On Testing P Systems

Marian Gheorghe[1] and Florentin Ipate[2]

[1] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk

[2] Department of Computer Science, Faculty of Mathematics and Computer Science
The University of Piteşti
Str Targu din Vale 1, 110040 Piteşti
florentin.ipate@ifsoft.ro

Abstract. This paper presents a basic framework to define testing strategies for some classes of P systems. Techniques based on grammars and finite state machines are developed and some testing criteria are identified and illustrated through simple examples.

1 Introduction

In 1998, Gheorghe Păun initiated the field of research called *membrane computing* with a paper firstly available on the web [17]. Membrane computing, a new computational paradigm, aims at defining computational models which are inspired by the functioning and structure of the living cell. In particular, membrane computing starts from the observation that compartmentalization through membranes is one of the essential features of (eucaryotic) cells. Unlike bacterium, which generally consists of a single intracellular compartment, an eucaryotic cell is sub-divided into distinct compartments with well-defined structures and functions. Further on have been considered other biological phenomena like tissues, colonies of different organisms, various bio-chemical entities with dynamic structure in time and space. Membrane systems, also called *P systems*, consist now of different computational models addressing multiple levels of bio-complexity. There are *cell-like P systems*, relying on the hierarchical structure of the living cells, *tissue-like models*, reflecting the network structure of neurons and other bio-units arranged in tissues or more complex organs, *P colonies* and *population P systems*, drawing inspiration from the organization and behavior of bacterium colonies, social insects and other organisms living together in larger communities (see [18], [19]).

The most basic model and the first introduced, [17], the cell-like paradigm has three essential features: (i) a *membrane structure* consisting of a hierarchical arrangement of several compartments, called *regions*, delimited by *membranes*, (ii) *objects* occurring inside these regions, coding for various simple or more complex chemical molecules or compounds, and (iii) *rules* assigned to the regions of the membrane structure, acting upon the objects inside. In particular, each region is supposed to contain a finite set of rules and a finite multiset (or set) of objects. Rules encode for generic transformation processes involving objects and for transporting them, through membranes, from one region to an adjacent one. The application of the rules is performed in a non-deterministic maximally parallel manner: all the applicable rules that can be used to modify or transport existing objects, must be applied, and this is done in parallel for all membranes. This process abstracts the inherent parallelism that occurs at the cellular level.

Since this model has been introduced, many variants of membrane systems have been proposed, a research monograph [18] has been published and regular collective volumes are annually edited – a comprehensive bibliography of P systems can be found at [19]. The most investigated membrane computing 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 been significant developments in using the P systems paradigm to model, simulate and formally verify various systems [7]. For some of these applications suitable classes of P systems have been associated with and software packages developed. For these models corresponding formal semantics [1] and verification mechanisms [2] have been produced.

There are well-established application areas where the software specifications developed are also delivered together with a model and associated formal verification procedures. All software applications, irrespective of their use and purpose, are tested before being released, installed and used. Testing is not a replacement for a formal verification procedure, when the former is also present, but a necessary mechanism to increase the confidence in software correctness and ultimately a well-known and very well-established stage in any software engineering project [10]. Although formal verification has been applied for different models based on P systems, testing has been completely neglected so far in this context. In this paper we suggest some initial steps towards building a testing framework and its underpinning theory, based on formal grammars and finite state machines, that is associated to software applications derived from P systems specifications. We develop this testing theory based on formal grammars and finite state machines as they are the closest formalisms to P systems and the testing mechanisms for them are well-investigated. Of course, other testing approaches can be considered in this context as well, but all of them require a bigger effort of translation and inevitably difficulties in checking the correctness of this process and in mapping it back to P systems.

The paper is organized as follows: in Section 2 there are introduced basic concepts and definitions; a testing framework based on context-free grammars and finite state machines is built and some examples presented in Sections 3 and 4, respectively; conclusions are drawn in Section 5.

2 Basic Concepts and Notations

For an alphabet $V = \{a_1, ..., a_p\}$, V^* denotes the set of all strings over V. λ denotes the empty string. 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 $(|u|_{a_1}, ..., |u|_{a_p})$. We denote this by $\Psi_V(u)$.

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 [18]. We will call these processing rules and this model simply P system.

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

- V *is a finite set, called* alphabet*;*
- μ *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 rules applied in region* i*.*

The rules in each region have the form $a \rightarrow (a_1, t_1)...(a_m, t_m)$, 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, 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 symbols that may be processed, must be.

A configuration of the P system Π is a tuple $c = (u_1, ..., u_n)$, $u_i \in V^*$, $1 \leq i \leq n$. 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; $c = (u_1, ..., u_n)$ is a terminal configuration if there is no region i such that u_i can be further processed.

The set of all halting configurations is denoted by $L(\Pi)$, whereas the set of all configurations reachable from the initial one (including the initial configuration) is denoted by $S(\Pi)$.

Definition 2. *A* deterministic finite automaton *(abbreviated* DFA*),* M*, is a tuple* (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*, and* $h : Q \times A \longrightarrow Q$ *is the* next-state *function.*

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

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

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

Given $q \in Q$, a sequence of input symbols $s \in A^*$ is said to be accepted by M in q if $h(q, s) \in F$. The set of all input sequences accepted by M in q_0 is called the *language defined (accepted) by M*, denoted $L(M)$.

3 Grammar-Like Testing

Based on testing principles developed for context-free grammars [13], [14], some testing strategies aiming to achieve rule coverage for a P system will be defined and analyzed in this section.

In *grammar engineering*, formal grammars are used to specify complex software systems, like compilers, debuggers, documentation tools, code pre-processing tools etc. One of the areas of grammar engineering is *grammar testing* which covers the development of various testing strategies for software based on grammar specifications. One of the main testing methods developed in this context refers to rule coverage, i.e., the testing procedure tries to cover all the rules of a specification.

In the context of grammar testing it is assumed that for a given specification defined as a grammar, an implementation of it, like a parser, exists and will be tested. The aim is to build a test set, a finite set of sequences, that reveals potential errors in the implementation. As opposed to testing based on finite state machines, where it is possible to (dis)prove the equivalent behavior of the specification and implementation, in the case of general context-free grammars this is no longer possible as it reduces to the equivalence of two such devices, which is not decidable. Of course, for specific restricted classes of context-free grammars there are decidability procedures regarding the equivalence problem and these may be considered for testing purposes as well, but we are interested here in the general case. The best we can get is to cover as much as possible from the languages associated to the two mechanisms, specification and implementation grammars, and this is the role of a test set. We will define such test sets for P systems.

Given a specification G and an implementation G', a test set aims to reveal inconsistencies, like

– *incorrectness* of the implementation G' with respect to the specification language $L = L(G)$, if $L(G') \not\subset L$ and $L \neq L(G')$;
– *incompleteness* of the implementation G' with respect to the specification language $L = L(G)$, if $L \not\subset L(G')$ and $L \neq L(G')$.

We start to develop a similar method in the context of P systems. Although there are a number of similarities between context-free grammars utilized in grammar testing and basic P systems, like those considered in this paper, there are also major differences that pose new problems in defining suitable testing methods. Some of the difficulties that we encounter in introducing some grammar-like testing procedures are related to the main features that define such systems: hierarchical compartmentalization of the entire model, parallel behavior, communication mechanisms, the lack of a non-terminal alphabet.

We define some rule coverage criteria by firstly starting with one compartment P system, i.e., $\Pi = (V, \mu, w, R)$, where $\mu = [_1\ ]_1$. The rule coverage criteria are adapted from [13], [14]. In the sequel, if not otherwise stated, we will consider that the specification and the implementation are given by the P systems Π and Π', respectively. For such a P system Π, we define the following concepts.

Definition 3. *A multiset denoted by $u \in V^*$,* covers *a rule $r : a \rightarrow v \in R$, if there is a derivation $w \Longrightarrow^* xay \Longrightarrow x'vy' \Longrightarrow^* u$; $w, x, y, v, u \in V^*$, $a \in V$.*

If there is no further derivation from u, then this is called a *terminal coverage.*

Definition 4. *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 following one compartment P systems are considered, $\Pi_i, 1 \leq i \leq 4$, having the same alphabet and initial multiset:

$$\Pi_i = (V_i, \mu_i, w_i, R_i), \text{ where:}$$

- $V_1 = V_2 = V_3 = V_4 = \{s, a, b, c\}$;
- $\mu_1 = \mu_2 = \mu_3 = \mu_4 = [_1\ ]_1$;
- $w_1 = w_2 = w_3 = w_4 = s$;
- $R_1 = \{r_1 : s \rightarrow ab,\ r_2 : a \rightarrow c,\ r_3 : b \rightarrow bc,\ r_4 : b \rightarrow c\}$;
- $R_2 = \{r_1 : s \rightarrow ab,\ r_2 : a \rightarrow \lambda,\ r_3 : b \rightarrow c\}$;
- $R_3 = \{r_1 : s \rightarrow ab,\ r_2 : a \rightarrow bcc,\ r_3 : b \rightarrow \lambda\}$;
- $R_4 = \{r_1 : s \rightarrow ab,\ r_2 : a \rightarrow bc,\ r_3 : a \rightarrow c,\ r_4 : b \rightarrow c\}$.

In the sequel for each multiset w, we will use the following vector of non-negative integer numbers $(|w|_s, |w|_a, |w|_b, |w|_c)$.

The sets of all configurations expressed as vectors of non-negative integer numbers, computed by the P systems Π_i, $1 \leq i \leq 4$ are:

- $S(\Pi_1) = \{(1,0,0,0), (0,1,1,0)\} \cup \{(0,0,k,n) \mid k = 0,1; n \geq 2\}$;
- $S(\Pi_2) = \{(1,0,0,0), (0,1,1,0), (0,0,0,1)\}$;
- $S(\Pi_3) = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2)\}$;
- $S(\Pi_4) = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2), (0,0,0,3)\}$.

Test sets for Π_1 satisfying the RC criterion are

- $T_{1,1} = \{(0,1,1,0), (0,0,1,2), (0,0,0,2)\}$ and
- $T_{1,2} = \{(0,1,1,0), (0,0,1,2), (0,0,0,3)\}$,

whereas $T'_{1,1} = \{(0,1,1,0), (0,0,0,2)\}$ and $T'_{1,2} = \{(0,1,1,0), (0,0,1,2)\}$ are not, as they do not cover the rules r_3 and r_4, respectively. Both $T_{1,1}$ and $T_{1,2}$ show the incompleteness of Π_2 with respect to $S(\Pi_1)$ (Π_2 is also incorrect). $T_{1,1}$ does not show the incompleteness of Π_3 with respect to $S(\Pi_1)$, but $T_{1,2}$ does. None of these test sets does show the incompleteness of Π_4.

The sets of terminal configurations expressed as vectors of non-negative integer numbers, computed by the P systems Π_i, $1 \leq i \leq 4$ are:

- $L(\Pi_1) = \{(0,0,0,n) \mid n \geq 2\}$;
- $L(\Pi_2) = \{(0,0,0,1)\}$;
- $L(\Pi_3) = \{(0,0,0,2)\}$;
- $L(\Pi_4) = \{(0,0,0,2),(0,0,0,3)\}$.

A test set for Π_1 satisfying the RTC criterion is $T = \{(0,0,0,3)\}$. As $(0,0,0,3)$ is not in $L(\Pi_2)$ and $L(\Pi_3)$, it follows that Π_2 and Π_3 are incomplete with respect to $L = L(\Pi_1)$. However, the test set does not prove the incompleteness of Π_4.

The examples above show that none of the test sets provided is powerful enough to prove the incompleteness of Π_4, although $S(\Pi_4) \subset S(\Pi_1)$, and $L(\Pi_4) \subset L(\Pi_1)$.

A more powerful testing set is computed by considering a generalization of the RC criterion, called *context-dependent rule coverage* (CDRC) criterion.

Definition 5. *A rule $r \in R$, $r : b \to uav$, $u, v \in V^*$, $a, b \in V$, is called a* direct occurrence *of a. For every symbol $a \in V$, we denote by $Occs(\Pi, a)$, the set of all direct occurrences of a.*

For the P system Π_1, the following sets of direct occurrences are computed:

- $Occs(\Pi_1, s) = \emptyset$;
- $Occs(\Pi_1, a) = \{r_1 : s \to ab\}$;
- $Occs(\Pi_1, b) = \{r_1 : s \to ab,\ r_3 : b \to bc\}$;
- $Occs(\Pi_1, c) = \{r_2 : a \to c,\ r_3 : b \to bc,\ r_4 : b \to c\}$.

Definition 6. *A multiset denoted by $u \in L(\Pi)$* covers *the rule $r : b \to y \in R$ for the* direct occurrence *of b, $a \to ubv \in R$ if there is a derivation $w \Longrightarrow^* u_1av_1 \Longrightarrow u_1ubvv_1 \Longrightarrow u_1uyvv_1 \Longrightarrow^* z$; $z, u_1, v_1, u, v, y \in V^*$, $a, b \in V$. A set T_r is said to cover $r : a \to x$ for all direct occurrences of a if for any occurrence $o \in Occs(\Pi, a)$ there is $t \in T_r$ such that t covers r for o. A set T is said to achieve CDRC for Π if it covers all $r \in R$ for all their direct occurrences.*

Clearly, T_r covers r in the sense of Definition 3. Similar to the coverage rule criterion introduced by Definition 4 where a terminal coverage criterion (RTC) has been given, we can also extend CDRC by considering only terminal derivations for all z in Definition 6 and obtain the CDRTC criterion. Obviously, any test set that satisfies CDRC (CDRTC) criterion will also satisfies RC (RTC) criterion, as well.

For Π_1 the set

- $T' = \{(0,1,1,0),(0,0,1,2),(0,0,0,2),(0,0,1,3),(0,0,0,3))\}$ satisfies the CDRC criterion and
- $T'' = \{(0,0,0,2),(0,0,0,3),(0,0,0,4)\}$ satisfies the CDRTC criterion.

These sets show the incompleteness of Π_4 as well as the incompleteness of the other two P systems.

In all the above considerations we have considered maximal parallelism. If we consider sequential P systems, only one rule is used at a moment, then all the above considerations and the same sets remain valid.

Now we consider general P systems, as introduced by Definition 1, and reformulate the concepts introduced above for one compartment P systems:

– RC criterion becomes: the configuration $(u_1, ..., u_i, ..., u_n)$ covers a rule $r_i : a_i \to v_i \in R_i$ if there is a derivation $(w_1, ..., w_i ..., w_n) \Longrightarrow^* (x_1, ..., x_i a_i y_i, ..., x_n) \Longrightarrow (x'_1, ..., x'_i v_i y'_i, ..., x'_n) \Longrightarrow^* (u_1, ..., u_i, ..., u_n)$;
– a test set $T \subseteq (V^*)^n$ is defined similar to Definition 4.

In a P system with more than one compartment, two adjacent regions can exchange symbols. If the region i is contained in j and $r_i : a \to x(b, out)y \in R_i$ or $r_j : c \to u(d, in)v \in R_j$ then r_i, r_j are called communication rules between regions i and j.

Now Definition 5 can be rewritten as follows.

Definition 7. *A rule $r : b \to uav \in R_i$ or a communication rule between i and j, $r' : b' \to u'(a, t)v' \in R_j$, where i and j are two adjacent regions and $t \in \{in, out\}$, is called a direct occurrence of a. The set of all direct occurrences of a in region i is denoted by $Occs_i(\Pi, a)$ and consists of the set of all direct occurrences of a from i, denoted by S_i and the sets of communication rules, r', from the adjacent regions $j_1, ..., j_q$, denoted by $S_{j_1}, ..., S_{j_q}$.*

Let the two compartment P systems:

$$\Pi'_i = (V_i, \mu_i, w_{1,i}, w_{2,i}, R_{1,i}, R_{2,i}), \text{ where:}$$

– $V_1 = V_2 = \{s, a, b, c\}$;
– $\mu_1 = \mu_2 = [_1[_2\ ]_2]_1$;
– $w_{1,1} = s,\ w_{2,1} = \lambda,\ w_{1,2} = s,\ w_{2,2} = \lambda$;
– $R_{1,1} = \{r_1 : s \to sa(b, in),\ r_2 : s \to ab,\ r_3 : b \to a,\ r_4 : a \to c\}$;
– $R_{2,1} = \{r_1 : b \to bc,\ r_2 : b \to c\}$;
– $R_{1,2} = \{r_1 : s \to sa(b, in),\ r_2 : s \to ab(b, in)(c, in),\ r_3 : b \to a,\ r_4 : a \to c\}$;
– $R_{2,2} = \{r_1 : b \to \lambda,\ r_2 : b \to c\}$.

For the P system Π'_1, the following sets of direct occurrences are computed:

– $Occs_1(\Pi_1, a) = S_1 \cup S_2$, where $S_1 = \{r_1 : s \to sa(b, in),\ r_2 : s \to ab,\ r_3 : b \to a\}$ and $S_2 = \emptyset$;
– $Occs_2(\Pi'_1, b) = S_1 \cup S_2$, where $S_1 = \{r_1 : s \to sa(b, in)\}$ and $S_2 = \{r_1 : b \to bc\}$.

A test set T that satisfies the CDRC criterion is:

$$\{((1,1,0,0),(0,0,1,0)),((0,1,1,1),(0,0,1,1)),((0,1,0,2),(0,0,0,2)),$$
$$((0,0,0,3),(0,0,0,2))\},$$

which is obtained from the derivation

$$(s, \lambda) \Longrightarrow (sa, b) \Longrightarrow (bac, bc) \Longrightarrow (acc, cc) \Longrightarrow (ccc, cc).$$

The P system Π'_2 is incomplete as it does not contain configurations (c^k, c^h) with $h > k$, but T above, fails to show this fact.

We can consider the CDRTC criterion to check whether it distinguishes between Π'_1 and Π'_2. It is left to the reader to verify this fact.

4 Finite State Machine Based Testing

In this section we apply concepts and techniques from finite state based testing. In order to do this, we construct a finite automaton on the basis of the derivation tree of a P system.

We first present the process of constructing a DFA for a one compartment P system $\Pi = (V, \mu, w, R)$, where $\mu = [_1\ ]_1$. In this case, the configuration of Π can change as a result of the application of some rule in R or of a number of rules, in parallel. In order to guarantee the finiteness of this process, for a given integer k, only computations of maximum k steps will be considered. For example, for $k = 4$, the tree in Figure 1 depicts all derivations in Π_1 of length less than or equal to k. The terminal nodes are in bold.

As only sequences 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 tree that depicts all the derivations of a P system Π with rules $R = \{r_1, \ldots, r_m\}$ can be described by a DFA Dt over the alphabet $A = \{r_1^{i_1} \ldots r_m^{i_m} \mid 0 \leq i_1 \leq N_1, \ldots, 0 \leq i_m \leq N_m\}$, where $r_1^{i_1} \ldots r_m^{i_m}$ describes the multiset with i_j occurrences of r_j, $1 \leq j \leq m$.

As Dt is a DFA over A, one can construct the minimal DFA that accepts *precisely* the language $L(Dt)$ defined by Dt. However, as only sequences of at most k transitions are considered, it is irrelevant how the constructed automaton will behave for longer sequences. Thus, a finite cover automaton can be constructed instead.

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.

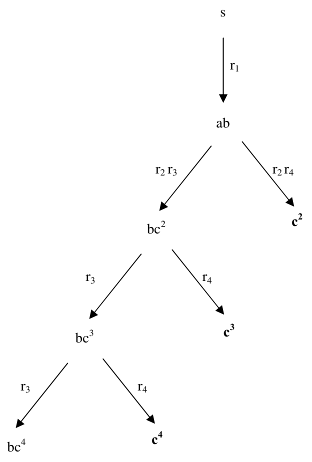

Fig. 1. Derivation tree for Π_1 and $k = 4$

Definition 8. *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 $L(A) \cap A[l] = U$, where $A[l] = \bigcup_{0 \leq i \leq l} U^i$ denotes the sets of sequences of length less than or equal to l with members in the alphabet A.*

A *minimal* DFCA of U is a DFCA of U having the least number of states. Unlike the case in which the acceptance of the precise language is required, the minimal DFCA is not necessarily unique (up to a renaming of the state space) [5], [6].

The concept of DFCA was introduced in [5], [6] and several algorithms for constructing a minimal DFCA of a finite language have been devised since [5], [6], [4], [3], [11], [12], [16]. The time complexity of these algorithms is polynomial in the number of states of the minimal DFCA. Interestingly, the minimization of DFCA can be approached as an inference problem ([8]), which had been solved several years earlier.

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, as shown by examples in this paper, a minimal DFCA of U may have considerably fewer states than the minimal DFA that accepts U.

A minimal DFCA of the language $L(Dt)$ defined by the previous derivation tree is represented in Figure 2; q_3 in Figure 2 is final state. It is implicitly assumed that a non-final "sink" state, denoted q_S, also exists, that receives all "rejected" transitions. For testing purposes we will consider all the states as final.

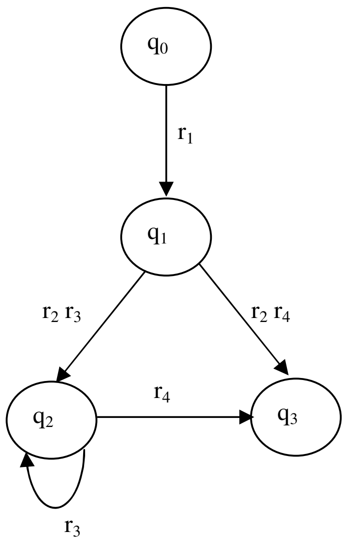

Fig. 2. Minimal DFCA for Π_1 and $k = 4$

Not only the minimal DFCA of $L(Dt)$ may have (significantly) less states than the minimal DFA that accepts $L(Dt)$, but it also provides the right approximation for the computation of a P system. Consider $u_1, u_2 \in V^*$, $w \Longrightarrow^* u_1$, $w \Longrightarrow^* u_2$, such that the derivation from u_1 is identical to the derivation from u_2, i.e., any sequence $s \in A^*$ that can be applied to u_1 can also be applied to u_2 and vice versa (e.g., $u_1 = bc^2$ and $u_2 = bc^3$ in Figure 1). Naturally, as the derivation from u_1 is identical to the derivation from u_2, 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 $L(Dt)$; on the other hand, u_1 and u_2 will be associated with distinct states in the minimal DFA that accepts $L(Dt)$, unless they appear at the same level in the derivation tree Dt. For example, in Figure 1, bc^2 and bc^3 appear at different levels in the derivation tree and so they will be associated with distinct states in the minimal DFA that accepts $L(Dt)$; on the other hand, bc^2 and bc^3 are mapped onto the same state (q_2) of the minimal DFCA represented in Figure 2. Furthermore, if the entire computation of the P system (i.e. for derivation sequences of *any* length) can be described by a DFA over some alphabet A, then this DFA model will be obtained as a DFCA of $L(Dt)$ for k sufficiently large.

Once the minimal DFCA $M = (A, Q, q_0, F, h)$ has been constructed, various specific coverage levels can be used to measure the effectiveness of a test set. In this paper we use two of the most widely known coverage levels for finite automata: *state coverage* and *transition coverage.*

Definition 9. *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 exists $u \in T$ 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 10. *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 ($h(q, a) \neq q_S$), 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.*

Clearly, if a test set satisfies TC, it also satisfies SC. A test set for Π_1 satisfying the SC criterion is

$T_{1,1} = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2)\}$,

whereas a test set satisfying the TC criterion is

$T_{1,s} = \{(1,0,0,0), (0,1,1,0), (0,0,1,2), (0,0,0,2), (0,0,1,3), (0,0,0,3)\}$.

The TC coverage criterion defined above is, in principle, analogous to the RC criterion given in the previous section. The TC criterion, however, does not only depend on the rules applied, but also on the state reached by the system when a given rule has been applied. Test suites that meet the RC and TC criteria can be efficiently calculated using automata inference techniques [9], [15]. A stronger criterion, in which all feasible transition sequences of length less than or equal to

2 must be triggered in any state can also be defined – this will correspond to the CDRC criterion defined in the previous section. Of course, a more careful analysis of the relationships between criteria used in testing based on grammars and those applied in the context of finite state machines, considered for P systems testing, needs to be conducted in order to identify the most suitable testing procedures for these systems.

The construction of a minimal DFCA and the coverage criteria defined above can be generalized for a multiple compartment P system

$$\Pi = (V, \mu, w_1, ..., w_n, R_1, ..., R_n).$$

In this case, the input alphabet will be defined as

$$A = \{(r_1^{i_{1,1}} \dots r_{m_1}^{i_{1,m_1}}, \dots, r_1^{i_{n,1}} \dots r_{m_n}^{i_{n,m_n}}) \mid \\ 0 \leq i_{j,p} \leq N_{j,p}, 1 \leq j \leq m_p, 1 \leq p \leq n\},$$

where $N_{j,p}$ is the maximum number of times rule r_j, $1 \leq j \leq m_p$ from compartment p can be applied in one derivation step, $1 \leq p \leq n$. Analogously to one compartment P systems, only computations of maximum k steps are considered.

For $k = 3$, the derivation tree of Π_1' defined above is as represented in Figure 3. For clarity, in Figure 3 if the derivation from some node u (not found at the bottom level in the hierarchy) is the same as the derivation from some previous node u' at a higher level or at the same level in the hierarchy, then u is not expanded any further; we denote $u \sim u'$. Such nodes are (sac, bc) and (abc, c), $(sac, bc) \sim (sa, b)$ and $(abc, c) \sim (ab, \lambda)$; they are given in italics. A minimal DFCA of the language defined by the derivation tree is represented in Figure 4.

Similar to one compartment case, test sets for considered criteria, state and transition cover, can be defined in this more general context.

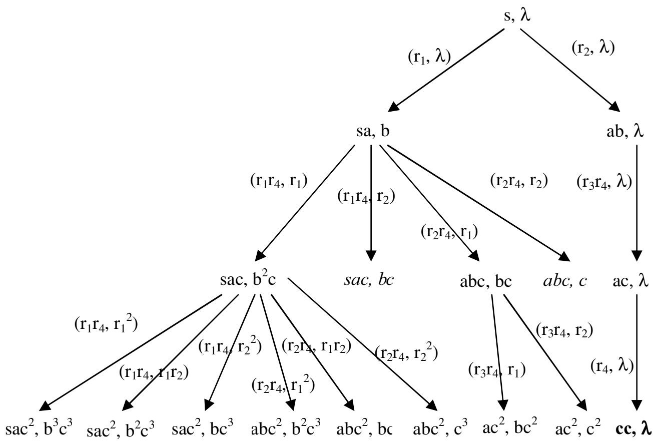

Fig. 3. Derivation tree for Π_1' and $k = 3$

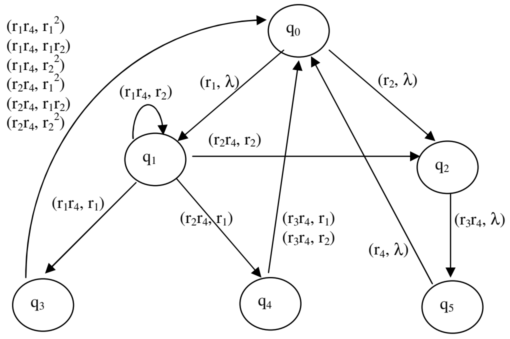

Fig. 4. Minimal DFCA for Π_1' and $k = 3$

5 Conclusions and Future Work

In this paper we have discussed how P systems are tested by introducing grammar and finite state machine based strategies. The approach is focussing on cell-like P systems, but the same methodology can be used for tissue-like P systems. Simple examples illustrate the approach and show their testing power as well as current limitations. This very initial research reveal a number of interesting preliminary problems regarding the construction of relevant test sets that point out faulty implementations.

This paper has focused on coverage criteria for P system testing. In grammar based testing, coverage is the most widely used test generation criteria. For finite state based testing we have considered some simple state and transition coverage criteria, but criteria for conformance testing (based on equivalence proofs) can also be used; this approach is the subject of a paper in progress. Future research is also intended to cover relationships between components and the whole systems with respect to testing, other testing principles based on the same criteria and strategies, as well as new strategies and different testing methods. Relationships between testing criteria based on grammars and those used in the case of finite state machine based specifications remain to be further investigated in a more general context.

Acknowledgements. The authors of the paper are grateful to the anonymous referees for their comments and suggestions.

References

1. Andrei, O., Ciobanu, G., Lucanu, D.: Structural operational semantics of P systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 31–48. Springer, Heidelberg (2006)
2. Andrei, O., Ciobanu, G., Lucanu, D.: A rewriting logic framework for operational semantics of membrane systems. Theoretical Computer Sci. 373, 163–181 (2007)

3. Câmpeanu, C., Păun, A., Smith, J.R.: Incremental construction of minimal deterministic finite cover automata. Theoretical Computer Sci. 363, 135–148 (2006)
4. Câmpeanu, C., Păun, A., Yu, S.: An efficient algorithm for constructing minimal cover automata for finite languages. Intern. J. Foundation of Computer Sci. 13, 83–97 (2002)
5. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. In: Champarnaud, J.-M., Maurel, D., Ziadi, D. (eds.) WIA 1998. LNCS, vol. 1660, pp. 43–56. Springer, Heidelberg (1999)
6. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. Theoretical Computer Sci. 267, 3–16 (2002)
7. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer, Heidelberg (2006)
8. García, P., Ruiz, J.: A note on the minimal cover-automata for finite languages. Bulletin of the EATCS 83, 193–194 (2004)
9. Gold, M.E.: Complexity of automaton identification from given data. Information and Control 37, 302–320 (1978)
10. Holcombe, M., Ipate, F.: Correct Systems – Building Business Process Solutions. Springer, Heidelberg (1998)
11. Körner, H.: On minimizing cover automata for finite languages in O(n log n) time. In: Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2002. LNCS, vol. 2608, pp. 117–127. Springer, Heidelberg (2003)
12. Körner, H.: A time and space efficient algorithm for minimizing cover automata for finite languages. Intern. J. Foundation of Computer Sci. 14, 1071–1086 (2003)
13. Lämmel, R.: Grammar testing. In: Hussmann, H. (ed.) FASE 2001. LNCS, vol. 2029, pp. 201–216. Springer, Heidelberg (2001)
14. Li, H., Jin, M., Liu, C., Gao, Z.: Test criteria for context-free grammars. COMPSAC 1, 300–305 (2004)
15. Oncina, J., García, P.: Inferring regular languages in polynomial update time. In: de la Blanca, N.P., Sanfeliu, A., Vidal, E. (eds.) Pattern Recognition and Image Qnalysis, pp. 49–61. World Scientific, Singapore (1992)
16. Păun, A., Sântean, N., Yu, S.: An O(n^2) algorithm for constructing minimal cover automata for finite languages. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 243–251. Springer, Heidelberg (2001)
17. Păun, G.: Computing with membranes. J. Computer and System Sci. 61, 108–143 (2000)
18. Păun, G.: Membrane Computing. An Introduction. Springer, Heidelberg (2002)
19. The P Systems Web Site, `http://ppage.psystems.eu`