

# Finite state based testing of P systems

Florentin Ipate · Marian Gheorghe

Published online: 10 September 2008  
© Springer Science+Business Media B.V. 2008

**Abstract** In this paper, we propose an approach to P system testing based on finite state machine conformance techniques. Of the many variants of P systems that have been defined, we consider cell-like P systems which use non-cooperative transformation and communication rules. We show that a (minimal) *deterministic finite cover automaton (DFCA)* (a finite automaton that accepts all words in a given finite language, but can also accept words that are longer than any word in the language) provides the right approximation for the computation of a P system. Furthermore, we provide a procedure for generating test sets directly from the P system specification (without explicitly constructing the minimal DFCA model).

**Keywords** Membrane computing · P systems · Conformance testing · Test generation · Finite automata

## 1 Introduction

Membrane computing, the research field initiated by Gheorghe Păun in 1998 (Păun 2000), aims to define computational models, called P systems, which are inspired by the behaviour and structure of the living cell. Since its introduction in 1998, the P system model has been intensively studied and developed: many variants of membrane systems have been proposed, a research monograph (Păun 2002) has been published and regular collective volumes are annually edited—a comprehensive bibliography of P systems can be found at The P Systems Web (2008). The most investigated membrane computing

---

F. Ipate (✉)  
Department of Computer Science, The University of Pitesti, Str Targu din Vale 1, 110040 Pitesti,  
Romania  
e-mail: florentin.ipate@ifsoft.ro

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

topics are related to the computational power of different variants, their capabilities to solve hard problems, like NP-complete ones, decidability, complexity aspects and hierarchies of classes of languages produced by these devices. In the last years there have also been significant developments in using the P systems paradigm to model, simulate and formally verify various systems (Ciobanu et al. 2006). Suitable classes of P systems have been associated with some of these applications and software packages have been developed.

Testing is an essential part of software development and all software applications, irrespective of their use and purpose, are tested before being released. Testing is not a replacement for a formal verification procedure, when the former is also present, but rather a complementary mechanism to increase the confidence in software correctness (Holcombe and Ipaté 1998). Although formal verification has been applied to different models based on P systems (Andrei et al. 2007), testing is completely neglected in this context. The main testing strategies involve either (1) knowing the specific function or behaviour a product is meant to deliver (black-box testing) or (2) knowing the internal structure of the product (white-box testing). In black-box testing, the test generation may be based on a formal specification or model, in which case the process could be automated. There is a large class of formal models used in software specification: finite state machines, Petri nets, process algebras, Z, VDM, etc. We can add now P systems as a formal approach (Ciobanu et al. 2006) to specifying various applications in linguistics, graphics and, more recently, biology, especially for defining signalling pathways.

A recent paper, Gheorghé and Ipaté (2008) suggests some initial steps toward building a P system testing theory based on rule coverage. In this paper, we propose an approach to P system testing based on finite state machine conformance techniques. Of the many variants of P systems that have been defined, we consider cell-like P systems which use non-cooperative transformation and communication rules (Păun 2002). We will also briefly discuss other classes of P systems that can be used within this framework.

State based languages are established means of modelling computer systems and powerful techniques for testing from finite state machine specifications exist (Lee and Yannakakis 1996). Given a finite state machine specification and an implementation, which is a “black box” for which we can only observe its behaviour, we want to test whether the implementation under test (IUT) conforms to the specification; this is called *conformance testing* or *fault detection* and a finite set of sequences that solves this problem is called a *test suite*.

In the case of a P system specification, it is not generally possible to (dis)prove the equivalent behaviour of the specification and IUT, as this is not decidable. However, the implementation will always be finite (i.e. consisting of a finite sequence of steps) and so it would be possible to model it by a finite state machine; consequently, testing techniques based on this formalism can be used. However, key issues still remain to be addressed: defining a finite state model which correctly approximates the behaviour of the P system and generating appropriate test sequences.

This paper shows that a (minimal) *deterministic finite cover automaton* (DFCA) (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) provides the right (finite) approximation for the computation of a P system. Furthermore, based on previous authors' results on DFCA testing (the *W*-method for finite cover automata; Ipaté 2006, 2008), the paper provides a procedure for generating test sets directly from the P system specification (without explicitly constructing the minimal DFCA model).

The paper is structured as follows. Section 2 provides basic concepts and results to be used later in the paper. Section 3 briefly presents the  $W$ -method (for test generation from finite automata) and shows how it can be adapted to DFCA testing (i.e. when there is an upper bound on the length of the strings considered). Section 4 shows how a finite computation of a P system can be approximated by a finite state model while the next section provides the test generation procedure. Finally, conclusions are drawn and further work is outlined in the last section.

## 2 Preliminaries

Before proceeding, we introduce the notations used in the paper. For an alphabet  $V = \{a_1, \dots, a_p\}$ ,  $V^*$  denotes the set of all strings (sequences) over  $V$ . For a string  $u \in V^*$ ,  $|u|$  denotes the length (number of symbols) of  $u$ ; in particular  $|\lambda| = 0$ , where  $\lambda$  is the empty string.  $V^n$  denotes the set of all strings of length  $n$ ,  $n \geq 0$ , with members in the alphabet  $V$  and  $V[n] = \bigcup_{0 \leq i \leq n} V^i$ . For a string  $u \in V^*$ ,  $|u|_{a_i}$  denotes the number of  $a_i$  occurrences in  $u$ . For a string  $u$  we associate a vector of non-negative integer values  $(|u|_{a_1}, \dots, |u|_{a_p})$ . We denote this by  $\Psi_V(u)$ .

### 2.1 P systems

A basic cell-like P system is defined as a hierarchical arrangement of membranes identifying corresponding regions of the system. With each region there are associated a finite multiset of objects and a finite set of rules; both may be empty. A multiset is either denoted by a string  $u \in V^*$ , where the order is not considered, or by  $\Psi_V(u)$ . The following definition refers to one of the many variants of P systems, namely cell-like P system, which uses non-cooperative transformation and communication rules (Păun 2002). We will call these processing rules. Since now onwards we will refer to this model as simply P system.

**Definition 1** A P system is a tuple  $\Pi = (V, \mu, w_1, \dots, w_n, R_1, \dots, R_n)$ , where

- $V$  is a finite set, called *alphabet*;
- $\mu$  defines the membrane structure; a hierarchical arrangement of  $n$  compartments called *regions* delimited by *membranes*; these membranes and regions are identified by integers 1 to  $n$ ;
- $w_i$ ,  $1 \leq i \leq n$ , represents the initial multiset occurring in region  $i$ ;
- $R_i$ ,  $1 \leq i \leq n$ , denotes the set of processing rules applied in region  $i$ .

The membrane structure,  $\mu$ , is denoted by a string of left, [, and right, ], brackets, each with the label of the membrane it points to;  $\mu$  also describes the position of each membrane in the hierarchy. For instance, a structure of three membranes in which membrane 1 contains membranes 2 and 3 can be described by either  $[_1[_2[_3]_3]_2]_1$  or  $[_1[_3]_3[_2]_2]_1$ . The rules in each region have the form  $a \rightarrow (a_1, t_1) \dots (a_m, t_m)$ , where  $a$ ,  $a_i \in V$ ,  $t_i \in \{in, out, here\}$ ,  $1 \leq i \leq m$ . When such a rule is applied to a symbol  $a$  in the current region, the symbol  $a$  is replaced by the symbols  $a_i$  with  $t_i = here$ ; symbols  $a_i$  with  $t_i = out$  are sent to the outer region or outside the system when the current region is the external compartment and symbols  $a_i$  with  $t_i = in$  are sent into one of the regions contained in the current one, arbitrarily chosen. In the following definitions and examples all the symbols  $(a_i, here)$  are used as  $a_i$ . The rules are applied in maximally parallel mode which means that they are used in all the regions in the same time and in each region all the symbols that may be processed, must be.

A configuration of the P system  $\Pi$ , is a tuple  $c = (u_1, \dots, u_n)$ , where  $u_i \in V^*$ , is the multiset associated with region  $i$ ,  $1 \leq i \leq n$ . A derivation of a configuration  $c_1$  to  $c_2$  using the maximal parallelism mode is denoted by  $c_1 \Longrightarrow c_2$ . In the set of all configurations we will distinguish terminal configurations;  $c = (u_1, \dots, u_n)$  is a terminal configuration if there is no region  $i$  such that  $u_i$  can be further derived.

## 2.2 Finite automata

**Definition 2** A *deterministic finite automaton* (abbreviated *DFA*) is a tuple  $M = (A, Q, q_0, F, h)$ , where:

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

The next-state function  $h$  can be extended to a function  $h : Q \times A^* \longrightarrow Q$  defined by:

- $h(q, \lambda) = q, q \in Q$ ;
- $h(q, s a) = h(h(q, s), a), q \in Q, s \in A^*, a \in A$ .

For simplicity the same name  $h$  is used for the next-state function and for the extended function.

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

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

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

A DFA  $M$  is called *minimal* if any DFA that accepts  $L_M$  has at least the same number of states as  $A$ . A DFA  $M$  is minimal if and only if  $M$  is reachable and reduced. Furthermore, the minimal DFA that accepts the same language as a given DFA  $M$  is unique (up to a renaming of the state set). These are well-known results; for proofs and other details see for example (Hopcroft et al. 2006).

## 2.3 Finite cover automata

A *deterministic finite cover automaton (DFCA)* of a finite language  $U$  is a DFA that accepts all sequences in  $U$  and possibly other sequences that are longer than any sequence in  $U$ . The concept was introduced by Cămpeanu et al. (1998, 2001).

**Definition 3** Let  $M = (A, Q, q_0, F, h)$  be a DFA,  $U \subseteq A^*$  a finite language and  $l$  the length of the longest sequence(s) in  $U$ . Then  $M$  is called a *deterministic finite cover automaton (DFCA)* of  $U$  if  $L_M \cap A[l] = U$ .

A *minimal DFCA* of  $U$  is a DFCA of  $U$  having the least number of states. Any DFA that accepts  $U$  is also a DFCA of  $U$  and so the size (number of states) of a minimal DFCA of  $U$

cannot exceed the size of the minimal DFA that accepts  $U$ . On the other hand, a minimal DFCA of  $U$  may have considerably fewer states than the minimal DFA that accepts  $U$  (as also shown by examples in this paper).

In the remainder of this section we provide the necessary concepts for characterizing and constructing a minimal DFCA. These are largely from Câmpeanu et al. (1998) and Körner (2003).

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

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

Recall that a minimal DFA that accepts  $U$  is a DFA 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.

**Definition 4** Let  $M = (A, Q, q_0, F, h)$  be a reachable DFA. States  $q_1$  and  $q_2$  are said to be similar, written  $q_1 \sim q_2$  if  $q_1$  and  $q_2$  are  $A[j]$ -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  $q_1 \approx q_2$ .

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

**Definition 5** Let  $M = (A, Q, q_0, F, h)$  be a reachable DFA.  $(Q_i)_{1 \leq i \leq n}$  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  $q_1 \approx q_2$ .

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** (Körner 2003) *Let  $M = (A, Q, q_0, F, h)$  be a reachable DFCA of  $U$  and let  $(Q_i)_{1 \leq i \leq n}$  be a SSD of  $Q$ . For every  $i$ , choose  $q_i$  such that  $level(q_i) = \min\{level(q) \mid q \in Q_i\}$ . Define  $M' = (A, Q', q_0', F', h')$  by  $Q' = \{Q_1, \dots, Q_n\}$ ,  $q_0' = [q_0]$ ,  $F' = \{[q] \mid q \in F\}$  and  $h'(Q_i, a) = [h(q_i, a)]$  for all  $i$ ,  $1 \leq i \leq n$ , and  $a \in A$ . Then  $M'$  is a minimal DFCA of  $U$ .*

As similarity is not an equivalence relation, the SSD may not be unique and, thus, there may be more than one DFCA of the same finite language  $U$  (unlike in the case in which the acceptance of the precise language is required). Several algorithms for constructing a minimal DFCA exist (Câmpeanu et al. 1998, 2001, 2002, 2006; Körner 2002, 2003; Păun et al. 2000). The time complexity of these algorithms is polynomial in the number of states  $n$  of the original automaton; the best known algorithm (Körner 2002) requires  $O(n \log n)$  time. Interestingly, García and Jose (2004) note that the minimization of DFCA can be approached as an inference problem, which had been solved several years earlier.

### 3 The $W$ -method for testing finite cover automata

In this section we present the  $W$ -method for generating test suites from finite state specifications. We first give the original form of the method and then we show how it can be adapted for testing based on finite cover automata.

Suppose we have a DFA  $M$ . In conformance testing, it is assumed that the IUT can be modelled by an unknown DFA  $M'$  belonging to a known set, called the *fault domain*, identified by the assumptions which can be made about the implementation (e.g. the DFA model of the implementation may have incorrect transitions—when the next state(s) of one or more transitions is (are) incorrect—missing or extra states). There are a number of more or less realistic assumptions that one can make about the form and size of the implementation model  $M'$  and these, in turn, give rise to different techniques for generating test suites (Lee and Yannakakis 1996). One of the least restrictive assumptions refers to the number of states in  $M'$  and is the basis for the  $W$ -method (Chow 1978): the difference between the number of states of the implementation model and that of the specification has to be at most  $\sigma$ , a non-negative integer estimated by the tester.

The  $W$ -method involves the selection of two sets of input sequences, a state cover  $S$  and a characterization set  $W$  defined as follows:

**Definition 6**  $S \subseteq A^*$  is called a *state cover* of  $M$  if  $\lambda \in S$  and for every state  $q \in Q$  there exists  $s \in S$  that reaches  $q$  (i.e.  $h(q_0, s) = q$ ).

**Definition 7**  $W \subseteq A^*$  is called a *characterization set* of  $M$  if  $\lambda \in W$  and every two distinct states of  $M$  are  $W$ -distinguishable.

It is assumed that the specification  $M$  is minimal (if not it can be minimized beforehand) and so a state cover and a characterization set exist. Once  $S$  and  $W$  have been constructed, a test suite is generated using the formula

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

**Theorem 2** (Chow 1978; Balanescu et al. 2003) *For every DFA  $M'$  which has at most  $\sigma$  more states than  $M$ ,  $L_M = L_{M'}$  if and only if  $L_M \cap U_\sigma = L_{M'} \cap U_\sigma$ .*

The idea is that the set  $P = S \cup SA$  (usually called a *transition cover* of  $M$ ) ensures that all the states and all the transitions of  $M$  are also present in  $M'$  and the set  $A[\sigma]W$  ensures that the destination state of each transition of  $M'$  is also correct. Notice that the latter contains  $W$  and also all sets  $A^i W$ ,  $1 \leq i \leq \sigma$ . This ensures that  $M'$  does not contain extra states; if there were up to  $\sigma$  extra states, each of them would be reached by some input sequence of up to length  $\sigma$  from the existing states.

The result above shows that  $W$ -method not only provides a rigorous way of obtaining the test suits, but also guarantees the equivalence of the two mechanisms considered.

The problem of conformance testing can also be formulated for bounded sequences: given a DFA specification  $M$  and an integer  $l \geq 1$  (the upper bound), 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  $A[l]$ . If  $L_M$  contains at least one sequence of length  $l$  then  $M$  is a DFCA of  $L_M \cap A[l]$  and so the test suite will check whether the IUT model  $M'$  is also a DFCA of  $L_M \cap A[l]$ . Recently, it was shown that the underlying idea of the  $W$ -method can also be applied in the case of bounded sequences, provided that the sets  $S$  and  $W$  used in the construction of the test suite contain sequences of *minimum*

length that reach or distinguish the states of  $M$  (Ipatе 2006, 2008).  $S$  and  $W$  thus defined are called a proper state cover and strong characterization set, respectively.

**Definition 8**  $S \subseteq A^*$  is called a *proper state cover* of  $M$  if for every state  $q$  of  $M$  there exists  $s \in S$  such that  $h(q_0, s) = q$  and  $|s| = \text{level}(q)$ .

**Definition 9**  $W \subseteq A^*$  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  $q_2$  are  $A[j]$ -distinguishable then  $q_1$  and  $q_2$  are  $(W \cap A[j])$ -distinguishable.

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

A test suite for the bounded case is generated using the formula

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

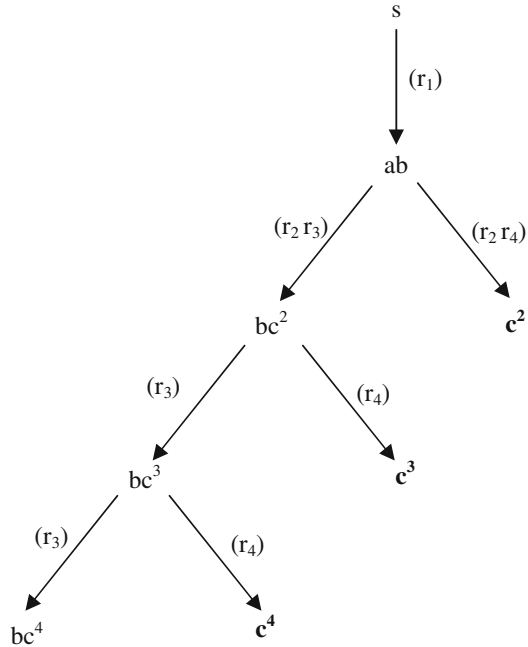
**Theorem 3** (Ipatе 2008) *for every DFA  $M'$  which has at most  $\sigma$  more states than  $M$ ,  $L_M \cap A[l] = L_{M'} \cap A[l]$  if and only if  $L_M \cap V_\sigma = L_{M'} \cap V_\sigma$ .*

### 4 Modelling P systems using finite automata

In order to apply the  $W$ -method (or any other finite state machine based method) to a P system specification, an appropriate DFA model of the specification needs to be defined first. In general, the computation of a P system cannot be fully modelled by a finite automaton and so an *approximate* DFA model will be sought. The problem will be addressed in two steps. Firstly, the derivation tree of a P system will be represented as a finite automaton. In order to guarantee the finiteness of this process, an upper bound  $k$  on the length of any derivation will be set and only computations of maximum  $k$  transitions will be considered at a time (otherwise the derivation may continue forever and an infinite number of nodes and arcs will be needed). Secondly, a *minimal* model, that preserves the required behaviour, will be defined on the basis of the aforementioned derivation tree.

For the sake of clarity, we consider first the derivation tree of a one compartment P system  $\Pi = (V, \mu, w, R)$ , where  $\mu = [1]_1$ . A configuration of  $\Pi$  can change as a result of the application of one or more rules, in parallel. As only computations of maximum  $k$  steps are considered, for every rule  $r_i \in R$  there will be some  $N_i$  such that, in any step,  $r_i$  can be applied at most  $N_i$  times. Thus, the derivation tree of a P system  $\Pi$  with rules  $R = \{r_1, \dots, r_m\}$  can be described by a DFA  $Dt_k$  over an alphabet  $A_k \subseteq \{(r_1^{i_1} \dots r_m^{i_m}) \mid 0 \leq i_1 \leq N_1, \dots, 0 \leq i_m \leq N_m\}$ ; when position  $i_j$  has the exponent 0 then it will be removed from this notation. Moreover, the minimal alphabet  $A_k$  will be chosen; that is  $A_k$  will contain only the transition labels that actually appear in the derivation tree. The states of  $Dt_k$  correspond to the nodes of the tree. For testing purposes we will consider all the states as final. It is implicitly assumed that a non-final “sink” state  $q_{sink}$  that receives all “rejected” transitions, also exists. Consider, for example,  $\Pi_1 = (V, \mu, w_1, R_1)$  defined by  $V = \{s, a, b, c\}, \mu = [1]_1, w_1 = s, R_1 = \{r_1 : s \rightarrow ab, r_2 : a \rightarrow c, r_3 : b \rightarrow bc, r_4 : b \rightarrow c\}$ . The derivation tree of  $\Pi_1$ , for  $k = 4$ , is as represented in Fig. 1. The terminal configurations (i.e. for which there is no derivation) are in bold (however, these should not be confused with the final states of the DFA—as stated above, all nodes in the tree are considered final states).

**Fig. 1** Derivation tree for  $\Pi_1$  when  $k = 4$



The construction can be generalized for a multiple compartment P system  $\Pi = (V, \mu, w_1, \dots, w_n, R_1, \dots, R_n)$ . Again, only computations of maximum  $k$  steps are considered. In this case, the derivation tree can be described by a DFA over  $A_k \subseteq \{(r_{1,1}^{i_1} \dots r_{1,m_1}^{i_{m_1}}, \dots, r_{n,1}^{i_n} \dots r_{n,m_n}^{i_{m_n}}) \mid 0 \leq i_{j,p} \leq N_{j,p}, 1 \leq j \leq n, 1 \leq p \leq m_j\}$ , where  $N_{j,p}$  is the maximum number of times the rule  $r_{j,p}$ , from compartment  $j$  can be applied in one derivation step. As above, the alphabet  $A_k$  will contain only the transition labels that appear in the derivation tree. Consider, for example, the two compartment P systems  $\Pi_2 = (V', \mu', w'_1, w'_2, R'_1, R'_2)$  with  $V' = \{s, a, b, c\}$ ,  $\mu' = [1[2]_2]_1$  (i.e. region 1 contains 2),  $w'_1 = s, w'_2 = \lambda, R'_1 = \{r_1 : s \rightarrow sa(b, in), r_2 : s \rightarrow ab, r_3 : b \rightarrow a, r_4 : a \rightarrow c\}, R'_2 = \{r_1 : b \rightarrow bc, r_2 : b \rightarrow c\}$ . The derivation tree of  $\Pi_2$  for  $k = 3$  is as represented in Fig. 2. Due to space constraints, in Fig. 2 the derivations from nodes  $(sac, bc)$  and  $(abc, c)$  (given in italics) are omitted; these nodes are similar (as DFA states) to nodes at higher levels in the hierarchy:  $(sac, bc) \sim (sa, b)$  and  $(abc, c) \sim (ab, \lambda)$ .

As  $Dt_k$  is a DFA over  $A_k$ , one can find the minimal DFA that accepts exactly the language  $L_{Dt_k}$  defined by  $Dt_k$ . However, as only sequences of at most  $k$  transitions are considered, it is irrelevant how the constructed automaton will behave for longer sequences. Consequently, a DFCA of  $L_{Dt_k}$  will be sufficient.

Consider now how Theorem 1 can be applied to obtain a minimal DFCA of  $L_{Dt_k}$ . Let  $Q_k$  denote the nodes of the derivation tree (thus the state set of  $Dt_k$  is  $Q_k \cup \{q_{sink}\}$ ) and  $q_0 \in Q_k$  the root node. Let  $\leq$  be a total order on  $Q_k$  such that  $level(q_1) \leq level(q_2)$  whenever  $q_1 \leq q_2$  and denote  $q_1 < q_2$  if  $q_1 \leq q_2$  and  $q_1 \neq q_2$ . That is, the node at the superior level is before the node at the inferior level; if the nodes are at the same level then their order can be arbitrarily chosen. Define  $P_k = \{q \in Q_k \mid \neg \exists q' \in Q_k \cdot q' \sim q, q' < q\}, [q] = \{q' \in Q_k \mid q' \sim q \wedge \neg \exists q'' \in P_k \cdot q'' \sim q', q'' < q\}$  for every  $q \in P_k$  (i.e.  $[q]$  denotes the set of all states  $q'$  for which  $q$  is the minimum state similar to  $q'$ ) and  $[q_{sink}] = \{q_{sink}\}$ . Then we have the following result.

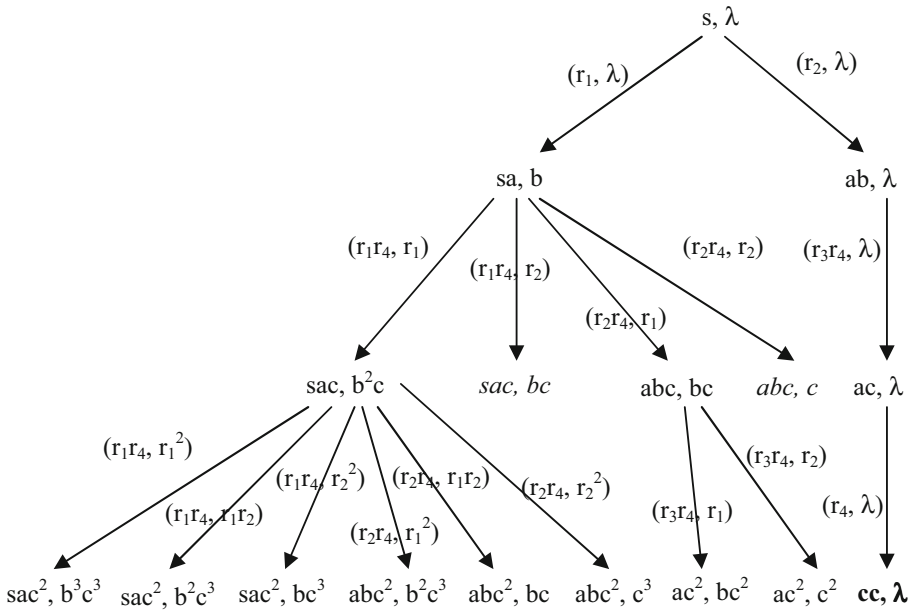


Fig. 2 Derivation tree for  $\Pi_2$  when  $k = 3$

**Theorem 4** Define  $M = (A_k, Q, [q_0], F, h')$  by  $Q = \{[q] | q \in P_k \cup \{q_{sink}\}\}$ ,  $F = \{[q] | q \in P_k\}$  and  $h'([q], a) = [h(q, a)]$  for all  $q \in P_k \cup \{q_{sink}\}$  and  $a \in A_k$ .

Then  $M$  is a minimal DFCA of  $L_{Dt_k}$ .

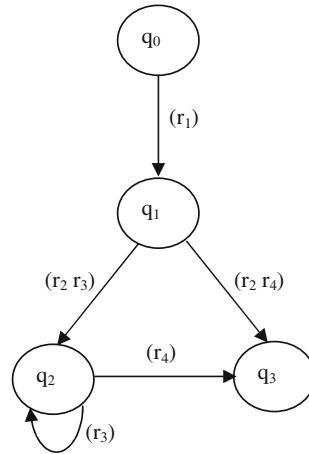
*Proof* Let  $q \in Q_k$ ,  $q_1, q_2 \in [q]$ . Since  $q_1 \sim q, q_2 \sim q, level(q_1) \leq level(q)$  and  $level(q_2) \leq level(q), q_1 \sim q_2$ . Then  $\{[q]\}_{q \in P_k \cup \{q_{sink}\}}$  is a SSD of  $Dt_k$  (the other conditions of Definition 5 follow directly from the definition of  $[q]$ ). Thus, by Theorem 1,  $M$  is a minimal DFCA of  $L_{Dt_k}$ .

Consider, for example,  $L_{Dt_k}$  defined by the derivation tree (of the one compartment P system example  $\Pi_1$  defined earlier) represented in Fig. 1. As the system configurations that label the nodes are pairwise distinct, the set  $Q_k$  can be defined by these configurations. Then  $Q = \{[s], [ab], [bc^2], [c^2], [q_{sink}]\}$ ,  $[s] = \{s, bc^4, c^4\}$ ,  $[ab] = \{ab\}$ ,  $[bc^2] = \{bc^2, bc^3\}$ ,  $[c^2] = \{c^2, c^3\}$ . A minimal DFCA of the language  $L_{Dt_k}$  is as represented in Fig. 3 (the “sink” state is not explicitly shown).

Not only a minimal DFCA of  $L_{Dt_k}$  may have (significantly) less states than the minimal DFA that accepts  $L_{Dt_k}$ , but it also has an important property: at limit, if the entire computation (i.e. for derivation sequences of any length) of a P system can be described by a DFA  $M$ , then  $M$  is actually a DFCA of  $L_{Dt}$  for  $k$  sufficiently large. This is formally proven next.

Suppose there exists a DFA  $M'$  over the input alphabet  $A$  and there exists  $n \geq 1$  such that, for every  $k \geq n$ ,  $A_k = A$  and  $L_{M'} \cap A[k] = L_{Dt_k} \cap A[k]$ . Clearly,  $L_{M'} = L_{M'}$  for any DFA  $M''$  that satisfies the above property. We denote by  $M$  the minimal DFA that accepts  $L_{M'}$  and we say that  $\{Dt_k\}$  converges to  $M$ .

**Fig. 3** Minimal DFCA for  $\Pi_1$  when  $k = 4$



**Theorem 5** Suppose that  $\{Dt_k\}$  converges to  $M$ . Then there exists  $m \geq 1$  such that for every  $k \geq m$ ,  $M$  is the unique (up to a renaming of the state set) minimal DFCA that accepts  $L_{Dt_k}$ .

*Proof* Since  $\{Dt_k\}$  converges to  $M$  there exists  $n \geq 1$  such that for every  $k \geq n$ ,  $A_k = A$  and  $L_M \cap A[k] = L_{Dt_k} \cap A[k]$ . Thus  $M$  is a DFCA of  $L_{Dt_k}$  for every  $k \geq n$ . Let  $n_M$  be the number of states of  $M$ . Then, by Theorem 2 (the  $W$ -method), there exists a finite set of input sequences  $X \subseteq A^*$  such that for every DFA  $M'$  over input alphabet  $A$  with at most  $n_M$  states,  $L_M = L_{M'}$  whenever  $L_M \cap X = L_{M'} \cap X$ . Let  $m$  be the length of the longest sequence(s) in  $X$ . We prove that, for every  $k \geq m$ ,  $M$  is the unique minimal DFCA of  $L_{Dt_k}$ . Let  $k \geq m$  and let  $M''$  be a minimal DFCA of  $L_{Dt_k}$ . Then  $L_M \cap A[k] = L_{M''} \cap A[k]$  and since  $X \subseteq A[k]$ ,  $L_M \cap X = L_{M''} \cap X$ . Thus  $L_M = L_{M''}$  and since  $M$  and  $M''$  are minimal DFAs, it follows that they are identical (up to a renaming of the state set).

Consider the one compartment P system,  $\Pi_1$  defined above.  $\{Dt_k\}$  converges to  $M$ , the minimal DFCA, for  $k = 4$ , represented in Fig. 3. Take, for instance, the nodes  $u_1 = bc^2$  and  $u_2 = bc^3$  from the derivation tree represented in Fig. 1. As the derivation from  $u_1$  is identical to the derivation from  $u_2$  (i.e. any sequence  $s \in A^*$  (of any length) that can be applied to  $u_1$  can also be applied to  $u_2$  and vice versa)  $u_1$  and  $u_2$  should be represented by the same state of a DFA that models the computation of the P system. This is the case when the DFA model considered is a minimal DFCA of  $L_{Dt_k}$  (since  $u_2 \in [u_1]$ ). On the other hand,  $u_1$  and  $u_2$  will be associated with distinct states in the minimal DFA that accepts  $L_{Dt_k}$  (they are not equivalent states since they appear at different levels in the derivation tree).

### 5 Generating test sequences from a P system

Suppose we have a P system specification  $\Pi$  and an implementation  $I$  of  $\Pi$ . As discussed above, given an upper bound  $k$ , the computation of  $\Pi$  can be approximated by a minimal DFCA  $M$  of the language  $L_{Dt_k}$  defined by the derivation tree. Then the  $W$ -method can be applied to check whether  $I$  conforms to  $\Pi$  for transition sequences of length up to  $k$ . This involves the selection of a proper state cover  $S$  and of a strong characterization set  $W$  of  $M$ .

In this section, we provide a procedure for constructing these two sets. Furthermore, we do not assume that an explicit representation of  $M$  has been actually obtained; this would

involve the construction of the (entire) derivation tree  $Dt_k$  and the application of one of the existing minimization algorithms for DFCA, whose time complexity is polynomial in the number of states of  $Dt_k$ . Instead, we show how  $S$  and  $W$  can be directly derived from  $\Pi$ ; it will transpire that, for this purpose, it might not be even necessary to construct the entire derivation tree  $Dt_k$ .

The construction of  $W$  is based on the observation that, whenever two states of  $Dt_k$  are dissimilar, it is sufficient to arbitrarily select any transition label (from either of them) in order to distinguish between them.

**Theorem 6** *Let  $q$  and  $q'$  be states of  $Dt_k$  for some  $k \geq 1$ . If  $A_k \cap L_M^q \cap L_M^{q'} \neq \emptyset$  then  $q \sim q'$ .*

*Proof* Let  $c = (u_1, \dots, u_n)$  and  $c' = (u'_1, \dots, u'_n)$  be the configurations corresponding to  $q$  and  $q'$ , respectively and let  $a \in A_k \cap L_M^q \cap L_M^{q'}$ . Then  $u_i = x_i y_i$  and  $u'_i = x_i y'_i$ ,  $x_i, y_i, y'_i \in V^*$ , such that  $x_i$  is the multiset of symbols consumed by the rules in  $a$ ,  $1 \leq i \leq n$ . As the rules are applied in maximal parallel mode, the sets of symbols that have not been consumed,  $y_i$  and  $y'_i$ , will not affect the future computation of the system. Thus  $(u_1, \dots, u_n)$  and  $(u'_1, \dots, u'_n)$  will be derived identically. Hence  $q \sim q'$ .

A corollary of the above theorem is that, if there are no new transition labels between level  $i$  and level  $i + 1$  in the derivation tree, there will be no new dissimilar states at any level  $j \geq i + 1$  and so the DFCA model will converge.

**Corollary 1** *If there exists  $i \geq 1$  such that  $A_i = A_{i+1}$  then  $\{Dt_k\}$  converges.*

*Proof* Let  $M$  be a minimal DFCA of  $Dt_i$  constructed as in Theorem 4 and let  $k \geq i + 1$ . Since the  $(i + 1)$ th level of  $Dt_k$  contains no new transition labels, by Theorem 6, any state at the  $(i + 1)$ th level will be similar to some state at a higher level in the tree. From this, it follows that, for any  $j \geq i + 1$ , any state at level  $j$  is similar to some state at level  $j' \leq i$ . Then, by Theorem 4,  $M$  is a minimal DFCA of  $Dt_k$ . Thus  $\{Dt_k\}$  converges to  $M$ .

In general, a (proper) state cover of a minimal DFA can be obtained by constructing a testing tree in a breadth-first fashion (Chow 1978). Let  $M$  be the minimal DFCA of  $Dt_k$  constructed as in Theorem 4, in which the order of the nodes at the same level in the derivation tree is from left to right. On the basis of Theorem 6 and Corollary 1, we will construct a testing tree  $T$  of  $M$ , without explicitly constructing  $M$ . A proper state cover  $S$  and a strong characterization set  $W$  of  $M$  will then be derived from  $T$ .

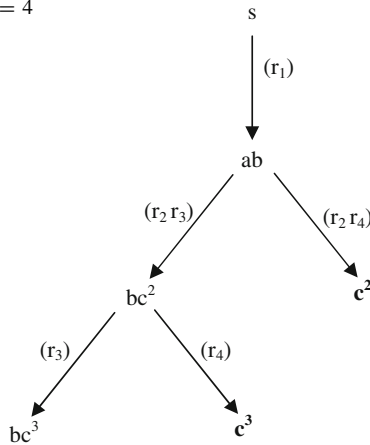
The algorithm is presented in what follows. Analogously to the derivation tree  $Dt_k$ , the nodes of  $T$  are labelled with configurations of  $\Pi$  and the branches of  $T$  are labelled with symbols from  $A_k$ .

- The root of  $T$  is labelled with the initial configuration of  $\Pi$ . This is the level 0 of  $T$ .
- Suppose we have already built  $T$  to a level  $i < k$ . Then the  $(i + 1)$ th level is built by examining nodes at the  $i$ th level from left to right. A node labelled  $c$  at the  $i$ th level is called a *leaf* if either  $c$  is a terminal configuration ( $c$  cannot be derived any further) or  $c$  can be derived using a symbol  $a \in A_k$  that already labels some branch of the tree. If the node is not a leaf then a branch is attached for each derivation of  $c$ .

The algorithm terminates when either some level  $i < k$  contains only leaf nodes or level  $k$  has been reached. By definition, all nodes at level  $k$  are leaf nodes.

We denote by  $X$  the set of all non-leaf nodes of  $T$ . We also choose a node  $y$  (if it exists) as follows: we denote by  $Y$  the set of all leaf nodes which are labelled by terminal configurations of  $\Pi$  and are not at the  $k$ th level of the hierarchy; if  $Y$  is not empty,  $y$  is the

**Fig. 4** Test tree for  $\Pi_1$  and  $k = 4$



node in  $Y$  at the highest level (if there are two or more such nodes, one is arbitrarily chosen). Then  $S$  and  $W$  are constructed as follows:

- $S$  will contain all partial paths in  $T$  that lead to nodes from  $X$ ; furthermore, if  $Y \neq \emptyset$ ,  $S$  will also contain the path that leads to  $y$ .
- $\lambda \in W$  and, for each non-leaf node  $x \in X$ ,  $W$  will contain the label of one transition that emerges from  $x$ .<sup>1</sup>

The testing tree for the one compartment P system  $\Pi_1$  defined earlier and  $k = 4$  is given in Fig. 4. As all configurations at the third level are either terminal configurations or can be derived using symbols from higher level branches, all nodes at the third level are leaves.  $S_1 = \{\lambda, (r_1), (r_1) (r_2 r_3)\}$  is the set of all partial paths that lead to non-leaf nodes and  $s_2 = (r_1) (r_2 r_4)$  is the shortest path that leads to a leaf node which is labelled by a terminal configuration. Thus  $S = S_1 \cup \{s_2\}$ . On the other hand,  $(r_1)$ ,  $(r_2 r_3)$  and  $(r_3)$ , respectively, are labels of transitions from each of the 3 non-leaf nodes. Thus  $W = \{\lambda, (r_1), (r_2 r_3), (r_3)\}$ .  $S$  and  $W$  are a proper state cover and a strong characterization set, respectively, of the minimal DFCA represented in Fig. 3.

**Theorem 7** For every  $k \geq 1$ , there exists  $M$  a minimal DFCA of  $L_{Dt_k}$  such that  $S$  and  $W$  thus constructed are a proper state cover and a strong characterization set of  $M$ , respectively.

*Proof* Let  $Q_k$  and  $Q'_k$  be the nodes of  $Dt_k$  and  $T$ , respectively. With slight abuse of notation  $Q'_k \subseteq Q_k$ . By Theorem 6 and Corollary 1, every node in  $Q'_k \setminus Y$  is similar to some node in  $X$ . Furthermore, every node in  $Y$  is similar to  $y$  and every node at the  $k$ th level of  $T$  (if this level exists) is similar to the initial node. Let  $M$  be the minimal DFCA of  $Dt_k$  constructed as in Theorem 4 in which the order of the nodes at the same level is from left to right. Then, by construction,  $S$  is a proper state cover of  $M$ . The fact that  $W$  is a strong characterization set of  $M$  follows directly from Theorem 6.

<sup>1</sup>  $\lambda$  distinguishes the (non-final) “sink” state from the other (final) states of  $M$ . On the other hand, according to the results in Balanescu et al. (2003), since  $\lambda \in W$ , the “sink” state needs not be reached by  $S$ .

## 6 Conclusions

This paper proposes an approach to P system conformance testing using finite state machine based techniques. The computation of the P system specification is approximated by a DFCA, (constructed on the basis of the derivation tree) and a procedure for generating test sets (for the DFCA model) directly from the P system specification is provided.

Of the many variants of P systems that have been defined, the paper considers cell-like P systems which use non-cooperative transformation and communication rules. The same methodology is applicable either directly to tissue P systems using the same type of rules or with few adequate modifications for other classes of cell-like or tissue P systems relying on other communication rules, like symport/antiport, or using catalysts, promoters/inhibitors and general non-cooperative rules. The semantics regarding the use of these rules, in the current case, maximal parallelism, does not affect the methodology. Further work will concentrate on more complex types of P systems, like those with active membranes or population P systems with bonding rules that involve a dynamic structure of the system. Other approaches, such as mutation based test generation [Nilsson et al. 2006], may also be considered.

**Acknowledgments** The authors would like to thank the anonymous referees for their very helpful comments that allowed the improvement of this paper.

## References

- Andrei O, Ciobanu G, Lucanu D (2007) A rewriting logic framework for operational semantics of membrane systems. *Theoretical Comput Sci* 373(3):163–181
- Balanescu T, Gheorghe M, Ipate F (2003) Formal black box testing for partially specified deterministic finite state machines. *Found Comp Decision Syst* 28(1):17–28
- Câmpeanu C, Santean N, Yu S (1998) Minimal cover-automata for finite languages. In: *Workshop on Implementing Automata. Lecture notes in computer science*, vol 1660. Springer, Berlin, pp 43–56
- Câmpeanu C, Santean N, Yu S (2001) Minimal cover-automata for finite languages. *Theoretical Comput Sci* 267(1–2):3–16
- Câmpeanu C, Păun A, Yu S (2002) An efficient algorithm for constructing minimal cover automata for finite languages. *Int J Found Comp Sci* 13(1):83–97
- Câmpeanu C, Păun A, Smith JR (2006) Incremental construction of minimal deterministic finite cover automata. *Theoretical Comput Sci* 363(2):135–148
- Chow TS (1978) Testing software design modeled by finite-state machines. *IEEE Trans Softw Eng* 4(3):178–187
- Ciobanu G, Păun Gh, Pérez-Jiménez MJ (eds) (2006) *Applications of membrane computing*. Springer, Berlin
- García P, Jose Ruiz (2004) A note on the minimal cover-automata for finite languages. *Bull EATCS* 83:193–194
- Gheorghe M, Ipate F (2008) On testing P systems. In: *Proceedings of WMC*, pp 173–188
- Holcombe H, Ipate F (1998) *Correct systems—building business process solutions*. Springer, Berlin
- Hopcroft JE, Motwani R, Ullman JD (2006) *Introduction to automata theory, languages, and computation* (3rd edition). Addison-Wesley Longman
- Ipate F (2006) Bounded sequence testing from non-deterministic finite state machines. In: *Proceedings of TesCom 2006*, pp 55–70
- Ipate F (2008) Bounded sequence testing from deterministic finite state machines (submitted)
- Körner H (2002) On minimizing cover automata for finite languages in  $O(n \log n)$  time. In: *CIAA 2002. Lecture notes in computer science*, vol 2608. Springer, Berlin, pp 117–127
- Körner H (2003) A time and space efficient algorithm for minimizing cover automata for finite languages. *Int J Found Comp Sci* 14(6):1071–1086
- Lee D, Yannakakis M (1996) Principles and methods of testing finite state machines—a survey. In: *Proceedings of the IEEE 1996*, vol 84, pp 1090–1126

- Nilsson R, Offutt J, Mellin J (2006) Test case generation for mutation-based testing of timeliness. *Electr Notes Theor Comput Sci* 164(4):97–114
- Păun A, Santean N, Yu S (2000) An  $O(n^2)$  algorithm for constructing minimal cover automata for finite languages. In: CIAA 2000. Lecture notes in computer science, vol 2088. Springer, Berlin, pp 243–251
- Păun Gh (2000) Computing with membranes. *J Comp System Sci* 61(1):108–143
- Păun Gh (2002) Membrane computing. An introduction. Springer, Berlin
- The P Systems Web Site (2008) <http://ppage.psystems.eu>