained:

- Generally, context-dependent rule coverage achieves a slightly improved fault detection score over simple rule coverage (when the same type of methodology is used); however, the increase in fault detection is well below the corresponding increase in test size. On the other hand, both types of coverage achieve 100% fault detection when the methodology (c) is used.
- The performance of the finite state machine based techniques depend heavily on the  $k$  used; this corresponds to our intuition, a larger value for  $k$  will produce a better approximation of the real model. In general, the transition coverage and the  $W$ -method based test sets produce similar scores, outperforming state coverage.

- A mutation score of 100% fault detection is achieved only when the methodology (c) is used; this indicates that, for rigorous testing, not only the resulting configurations need to be checked but also their sequence.

## 5 Conclusions

This paper reviews the existing techniques for P system testing, proposes a number of testing methodologies for practical use and performs an empirical evaluation of their fault-detection efficiency, based on mutation analysis. Future work may involve further experimental evaluations from different perspectives (e.g., seeding faults in real P system implementations) as well consider other types of P systems.

**Acknowledgements** This work was supported by CNCISIS – UEFISCSU, project number PNII – IDEI 643/2008. The authors would like to thank Ignacio Pérez-Hurtado and the other members of the Research Group on Natural Computing from the University of Seville, for their kind and effective help in adapting P-lingua to fit the class of P systems considered in this paper and to the anonymous reviewers for their comments and suggestions.

## References

- Andrews JH, Briand LC, Labiche Y (2005) Is mutation an appropriate tool for testing experiments? In: International conference on software engineering. ACM, New York, pp 402–411
- Díaz-Pernil D, Pérez-Hurtado I, Pérez-Jiménez MJ, Riscos-Núñez A (2008) A P-lingua programming environment for membrane computing. In: Workshop on membrane computing. Lecture notes in computer science, vol 5391. Springer, Berlin, pp 187–203
- García-Quismondo M, Gutiérrez-Escudero R, Pérez-Hurtado I, Pérez-Jiménez MJ, Riscos-Núñez A (2009) An overview of P-lingua 2.0. In: Workshop on membrane computing. Lecture notes in computer science, vol 5957. Springer, Berlin, pp 264–288
- Gheorghe M, Ipate F (2008) On testing P systems. In: Workshop on membrane computing. Lecture notes in computer science, vol. 5391. Springer, Berlin, pp 204–216
- Ipate F (2006) Bounded sequence testing from non-deterministic finite state machines. In: Testing of communicating systems. Lecture notes in computer science, Springer, Berlin, vol 3964, pp 55–70
- Ipate F, Gheorghe M (2009a) Finite state based testing of P systems. *Nat Comput* 8(4):833–846
- Ipate F, Gheorghe M (2009b) Mutation based testing of P systems. *Int J Comput Commun Control* 4(3): 253–262
- Ipate F, Gheorghe M (2009c) Testing non-deterministic stream X-machine models and P systems. *Electr Notes Theor Comput Sci* 227:113–126
- Kari L, Rozenberg G (2008) The many facets of natural computing. *Commun ACM* 51(10):72–83
- Offutt J, Ammann P, Liu L (2006) Mutation testing implements grammar-based testing. In: Second workshop on mutation analysis. IEEE Computer Society, Washington, DC, pp 12–12
- Păun Gh (1998) Computing with membranes. TUCS Report 208, Turku Center for Computer Science
- Păun Gh (2000) Computing with membranes. *J Comp System Sci* 61(1):108–143
- Păun Gh (2002) Membrane computing: an introduction. Springer, Berlin
- Păun Gh, Rozenberg G (2002) A guide to membrane computing. *Theor Comput Sci* 287(1):73–100
- Păun Gh, Rozenberg G, Salomaa A, eds. (2009) The Oxford Handbook on membrane computing. Oxford University Press, Oxford
- The P Systems Web Site (2009) <http://ppage.psystems.eu>
- The P-Lingua Web Site (2009) <http://www.p-lingua.org>