Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 227 (2009) 113–126
www.elsevier.com/locate/entcs

Testing Non-deterministic Stream X-machine Models and P systems

Florentin Ipate [1]

Department of Computer Science
University of Pitesti
Str Targu din Vale 1, 110040 Pitesti, Romania

Marian Gheorghe[2]

Department of Computer Science
University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK

Abstract

Under certain well defined conditions, the stream X-machine testing method can produce a test set that is guaranteed to determine the *correctness* of an implementation. The testing method has originally assumed that an implementation of each processing function or relation is proven to be correct before the actual testing can take place. Such a limitation has been removed in a subsequent paper, but only for deterministic X-machines. This paper extends this result to non-deterministic stream X-machines and considers a conformance relationship between a specification and an implementation, rather than mere equivalence. Furthermore, it shows how this method can be applied to test a P system by building a suitable stream X-machine from the derivation tree associated with a partial computation.

Keywords: testing, correctness, X-machines, P systems, finite state machines, non-determinism.

1 Introduction

The *stream X-machine* (*SXM*) [10] is a form of extended finite state machine (FSM). It describes a system as a finite set of states and a number of transitions between states. In addition, a SXM contains an internal store, called *memory*. A transition is triggered by an input value, produces an output value and may access and/or alter the memory. A transition diagram of a SXM is a finite automaton (called an *associated automaton*) in which the arcs are labelled by relation names (referred to as *processing relations*). Under certain well defined *design for test conditions*, it is

[1] Email: florentin.ipate@ifsoft.ro
[2] Email: m.gheorghe@dcs.shef.ac.uk

1571-0661/
doi:10.1016/j.entcs.2008.12.107

possible to produce a test set that is guaranteed to determine the correctness of an implementation under test (IUT) [14,10,2,12,16].

Traditional extended finite-state machine test generation methods [3,17] rely on the construction of an equivalent finite state machine, where states are the state/memory pairs of the original SXM. For complex specifications, this leads to a known state explosion problem; the SXM testing method does not perform such a construction. Instead, however, the method assumes that the processing functions (relations) are correctly implemented and reduces testing of a SXM to testing of its associated automaton. Therefore, it is fair to say that the method only tests the *integration* of the implementation of processing functions (relations). In practice, the correctness of an implementation of a processing function (relation) is checked by a separate process [10], using the SXM testing method or alternative functional methods. The method (called in what follows *SXM integration testing*) was first developed in the context of *deterministic* SXMs (i.e. those SXMs in which all labels represent partial functions rather than relations and an input can trigger at most one transition in any state and for any memory value). The deterministic SXM ($DSXM$) integration testing method [14,10] was extended to the non-deterministic case ($NSXM$ integration testing) in paper [15]. Conformance testing for SXMs has been previously considered in [8] for a subclass of quasi-non-deterministic SXMs; later work [9] uses a rather general definition of conformance, but requires an implementation to compute a function (this paper assumes a non-deterministic IUT).

The applicability of the mentioned integration testing methods is limited by the assumption that the implementation of each processing function (relation) can be tested *in isolation* from the rest of the system. This is not always a realistic assumption. This limitation has been removed [11] in the context of deterministic SXMs, so that testing of functions can be performed along with the integration testing. This paper extends [11] for non-deterministic SXMs. This is called *complete NSXM testing*. Such an extension is not trivial for the following reasons: (a) in a non-deterministic SXM an input sequence may potentially produce an infinite number of output sequences, while a test set will be applied to an IUT for a limited number of times (as few as possible); (b) in a non-deterministic SXM some paths may never be exercised.

Since their introduction in 1998 [19], P systems have been intensively studied and developed, in particular with regard to the computational power of different variants and their capability to solve hard problems. In the last years there have also been significant developments in using the P systems paradigm to model, simulate and formally verify various systems [5]. Suitable classes of P systems have been associated with some of these applications and software packages have been developed. Although formal verification has been applied to different models based on P systems [10], testing is completely neglected in this context. Testing is an essential part of software development and all software applications, irrespective of their use and purpose, are tested before being released. Two recent papers provide initial steps towards building a P system testing theory: based on rule coverage [7] and on FSM conformance techniques [13]. In this paper we develop a testing method

for non-deterministic SXMs and show how this can be applied to P systems: a way of deriving a SXM from a P system is provided and the method is applied to the obtained model. This approach significantly extends the previous testing methods for P systems by considering (1) a SXM model of a P system instead of a simple FSM and also (2) non-determinism, a widely spread characteristic of P systems.

2 Preliminaries

This section introduces the notation and the formalisms used in the paper: finite state machines, stream X-machines and P systems.

For a finite alphabet A, A^* denotes the set of all finite sequences with members in A; ϵ denotes an empty sequence. For $a, b \in A^*$, ab denotes the concatenation of sequences a and b; a^n is defined by $a^0 = \epsilon$ and $a^n = a^{n-1}a$ for $n \geq 1$. For $U, V \subseteq A^*$, $UV = \{ab \mid a \in U, b \in V\}$; U^n is defined by $U^0 = \{\epsilon\}$ and $U^n = U^{n-1}U$ for $n \geq 1$. For a relation $f : A \longleftrightarrow B$, $dom(f)$ denotes the domain of f and $Im(f)$ denotes the image of f. If $a \notin dom(f)$, we write $f(a) = \emptyset$. For $U \subseteq A$, $f(U) = \cup_{a \in U} f(a)$ and $f \mid U : U \longleftrightarrow B$ denotes the restriction of f to U, i.e. $f \mid U(a) = f(a), \forall a \in U$. For two relations $f, g : A \longleftrightarrow B$, we use $f \sqsubseteq g$ to denote that $dom(f) = dom(g)$ and for any $a \in dom(f)$, $f(a) \subseteq g(a)$. For $\phi : M \times \Sigma \longleftrightarrow \Gamma \times M$ and $m \in M$ we define $\omega_m^\phi : \Sigma \longleftrightarrow \Gamma$ by $\omega_m^\phi(\sigma) = \pi_\Gamma(\phi(m, \sigma))$, $\sigma \in \Sigma$, where $\pi_\Gamma : \Gamma \times M \longrightarrow \Gamma$ denotes the projection function. We shall also use the projection function $\pi_M : \Gamma \times M \longrightarrow M$. For a finite set A, $\#A$ denotes the number of elements of A.

Definition 2.1 *A* finite automaton *(*FA *for short) is a tuple* (Σ, Q, F, I)*, where* Σ *is a finite* input alphabet*;* Q *is a finite* set of states*;* F *is a (partial)* next state function*,* $F : Q \times \Sigma \longrightarrow 2^Q$*;* I *is a set of* initial states*,* $I \subseteq Q$*; all states are assumed terminal.*

Function F is usually described by a state transition diagram. A FA is called *deterministic* if there is one initial state ($I = \{q_0\}$) and F maps each state/input pair into at most one state ($F : Q \times \Sigma \longrightarrow Q$). The next state function can be extended to a partial function $F^* : Q \times \Sigma^* \longrightarrow Q$ defined by $F^*(q, \epsilon) = q$, $\forall q \in Q$; $F^*(q, s\sigma) = F(F^*(q, s), \sigma))$, $\forall q \in Q, s \in \Sigma^*, \sigma \in \Sigma$. For $q \in Q$, $L_A(q) = \{s \in \Sigma^* \mid (q, s) \in dom(F^*)\}$. If $q = q_0$ then the set is simply denoted L_A and called the *language accepted by* A. A state $q \in Q$ is called *accessible* if $\exists s \in \Sigma^*$ such that $F^*(q_0, s) = q$. A is called *accessible* if $\forall q \in Q$, q is accessible. For $U \subseteq \Sigma^*$, two states $q_1, q_2 \in Q$ are called U*-equivalent* if $L_A(q_1) \cap U = L_A(q_2) \cap U$; otherwise, q_1 and q_2 are called U*-distinguishable*. If $U = \Sigma^*$ then q_1 and q_2 are simply called *equivalent* or *distinguishable*. A is called *reduced* if $\forall q_1, q_2 \in Q$, $((q_1 \neq q_2) \Longrightarrow (q_1$ and q_2 are distinguishable)). A deterministic FA A is called *minimal* if any other FA that accepts the same language as A has at least the same number of states as A. The reader is assumed to be familiar with basic automata theory, for details see for example [6].

Given a FA specification A and a class of implementations C, a *test set of* A *w.r.t.* C is a set of input sequences that, when applied to any implementation A' in the class C, will detect any response in A' that does not conform to the

response specified by A, i.e. $\forall A' \in C$, $(L_A \cap Y = L_{A'} \cap Y \Longrightarrow L_A = L_{A'})$. The class C is identified by the assumptions we can make about an implementation A'. If no information is available, a test set may not exist for even very simple FA specifications. There are a number of more or less realistic assumptions that one can make about the form and size of an implementation and these, in turn, give rise to different techniques for generating test sets [17]. One of the least restrictive assumptions refers to the number of states of A' and is the basis for the W method [4,1]: the difference between the number of states of an implementation and that of a specification has to be at most k, a non-negative integer estimated by a tester.

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

Definition 2.2 *$S \subseteq \Sigma^*$ is called a* state cover *of A if $\epsilon \in S$ and $\forall q \in Q \setminus \{q_0\}$, $\exists s \in S$ such that $F^*(q_0, s) = q$. $P \subseteq \Sigma^*$ is called a* transition cover *of A if $S \cup S\Phi \subseteq P$ for some state cover S of A. $W \subseteq \Sigma^*$ is called a* characterisation set *of A if any two distinct states $q_1, q_2 \in Q$, $q_1 \neq q_2$, are W-distinguishable.*

Note that a state cover, a transition cover and a characterisation set exist if A is minimal.

Theorem 2.3 *[1] Let A be a deterministic FA having input alphabet Σ, n the number of states of A, $m \geq n$ and C_m the set of deterministic FAs with an input alphabet Σ and each with a maximal number of states m. If P is a transition cover and W a characterisation set of A then $Y_{m-n} = P(\Sigma^{m-n} \cup \Sigma^{m-n-1} \cup \ldots \cup \{\epsilon\})(W \cup \{\epsilon\})$ is a test set of A w.r.t. C_m.*

The above theorem is the theoretical basis for the W-method in the context of partially specified deterministic FA. The reason ϵ is included in W is to ensure that if an IUT has ignored the last element σ of a sequence of inputs verifying the existence of a transition triggered by σ, this will be detected.

A SXM is a form of extended FSM as defined next.

Definition 2.4 *A SXM is a tuple $Z = (\Sigma, \Gamma, Q, M, \Phi, F, I, m_0)$, where Σ and Γ are finite sets called an* input alphabet *an* output alphabet *respectively; Q is a finite set of* states*; M is a (possibly) infinite set called* memory*; Φ, called the* type *of Z, is a finite set of distinct* processing relations *that the machine can use. A processing relation is a non-empty relation of the form $\phi : M \times \Sigma \longleftrightarrow \Gamma \times M$. Often Φ is a set of (partial) functions. F is the (partial)* next state function*, $F : Q \times \Phi \longrightarrow 2^Q$. In a similar way to finite automata, F is usually described by a state transition diagram. I is a set of* initial states*, $I \subseteq Q$; m_0 is the* initial memory value*, $m_0 \in M$.*

It is sometimes helpful to think of a SXM as a finite automaton with the arcs labelled by relations from the type Φ. The FA $A_Z = (\Phi, Q, F, I, T)$ over an alphabet Φ is called the *associated FA* of Z.

Definition 2.5 *Given a sequence $p \in \Phi^*$, p induces the relation $|p| : M \times \Sigma^* \longleftrightarrow \Gamma^* \times M$ defined as follows: (1) $(m, \epsilon) \mid \epsilon \mid (\epsilon, m)$, $\forall m \in M$, (2) $\forall p \in \Phi^*, \phi \in \Phi$, $(m, s\sigma) \mid p\phi \mid (g\gamma, m')$, where $\forall m, m' \in M, s \in \Sigma^*, g \in \Gamma^*, \sigma \in \Sigma, \gamma \in \Gamma$ are such that $\exists m'' \in M$ with $(m, s) \mid p \mid (g, m'')$ and $(m'', \sigma)\phi(\gamma, m')$.*

A sequence $\mid p \mid$ can be considered a relation between a (memory, input string) pair and the (output string, memory) pairs produced by a consecutive application of the relations in p. It is easy to see that if Φ is a set of (partial) functions rather than relations then $\mid p \mid$ is a also a (partial) function.

Definition 2.6 *A SXM Z is called* deterministic *if the following three conditions hold:* (1) *the associated FA of the machine Z is deterministic, i.e. Z has only one initial state ($I = \{q_0\}$) and the next state function maps each pair of (state, processing function) onto at most one state ($F : Q \times \Phi \longrightarrow Q$);* (2) *$\Phi$ is a set of (partial) functions rather than relations;* (3) *any two distinct processing functions that label arcs emerging from the same state have disjoint domains, i.e. $\forall \phi_1, \phi_2 \in \Phi$, $((\exists q \in Q$ with $(q, \phi_1), (q, \phi_2) \in dom(F)) \Longrightarrow (\phi_1 = \phi_2$ or $dom(\phi_1) \cap dom(\phi_2) = \emptyset))$.*

From the above definition, NSXMs can have three types of non-determinism: *state non-determinism* if $\#I > 1$ or $\exists q \in Q$, $\phi \in \Phi$ with $\#(F(q, \phi)) > 1$; *operator non-determinism* if some elements of Φ are relations but not partial functions; *domain non-determinism* if there exist $q \in Q, \phi_1, \phi_2 \in \Phi$, $\phi_1 \neq \phi_2$ with $(q, \phi_1), (q, \phi_2) \in dom(F)$ and $dom(\phi_1) \cap dom(\phi_2) \neq \emptyset$. It is not necessary to consider the case of state non-determinism since it can easily be eliminated by rewriting a NSXM using standard algorithms that take a non-deterministic FA and produce an equivalent deterministic FA; in general, this transformation may introduce domain non-determinism. Further, NSXMs are assumed to be free of state non-determinism and are denoted by a tuple $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where F is a partial function and q_0 is the initial state. The associated deterministic FA is then a tuple $A_Z = (\Phi, Q, F, q_0)$. In general, a non-deterministic SXM computes a relation since the application of an input sequence may produce more than one output sequence. The exact correspondence between an input sequence and an output produced is defined next.

Definition 2.7 *A SXM Z,* computes a relation $f_Z : \Sigma^* \longleftrightarrow \Gamma^*$ *defined by: $s\ f_Z\ g$ if there is $p \in \Phi^*, m \in M$ such that $p \in L_{A_Z}$ and $(m_0, s) \mid p \mid (g, m)$.*

When a path p is important, we will write $s\ f_Z^p\ g$. Note that for a SXM featuring domain non-determinism, p may not be uniquely identified by s. If Z is a deterministic SXM, then f_Z is a (partial) function rather than a relation. We define a computation of Z as any subset of f_Z that associates each input sequence s with output sequences produced when a machine exercises every path in Z that can be traversed by s at least once.

Definition 2.8 *A relation $\hbar : \Sigma^* \longleftrightarrow \Gamma^*$ is called a* computation *of Z if the following two conditions hold:* (1) $\forall\ s \in \Sigma$, $g \in \Gamma$, $s\ \hbar\ g \Longrightarrow s\ f_Z\ g$; (2) $\forall\ s \in \Sigma$, $g \in \Gamma$, $p \in \Phi^*$, $s\ f_Z^p\ g \Longrightarrow (s\ f_Z^p\ g'$ *and* $s\ \hbar\ g'$ *for some* $g' \in \Gamma)$.

The set of all computations of Z is denoted by $\mathcal{H}_Z$. It is easy to see that $f_Z \in \mathcal{H}_Z$ and $\forall\ \hbar \in \mathcal{H}_Z$, $\hbar \sqsubseteq f_Z$. If Φ is a set of (partial) functions (such as when Z is a DSXM) then $\mathcal{H}_Z = \{f_Z\}$, because in this case for each path through the machine there is only one sequence of possible outputs.

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 [20]. Since now onwards we will refer to this model as simply P system.

Definition 2.9 *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 (or environment when the current region is the most external) 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 $c_1 \Longrightarrow c_2$.

3 NSXM Integration Testing

This section presents the theoretical basis for the NSXM integration testing method [15]. This method generates a test set from a non-deterministic SXM specification, providing that the system components (i.e. processing relations) are implemented correctly. Therefore, it is assumed that the IUT is a NSXM having the same type (processing relations) as a specification. The concepts and results in this section are largely from [15], the presentation of which has been slightly modified to fit with other published work in the area [8,11]. The essence remains unchanged.

In order to test *non-deterministic* implementations, one usually makes a so-called *complete-testing assumption* [18]: it is possible, by applying a given input sequence s to an implementation for a finite number of times, to exercise all the paths of the implementation that can be traversed by s. Without such an assumption, no test suite can guarantee full fault coverage of non-deterministic implementations. For testing of an IUT Z' against a specification Z, we apply elements of a test set X a number of times, so as to ensure that both a specification and an implementation traverse all paths they can. During a test, only a subset of a (potentially infinite) set of outputs is observed from a non-deterministic implementation; for this reason, a possible *computation* $\hbar'$ of Z' is observed $(\hbar' \mid X)$. If all outputs of Z' in response to X can be produced by Z, it means that Z performed a computation $\hbar$ and

$\hbar \mid X = \hbar' \mid X$. In this case, we should be able to conclude that for any sequence of inputs Z' will produce an output which is allowed by Z. This justifies the following.

Definition 3.1 *Let Z be a SXM and C a set of SXMs having the same input alphabet (Σ) and output alphabet (Γ) as Z. Then a finite set $X \subseteq \Sigma^*$ is called a* test set *of Z w.r.t. C if $\forall Z' \in C$, $\hbar \mid X = \hbar' \mid X$ for some $\hbar \in \mathcal{H}_Z$, $\hbar' \in \mathcal{H}_{Z'}$ implies $f_Z = f_{Z'}$.*

It is assumed that an IUT is a NSXM having the same input alphabet, output alphabet, memory and an initial memory value as a specification; additionally, a SXM specification has to satisfy three conditions: input-completeness, output-distinguishability and observability.

Definition 3.2 *Two SXMs Z and Z' are called* weak testing compatible *if they have identical input alphabets, output alphabets, memory sets and initial memory values. Two weak testing compatible SXMs are called* testing compatible *if they have identical types.*
Φ is called input-complete *if $\forall \phi \in \Phi, m \in M$, $\exists \sigma \in \Sigma$ such that $(m, \sigma) \in dom(\phi)$.*
Φ is called output-distinguishable *if $\forall \phi_1, \phi_2 \in \Phi$, (there are $m \in M, \sigma \in \Sigma$ such that $\omega_m^{\phi_1}(\sigma) \cap \omega_m^{\phi_2}(\sigma) \neq \emptyset$), then $\phi_1 = \phi_2$ (ω_m^{ϕ} was introduced in Sect. 1).*
Φ is called observable *if $\forall \phi \in \Phi, \sigma \in \Sigma, \gamma \in \Gamma, m, m_1, m_2 \in M$, $((\gamma, m_1) \in \phi(m, \sigma)$, $(\gamma, m_2) \in \phi(m, \sigma)$ implies $m_1 = m_2)$.*

These three conditions (input-completeness, output-distinguishability and observability) are generally known as "design for test conditions" [10,14]. [3] Without them, it would be extremely difficult to test a system properly. The input-completeness condition ensures that all sequences of processing relations in the associated FA can be attempted using appropriate inputs, so they can be tested against the implementation. The output-distinguishability condition ensures that any processing relation can be identified from the machine computation by examining the outputs produced. When Φ is observable, the next memory value computed by relations can be determined. Note that if Φ is a set of (partial) functions then it is already observable, so this condition is not explicitly stated when considering such SXMs (such as DSXMs).

The basic idea of the testing method is to translate a test set of an associated FA into a test set of an IUT. In order to do this, we need a mechanism, called a *test function*, that converts sequences of processing relations into sequences of inputs.

Definition 3.3 *Let $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a SXM with an input-complete type Φ. A* test function of Z *$t : \Phi^* \longrightarrow \Sigma^*$ is defined as follows. First of all, $t(\epsilon) = \epsilon$; for $n > 0$ and $\phi_1, \ldots, \phi_n \in \Phi$, $t(\phi_1 \ldots \phi_n) = \sigma_1 \ldots \sigma_k$, where $\sigma_1, \ldots, \sigma_k \in \Sigma$ are such that $(m_0, \sigma_1 \ldots \sigma_k) \in dom(\mid \phi_1 \ldots \phi_k \mid)$. In order to determine the value $k \leq n$, two cases are considered, (1) $\phi_1 \ldots \phi_n \in L_{A_Z}$, in which case $k = n$; (2) $\phi_1 \ldots \phi_n \notin L_{A_Z}$ and for $0 \leq i < n$ there is $\phi_1 \ldots \phi_i \in L_{A_Z}$ such that $\phi_1 \ldots \phi_{i+1} \notin L_{A_Z}$, in which case $k = i + 1$.*

[3] The design for test conditions for a deterministic SXM have been relaxed in [12] and [16]

In other words, for any sequence $v = \phi_1 \dots \phi_n$ of processing relations, $t(v)$ is a sequence of inputs that exercises the longest prefix $\phi_1 \dots \phi_i$ of v that is a path in a specification NSXM and, if $i < n$, also exercises ϕ_{i+1}, the relation that follows after this prefix. Note that since Φ is input-complete there always exist $\sigma_1, \dots, \sigma_k$ as above, but these are not necessarily uniquely defined.

The result below is the theoretical basis for NSXM integration testing.

Theorem 3.4 *[15] Let Z be a SXM having type Φ input-complete, output-distinguishable and observable and C a set of SXMs testing compatible with Z. If t is a test function of Z and $Y \subseteq \Phi^*$ a test set of A_Z w.r.t. A_C, where $A_C = \{A_{Z'} \mid Z' \in C\}$, then $X = t(Y)$ is a test set of Z w.r.t. C.*

Since C is a set of (non-deterministic) SXMs testing compatible with Z, it is assumed that the processing relations are implemented correctly, i.e. an IUT uses the same set of processing relations as a specification. Therefore, the method only tests the *integration* of processing relations. The correctness of implementation of these relations is checked by separate testing processes, as discussed in [15].

We can now use theorems 3.4 and 2.3 to generate a test set of Z w.r.t. C_m, the set of NSXMs testing compatible with Z whose number of states does not exceed $m \geq n$. This is $X_{m-n} = t(Y_{m-n})$, where $Y_{m-n} = P(\Phi^{m-n} \cup \dots \cup \{\epsilon\})(W \cup \{\epsilon\})$, n is the number of states in Z, P is a transition cover and W a characterisation set of A_Z and t is a test function of Z. More details about the applicability of the NSXM integration testing method can be found in reference [15].

4 Theoretical Basis for Complete NSXM Testing

This section presents the theoretical basis for the complete NSXM testing method. Unlike integration testing, no assumption is made here regarding the correctness of the implementation of processing relations. Therefore, the general case is considered when a specification and an IUT may have different types (i.e. are *weak* testing compatible). Furthermore, an IUT will be checked for conformance to a specification rather than for equivalence. An IUT Z' conforms to Z if Z' is defined on all inputs on which Z is defined and the behaviour of Z' is a subset of the behaviour of Z that traverses every path of Z at least once.

Definition 4.1 *Let Z and Z' be two SXMs having the same input alphabet (Σ) and output alphabet (Γ). We say that Z'* conforms *to Z, written $Z' \sqsubseteq Z$, if $f_{Z'} \sqsubseteq f_Z$ and $\mathcal{H}_Z \cap \mathcal{H}_{Z'} \neq \emptyset$.*

The definition of a test set is revised to reflect the more general situation.

Definition 4.2 *Let Z be a SXM and C a set of SXMs with the same input alphabet (Σ) and output alphabet (Γ) as Z. Then a finite set $X \subseteq \Sigma^*$ is called a* conformance test set *of Z w.r.t. C if $\forall Z' \in C$, $\hbar \mid X = \hbar' \mid X$ for some $\hbar \in \mathcal{H}_Z$, $\hbar' \in \mathcal{H}_{Z'}$ implies $Z' \sqsubseteq Z$.*

Obviously, any test set of Z w.r.t. C is also a conformance test set of Z w.r.t. C. If Z is a *deterministic* SXM then the notions of a test set and a conformance test

set coincide.

The output-distinguishability condition has to be updated for the situation where a specification and an implementation may have different types.

Definition 4.3 *Let* $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ *and* $Z' = (\Sigma, \Gamma, Q', M, \Phi', F', q'_0, m_0)$ *be two weak testing compatible SXMs having types* Φ *and* Φ'*, respectively. Then* Φ *is called* output-distinguishable w.r.t. Φ' *if there exists a bijective function* $c : \Phi \longrightarrow \Phi'$ *such that the following holds:* $\forall \phi \in \Phi, \phi' \in \Phi'$, $((\exists m \in M, \sigma \in \Sigma$ *such that* $\omega^{\phi}_m(\sigma) \cap \omega^{\phi'}_m(\sigma) \neq \emptyset) \Longrightarrow \phi' = c(\phi))$.

This says that, for a processing relation ϕ in Φ, we must be able to identify a corresponding relation $\phi' = c(\phi)$ in Φ' by examining outputs. Naturally, if Φ is output-distinguishable then Φ is also output-distinguishable w.r.t. itself. In the conditions of Def. 4.3 we denote by $A^{-c}_{Z'} = (\Phi, Q', F'_{-c}, q'_0)$ the FA obtained by substituting each arc $c(\phi)$ in $A_{Z'}$ with ϕ, i.e. $F'_{-c}(q', \phi) = F'(q', c(\phi))$. Obviously, for $\phi_1, \ldots, \phi_n \in \Phi$, $\phi_1 \ldots \phi_n \in L_{A^{-c}_{Z'}}$ iff $c(\phi_1) \ldots c(\phi_n) \in L_{A_{Z'}}$.

Design for test conditions imply two important properties of computations, (1) the $\mathcal{H}_Z \cap \mathcal{H}_{Z'} \neq \emptyset$ condition of Def. 4.1 requires A_Z and $A_{Z'}$ to be equivalent (reference [9] permits $L_{A'_Z} \subseteq L_{A_Z}$ rather than $L_{A'_Z} = L_{A_Z}$ as considered here, but restricts consideration to integration testing of an IUT which computes a function); (2) an equivalent definition to Def. 4.1 can be written as $dom f_{Z'} = dom f_Z \wedge f_{Z'} \in \mathcal{H}_Z$.

Since the method does not assume that processing relations are correctly implemented, we will have to test their implementations in addition to their integration. Therefore, a test set has to contain two components: an *integration* test set (from the previous section) and a set for testing processing relations (called a *relation test set*). The latter is defined below.

Definition 4.4 *Let* $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ *and* $Z' = (\Sigma, \Gamma, Q', M, \Phi', F', q'_0, m_0)$ *be two weak testing compatible SXMs. Then for* $\phi \in \Phi$ *and* $m \in M$*, a finite set* $\Sigma^{\phi}_m \subseteq \Sigma$ *is called an* m conformance test set of ϕ w.r.t. Φ' *if the following holds:* $\forall \phi' \in \Phi'$*, (if* $\psi \mid \Sigma^{\phi}_m = \psi' \mid \Sigma^{\phi}_m$ *for some* $\psi, \psi' : \Sigma \longleftrightarrow \Gamma$*,* $\psi \sqsubseteq \omega^{\phi}_m$*,* $\psi' \sqsubseteq \omega^{\phi'}_m$ *then* $\phi' \sqsubseteq \phi$*).*
Σ^{ϕ}_m *is called a* m test set of ϕ w.r.t. Φ' *if the following holds:* $\forall \phi' \in \Phi$*, (if* $\psi \mid \Sigma^{\phi}_m = \psi' \mid \Sigma^{\phi}_m$ *for some* $\psi, \psi' : \Sigma \longleftrightarrow \Gamma$*,* $\psi \sqsubseteq \omega^{\phi}_m$*,* $\psi' \sqsubseteq \omega^{\phi'}_m$ *then* $\phi' = \phi$*).*

An m (conformance) test set of ϕ w.r.t. Φ' is a finite set of inputs that checks ϕ (for conformance or equivalence) against any processing relation in Φ'.

Definition 4.5 *Let* Z *be a SXM having type* Φ*. Then a set* $V = \{v_1, \ldots, v_k\} \subseteq \Phi^*$ *is called a* relation cover *of* Z *if* Φ *can be written as* $\Phi = \{\phi_1, \ldots, \phi_k\}$ *such that the following hold: (1)* $v_1 = \epsilon$ *and* $\phi_1 \in L_{A_Z}$*; (2) for any* $2 \leq i \leq k$*,* $v_i \in \{\phi_1, \ldots, \phi_{i-1}\}^*$ *and* $v_i\phi_i \in L_{A_Z}$*.*

In the above definition, v_i is a sequence containing only the relations $\phi_1, \ldots, \phi_{i-1}$ that reach a state in a specification from which an arc with ϕ_i is defined. Therefore, V reaches every processing relation in A_Z using sequences of relations that have already been accessed. ¿From Def. 4.5, it follows that a relation cover of Z exists if

for any proper subset Φ_0 of Φ, $L_{A_Z} \setminus \Phi_0^* \neq \emptyset$. This happens if A_Z is accessible and all processing relations in Φ are actually used as labels in A_Z (i.e. $\pi_M(dom(F)) = \Phi$), as can be expected in practice.

Definition 4.6 *Let* $Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ *and* $Z' = (\Sigma, \Gamma, Q', M, \Phi', F', q_0', m_0)$ *be two weak testing compatible SXMs,* $\Phi = \{\phi_1, \ldots, \phi_k\}$ *input-complete,* $V = \{v_1, \ldots, v_k\}$ *a relation cover of* Z *and* t *a test function of* Z. *Then* $X_\Phi = \cup_{i=1}^k t(v_i)\Sigma_{m_i}^i$ *is called a* relation test set of Z w.r.t. Φ' *if for any* $1 \leq i \leq k$, $m_i \in \pi_M(|\ v_i \mid (m_0, t(v_i)))$ *and* $\Sigma_{m_i}^i$ *is an* m_i *conformance test set of* ϕ_i *w.r.t.* Φ' *such that* $\{m_i\} \times \Sigma_{m_i}^i \cap dom(\phi_i) \neq \emptyset$.

For simplicity, in the expression of X_Φ we used $t(v_i)$ instead of $\{t(v_i)\}$. A relation test set of Z exists if a relation cover of Z exists and Φ is input-complete. Due to non-determinism, relations can produce any possible values of memory; in order to test any given relation ϕ_i, it is necessary to attempt all values from $\Sigma_{m_i}^i$ for some value m_i of memory. If $t(v_i)$ reaches different values of memory every time it is attempted, this cannot be done. For this reason, we have to *strengthen the complete-testing assumption* by requiring that there is an output sequence that can always (i.e. eventually) be produced by a NSXM in response to an input sequence. From the observability condition, this ensures the existence of m_i which can be reached $\#\Sigma_{m_i}^i$ times by $t(v_i)$.

The idea behind the construction of a relation test set is to access and test every relation in A_Z using sequences of relations that have already been tested. Therefore, a relation test set is used to test the processing relations of a SXM specification against their implementations, as shown by the result below.

Lemma 4.7 *Let* Z *and* Z' *be two weak testing compatible SXMs having types* Φ *and* Φ', *respectively, such that* Φ *is input-complete, output-distinguishable w.r.t.* Φ' *and observable. If* X_Φ *is a relation test set of* Z *w.r.t.* Φ' *and* $\hbar \mid X_\Phi = \hbar' \mid X_\Phi$ *for some* $\hbar \in \mathcal{H}_Z$, $\hbar' \in \mathcal{H}_{Z'}$, *then there exists a bijective function* $c : \Phi \longrightarrow \Phi'$ *such that for any* $\phi \in \Phi$, $c(\phi) \sqsubseteq \phi$.

Proof. Let $\Phi = \{\phi_1, \ldots, \phi_k\}$, $V = \{v_1, \ldots, v_k\}$ and $X_\Phi = \cup_{i=1}^k t(v_i)\Sigma_{m_i}^i$ be as in Def. 4.6. Let also $c : \Phi \longrightarrow \Phi'$ be as in Def. 4.3. For simplicity, we use $c : \Phi^* \longrightarrow \Phi'^*$ to denote the free-semigroup morphism induced by c. We prove by induction on $1 \leq i \leq k$ the following statement: $c(\phi_i) \sqsubseteq \phi_i$ and $c(v_i\phi_i) \in L_{A_{Z'}}$.

For $i = 1$ this is $c(\phi_1) \sqsubseteq \phi_1$ and $c(\phi_1) \in L_{A_{Z'}}$. Let $\sigma_1 \in \Sigma_{m_0}^1$ and $\gamma_1 \in \Gamma$, such that $\gamma_1 \in \hbar(\sigma_1) \cap \omega_{m_0}^{\phi_1}(\sigma_1)$. Since $\hbar(\sigma_1) = \hbar'(\sigma_1)$, $\exists \phi' \in \Phi'$ such that $\phi' \in L_{A_{Z'}}$ such that $\gamma_1 \in \hbar'(\sigma_1) \cap \omega_{m_0}^{\phi'}(\sigma_1)$, so $\gamma_1 \in \omega_{m_0}^{\phi_1}(\sigma_1) \cap \omega_{m_0}^{\phi'}(\sigma_1)$. Since Φ is output-distinguishable w.r.t. Φ', we have $\phi' = c(\phi_1)$, so $c(\phi_1) \in L_{A_{Z'}}$. Furthermore, we define $o_1 : \Sigma_{m_0}^1 \longleftrightarrow \Gamma$ by $\sigma_1\ o_1\ \gamma_1$ for all σ_1 and γ_1 as above and define $\psi_1, \psi_1' : \Sigma \longleftrightarrow \Gamma$ by $\psi_1(\sigma) = o_1(\sigma)$, $\sigma \in \Sigma_{m_0}^1$, $\psi_1(\sigma) = \omega_{m_0}^{\phi_1}(\sigma)$, $\sigma \in \Sigma \setminus \Sigma_{m_0}^1$, $\psi_1'(\sigma) = o_1(\sigma)$, $\sigma \in \Sigma_{m_0}^1$, $\psi_1'(\sigma) = \omega_{m_0}^{\phi_1'}(\sigma)$, $\sigma \in \Sigma \setminus \Sigma_{m_0}^1$. It is easy to verify that $\psi_1 \sqsubseteq \omega_{m_0}^{\phi_1}$, $\psi_1' \sqsubseteq \omega_{m_0}^{\phi_1'}$ and $\psi_1 \mid \Sigma_{m_0}^1 = \psi_1' \mid \Sigma_{m_0}^1$. Since $\Sigma_{m_0}^1$ is an m_0 conformance test set of ϕ_1 w.r.t. Φ', we have $c(\phi_1) \sqsubseteq \phi_1$.

Assume the statement true for $1 \leq j \leq i-1$, $i > 1$. Let $s_i = t(v_i)$, $g_i \in \Gamma^*$, $\sigma_i \in \Sigma^i_{m_i}$, $\gamma_i \in \Gamma$ such that $g_i\gamma_i \in \hbar(s_i\sigma_i) \cap \omega_{m_0}^{|v_i\phi_i|}(s_i\sigma_i)$. Since Φ is observable, there is an unique $m_i \in M$ such that $(g_i, m_i) \in |\ v_i\ |\ (m_0, s_i)$. ¿From $\hbar(s_i\sigma_i) = \hbar'(s_i\sigma_i)$ it follows that $\exists v_i' \in \Phi'^*, \phi' \in \Phi'$ such that $v_i'\phi' \in L_{A_{Z'}}$ and $g_i\gamma_i \in \hbar'(s_i\sigma_i) \cap \omega_{m_0}^{|v_i'\phi'|}(s_i\sigma_i)$, so $g_i\gamma_i \in \omega_{m_0}^{|v_i\phi_i|}(s_i\sigma_i) \cap \omega_{m_0}^{|v_i'\phi'|}(s_i\sigma_i)$. Since Φ is output-distinguishable w.r.t. Φ', Φ is observable and $\forall 1 \leq j \leq i$, $c(\phi_j) \sqsubseteq \phi_j$, we have (1) $v_i' = c(v_i)$ and (2) $m_i \in M$ is the unique memory value such that $(g_i, m_i) \in |\ c(v_i)\ |(m_0, s_i)$. Furthermore, $c(v_i)\phi' \in L_{A_{Z'}}$ and $\gamma_i \in \omega_{m_i}^{\phi'}(\sigma_i)$, so $\gamma_i \in \omega_{m_i}^{\phi_i}(\sigma_i) \cap \omega_{m_i}^{c(\phi_i)}(\sigma_i)$. Since Φ is output-distinguishable w.r.t. Φ', we have $\phi' = c(\phi_i)$, so that $c(v_i\phi_i) \in L_{A_{Z'}}$. Similarly to the base case, we define $o_i : \Sigma^i_{m_i} \longleftrightarrow \Gamma$ by $\sigma_i\ o_i\ \gamma_i$ for all σ_i and γ_i; $\psi_i, \psi_i' : \Sigma \longleftrightarrow \Gamma$ are defined by $\psi_i(\sigma) = o_i(\sigma)$, $\sigma \in \Sigma^i_{m_i}$, $\psi_i(\sigma) = \omega_{m_i}^{\phi_i}(\sigma)$, $\sigma \in \Sigma \setminus \Sigma^i_{m_i}$, $\psi_i'(\sigma) = o_i(\sigma)$, $\sigma \in \Sigma^i_{m_i}$, $\psi_i'(\sigma) = \omega_{m_i}^{\phi_i'}(\sigma)$, $\sigma \in \Sigma \setminus \Sigma^i_{m_i}$. Then $\psi_i \sqsubseteq \omega_{m_i}^{\phi_i}$, $\psi_i' \sqsubseteq \omega_{m_i}^{\phi_i'}$ and $\psi_i \mid \Sigma^i_{m_i} = \psi_i' \mid \Sigma^i_{m_i}$. Since $\Sigma^i_{m_i}$ is an m_i conformance test set of ϕ_i w.r.t. Φ', we have $c(\phi_i) \sqsubseteq \phi_i$. □

Note that if Φ is observable/output-distinguishable and $(\forall \phi \in \Phi,\ c(\phi) \sqsubseteq \phi)$ then Φ' is also observable/output-distinguishable.

We can now prove the result we are after. Theorem 4.8 is the theoretical basis for the complete NSXM testing method.

Theorem 4.8 *Let Z be a SXM having type Φ and C a set of SXMs weak testing compatible with Z having type Φ' such that Φ is input-complete, output-distinguishable w.r.t. Φ' and observable. If t is a test function of Z, $Y \subseteq \Phi^*$ is a test set of A_Z w.r.t. A_C^{-c}, where $A_C^{-c} = \{A_{Z'}^{-c} \mid Z' \in C\}$ and X_Φ is a relation test set of Z w.r.t. Φ', then $X' = t(Y) \cup X_\Phi$ is a conformance test set of Z w.r.t. C.*

Proof. Let $Z' \in C$ and $\hbar \in \mathcal{H}_Z$, $\hbar' \in \mathcal{H}_{Z'}$ such that $\hbar \mid X' = \hbar \mid X'$. Then we have to prove that $Z' \sqsubseteq Z$. For simplicity, we use c to denote both the bijective function $c : \Phi \longrightarrow \Phi'$ and the free-semigroup morphism $c : \Phi^* \longrightarrow \Phi'^*$ induced by it. ¿From Lemma 4.7 it follows that $\forall \phi \in \Phi$, $c(\phi) \sqsubseteq \phi$. Let $v \in Y$. We prove that $v \in L_{A_Z}$ iff $c(v) \in L_{A_{Z'}}$.

"if": Assume $v \in L_{A_Z}$. Then from $\hbar(t(v)) = \hbar'(t(v))$ it follows that $\exists v' \in \Phi'^*$ such that $\omega_{m_0}^{|v|}(t(v)) \cap \omega_{m_0}^{|v'|}(t(v)) \neq \emptyset$. Since Φ is observable, output-distinguishable w.r.t. Φ' and $\forall \phi \in \Phi$, $c(\phi) \sqsubseteq \phi$, we have $v' = c(v)$, so $c(v) \in L_{A_{Z'}}$.

"only if:" Assume $v \notin L_{A_Z}$ and $c(v) \in L_{A_{Z'}}$. Then let $v = \phi_1 \dots \phi_n$ and let $1 \leq k \leq n$ be such that $\phi_1 \dots \phi_{k-1} \in L_{A_Z}$ and $\phi_1 \dots \phi_k \notin L_{A_Z}$. By definition of t, $(m_0, t(v)) \in dom(|\ \phi_1 \dots \phi_k\ |)$. Since $\forall \phi \in \Phi$, $c(\phi) \sqsubseteq \phi$ we have that $(m_0, t(v)) \in dom(|\ c(\phi_1) \dots c(\phi_k)\ |)$. Since $c(v) \in L_{A_{Z'}}$, we have $c(\phi_1) \dots c(\phi_k) \in L_{A_{Z'}}$, hence from $\hbar(t(v)) = \hbar'(t(v))$ it follows that $\exists v' \in \Phi^*$ such that $v' \in L_{A_Z}$ and $\omega_{m_0}^{|v'|}(t(v)) \cap \omega_{m_0}^{|c(\phi_1)\dots c(\phi_k)|}(t(v)) \neq \emptyset$. Since Φ is observable, output-distinguishable w.r.t. Φ' and $(\forall \phi \in \Phi,\ c(\phi) \sqsubseteq \phi)$, we have $v' = \phi_1 \dots \phi_k$, so $\phi_1 \dots \phi_k \in L_{A_Z}$, which is a contradiction.

Therefore $(v \in Y$ and $v \in L_{A_Z})$ iff $(v \in Y$ and $c(v) \in L_{A_{Z'}})$, hence $(v \in Y$ and $v \in L_{A_Z})$ iff $(v \in Y$ and $v \in L_{A_{Z'}}^{-c})$. Since Y is a test set of A_Z w.r.t. A_C^{-c}, it follows that $v \in L_{A_Z}$ iff $v \in L_{A_{Z'}}^{-c}$. Therefore, $v \in L_{A_Z}$ iff $c(v) \in L_{A_{Z'}}$. We have just proven

that there exists a bijective function $c : \Phi \longrightarrow \Phi'$ such that $\forall \phi \in \Phi$, $c(\phi) \sqsubseteq \phi$ and $c(L_{A_Z}) = L_{A_{Z'}}$. From this it follows easily that $Z' \sqsubseteq Z$. □

Corollary 4.9 *In the conditions of theorem 4.8, if all $\Sigma^i_{m_i}$ are m_i test sets (rather than m_i conformance test sets) of ϕ_i w.r.t. Φ' then $X' = t(Y) \cup X_\Phi$ is a test set of Z w.r.t. C.*

Proof. Since all $\Sigma^i_{m_i}$ are m_i test sets of ϕ_i w.r.t. Φ', from the proof of Lemma 4.7 it follows that $\forall \phi \in \Phi$, $\phi = c(\phi)$. ¿From this and the proof of Theorem 4.8 it follows that $L_{A_Z} = L_{A_{Z'}}$. Hence $f_Z = f_{Z'}$. □

5 Applying complete NSXM testing to P systems

In this section we discuss how Theorem 4.8 (or Corollary 4.9) and Theorem 2.3 can be applied to generate (conformance) test sets for a P system specification. In this respect we show how to build a stream X-machine from a P system. We will use the approach described in [7] to build a derivation tree for a predefined number of steps, k. Using the approach described here we not only develop a new testing method for P systems, but provide a clear way of ensuring that (1) each rule is correctly implemented and (2) each rule or set of rules expected to be applied are actually applied.

The proposed P system testing approach consists of two major steps: (1) deriving a SXM from a P system and (2) applying the NSXM testing method for the resulting model. In order to obtain the SXM model, it is necessary to: (a) build a derivation tree for a given number of steps, k; (b) consider this tree as the underlying structure of the SXM, in which the states of the machine are the nodes of the tree and the arcs are labelled by tuples of multisets of rules (This structure can be minimised as shown in [13]); (c) define the functions from Φ as follows: for each distinct node from the derivation tree there is a derivation step

$$(x_1x'_1, \ldots, x_nx'_n) \Longrightarrow (y_1x'_1, \ldots, y_nx'_n)$$

where x_i, $1 \leq i \leq n$, represents the part of the multiset $x_ix'_i$ which will derive into y_i by applying rules from R_i and x'_i is the multiset which is not changed in this step; if this derivation step is labelled by l then the processing function associated with l is defined by: for every x'_i, $1 \leq i \leq n$, and for given x_i, y_i, $1 \leq i \leq n$, as above,

$$\phi_l((x_1x'_1, \ldots x_nx'_n), (x_1, \ldots x_n)) = ((y_1, \ldots y_n), (y_1x'_1, \ldots y_nx'_n)).$$

The application of the NSXM testing method involves the following detailed steps: (a) checking the conditions imposed for testing (mainly output-distinguishability as discussed below); (b) obtaining test sets for the processing functions from Φ; (c) deriving the overall test set.

As Φ contains only processing functions (rather than relations), observability is outright satisfied. Furthermore, by construction, all processing functions can be exercised (using appropriate inputs) from any memory value that is attainable in the

test generation process, so, for testing purposes, Φ can be considered to be input-complete. Thus, only the output-distinguishability of Φ needs to be examined. The test set of each processing function will typically include the input that triggers the function (in the given memory value) as well as other values (outside the domain of the processing function, for which other transitions will actually be triggered) - this will ensure that the processing functions are correctly implemented. As Φ is a set of processing functions (not relations), these are test sets (rather than conformance test sets). Finally, a test set for the overall system will be derived using Corollary 4.9 and Theorem 2.3.

Note that, in order to properly adapt the NSXM testing approach to the case of P systems, we need to consider a specific testing strategy for SXMs based on breakpoints, which requires stopping the testing process in each state in order to observe the input. This is necessary in this case because we do not have proper inputs and they are identified from the current multisets as being those submultisets that are associated to rewriting rules.

Consider, for example, the two compartment P system
$\Pi = (\{s, a, b, c\}, [_1[_2]_2]_1, s, \lambda, R_1, R_2)$,
where
$R_1 = \{r_1 : s \rightarrow sa(b, in), r_2 : s \rightarrow ab\}$;
$R_2 = \{r_1 : b \rightarrow bc, r_2 : b \rightarrow c\}$
and its derivation tree for $k = 3$. Due to space constraints the derivation tree is not given here. It is not difficult to verify that the example satisfies the output-distinguishability property. Consequently, in this case, we can build a test set based on the methodology presented in this paper for SXMs and, according to this, we can identify at any moment the rule or set of rules applied. However, this property is not true for any P system. Indeed, let us consider a one compartment P system consisting of the following four rules $r_1 : a_1 \rightarrow a_2$, $r_2 : a_2 \rightarrow a_1$, $r_3 : a_1 \rightarrow a_1$, $r_4 : a_2 \rightarrow a_2$ and the initial multiset a_1a_2. In this case we can either apply the multiset r_1r_2 or r_3r_4. Obviously the two functions are associated with different rules and consequently they should be different, but they are not output-distinguishable. Designing P systems for which the output-distinguishability condition is always met will require further investigations.

Finally, note that the approach developed here for testing P systems uses functions instead of relations. In fact, the more general NSXM approach can be applied to P systems by also considering all the branches emerging from the same node as defining a relation.

6 Conclusions

The complete NSXM testing method presented here generalises the NSXM integration testing method. It no longer requires implementations of the processing relations to be proved correct before integration testing can take place. Instead, the testing of processing relations is performed along with the integration testing. This is an important advance since often implementations of the processing rela-

tions are not separate units of code (procedures, methods, etc.) that can be tested in isolation from the rest of the system.

The paper also shows how a SXM model of a P system can be obtained and how the NSXM testing approach can be applied to generate conformance test sets for the P system. Further work will involve devising alternative approaches for deriving SXM models of P systems as well as aspects related to the application of NSXM testing to P systems (e.g. designing P systems that meet the design for test conditions).

References

[1] Bălănescu, T., Gheorghe, M., Ipate, F., Holcombe, M., Formal black box testing for partially specified deterministic finite state machines, *Foundations of Computing and Decision Systems* **28 (1)** (2003) 17–28.

[2] Bogdanov, K., Holcombe, M., Ipate, F., Testing methods for X-machines, a review, *Formal Aspects of Computing* **18(1)** (2006), 3–30.

[3] Cheng, K.-T., Krishnakumar, A.S., Automatic functional test generation using the extended finite state machine model, *Proceedings of the 30th Design Automation Conference* (1993) 86–91.

[4] Chow, T. S., Testing software design modeled by finite state machines, *IEEE Transactions on Software Engineering* **4(3)** (1978) 178–187.

[5] Ciobanu, G., Păun, Gh., Pérez-Jiménez, M. J. (eds), "Applications of membrane computing". Springer, Berlin, 2006.

[6] Eilenberg, S., "Automata, languages and machines", Vol. A, Academic Press, New York, 1994.

[7] Gheorghe, M., Ipate, F., On testing P systems, *Proceedings of WMC* (2008) 173–188.

[8] Hierons, R.M., Harman, M., Testing conformance to a quasi-non-deterministic stream X-machine, *Formal Aspects of Computing* **12(6)** (2000) 423–442.

[9] Hierons, R., Harman, M., Testing conformance of a deterministic implementation against a non-deterministic stream X-machine, *Theoretical Computer Science* **323(1-3)** (2004) 191–233.

[10] Holcombe, M., Ipate, F., "Correct Systems: Building a business process solution. Springer Verlag", Berlin, 1998.

[11] Ipate, F., Complete deterministic stream X-machine testing, *Formal Aspects of Computing* **16(4)** (2004) 374–386.

[12] Ipate, F., Testing against a non-controllable stream X-machine using state counting, *Theoretical Computer Science* **353(1-3)** (2006) 291–316.

[13] Ipate, F., Gheorghe G., Finite state based testing of P systems, *Natural Computing*, to appear.

[14] Ipate, F., Holcombe, M., An integration testing method that is proved to find all faults, *International Journal of Computer Mathematics* **63** (1997) 159–178.

[15] Ipate, F., Holcombe, M., Generating test sequences from non-deterministic generalized stream X-machines, *Formal Aspects of Computing* **12(6)** (2000) 443–458.

[16] Ipate F., Holcombe M., Testing data processing-oriented systems from stream X-machine specifications, *Theoretical Computer Science* **403(2-3)** (2008) 176–191.

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

[18] Luo, G., v. Bochmann, G., Petrenko, A., Test selection based on communicating non-deterministic finite-state machines using a generalised Wp-method, *IEEE Transactions on Software Engineering* **20(2)** (1994) 149–161.

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

[20] Păun, Gh., "Membrane computing. an introduction", Springer, Berlin, 2002.