Journal of Universal Computer Science, vol. 5, no. 9 (1999), 494-507
submitted: 10/6/99, accepted: 15/7/99, appeared: 28/9/99 © Springer Pub. Co.

Communicating Stream X-Machines Systems are no more than X-Machines

Tudor Bălănescu
Faculty of Sciences, Piteşti University, Romania
e-mail balanesc@oroles.cs.unibuc.ro

Anthony J. Cowling
Department of Computer Science, Sheffield University, UK
e-mail: A.Cowling@dcs.shef.ac.uk

Horia Georgescu
Faculty of Mathematics, Bucharest University, Romania
e-mail: hg@oroles.cs.unibuc.ro

Marian Gheorghe
Faculty of Sciences, Piteşti University, Romania
e-mail: marian@oroles.cs.unibuc.ro

Mike Holcombe
Department of Computer Science, Sheffield University, UK
e-mail:M.Holcombe@dcs.shef.ac.uk

Cristina Vertan
Faculty of Mathematics, Bucharest University, Romania
e-mail: cri@oroles.cs.unibuc.ro

Abstract: A version of the communicating stream X-machine model is proposed, which gives a precise representation of the operation of transferring data from one X-machine to another. For this model it is shown that systems of communicating X-machines have the same computational power as single stream X-machines. This enable existing methods for deriving test strategies for stream X-machines to be extended to communicating stream X-machines.

Category: D.2.5, F.1.2, F.3.1

Key Words: Communicating Stream X-machines system, testing, communicating matrix, processing states, communicating states, software specification language.

1 Introduction

Introduced by Eilenberg in 1974 [Eilenberg 74], the X-machine model received little further study until Holcombe [Holcombe 88] used it as basis for a possible specification language. Since then, a lot of further research has been done, proving the power of this model.

An X-machine resembles a finite state machine, but it adds important new features. A basic set X is identified together with a set of basic processing functions Φ. For each state, a finite subset of functions from Φ may emerge from it; if possible, any of these functions may be applied to change the state. The set X characterizes an internal memory for the machine, which also has an input tape and an output tape. Moving from one state to another depends upon the current state, the content of the input tape, the content of the internal memory and the function chosen to be applied. When such a transition takes place, a new item may be added to the output tape.

Unfortunately, very little attention has been paid to the way in which many X-machines may be integrated into a system and how they can communicate.

In [Bălănescu, Georgescu, Gheorghe 99], stream X-machines are used to control a family of distributed grammars. Words of a given language are placed on the input tape. At any time, a single grammar is active; afterwards, it can be used again or the control may be passed to another grammar. The language produced by the system is the language of terminal strings obtained as output. The relationships are studied between the languages used as input and the corresponding languages that are produced, and results concerning the power of these mechanisms are obtained. The above mentioned model simulates the concurrent behaviour of a system of grammars. Another approach, based on cooperating distributed grammar systems for modeling the behaviour of concurrent processes under some synchronization assumptions, is presented in [Bălănescu, Georgescu, Gheorghe 98a], [Bălănescu, Georgescu, Gheorghe 98b], [Bălănescu, Georgescu, Gheorghe 98c].

In [Barnard, Whitworth, Woodward 96] is specified a model for communicating X-machines as an extension of the X-machine model. A communicating X-machine is a typed finite state machine that can communicate with other communicating X-machines via channels that connect ports on each of the machines. A modular system is developed. Another model is proposed in [Vertan 99] where the communication among processes (X-machines) is done through a matrix of communication.

In this paper communicating stream X-machines systems are introduced and it is proved that they can be modeled by an usual stream X-machine [Ipate, Holcombe 96]. Some testing problems are also stated in order to allow the application of some testing strategies already developed in [Chow 78], [Ipate, Holcombe 99], [Luo, von Bochman, Petrenko 94].

2 Basic definitions

For any set A, A^ϵ denotes the set $A \cup \{\epsilon\}$, where ϵ is the empty sequence. $A^\star$ denotes the free monoid generated by A.

Definition 1. A *stream X-machine* is a tuple $X = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m^0)$, where:

- Σ and Γ are finite sets called the *input* and *output alphabets* respectively;
- Q is the finite set of *states*;
- M is a (possibly infinite) set called *memory;*
- Φ is a finite set of *partial functions* of the form: $f : M \times \Sigma \to \Gamma \times M$;
- F is the *next state* partial function $F : Q \times \Phi \to 2^Q$;
- I and T are the sets of *initial* and *final* states;
- m^0 is the *initial memory* value.

We define a *configuration* of the X-machine by (m, q, s, g), where $m \in M, q \in Q, s \in \Sigma^\star, g \in \Gamma^\star$. A machine computation starts from an initial configuration, having the form $(m^0, q^0, s^0, \epsilon)$, where $q^0 \in I$ is an initial state and $s^0 \in \Sigma^\star$ is the input sequence.

Definition 2. *The output corresponding to an input sequence.* A *change of configuration*, denoted by $\vdash$:

$$(m, q, s, g) \vdash (m', q', s', g')$$

is possible if:

- $s = \sigma s'$, $\sigma \in \Sigma$
- there is a function $f \in \Phi$ emerging from q and reaching q', $q' \in F(q, f)$, so that $f(m, \sigma) = (\gamma, m')$ and $g' = g\gamma$, $\gamma \in \Gamma$.

$\overset{\star}{\vdash}$ denotes the reflexive and transitive closure of $\vdash$.

For any $s \in \Sigma^\star$, the output corresponding to this input sequence, computed by the stream X-machine X is defined as:

$$X(s) = \{g \in \Gamma^\star | \exists m \in M, q^0 \in I, q \in T, \text{so that } (m^0, q^0, s, \epsilon) \overset{\star}{\vdash} (m, q, \epsilon, g)\}$$

Definition 3. [Ipate, Holcombe 97] A stream X-machine is called *deterministic* if:

- I contains only one element: $I = \{q^0\}$;
- $F : Q \times \Phi \longrightarrow Q$;
- if f, f' are distinct arcs emerging from the same state, then $dom f \cap dom f' = \emptyset$.

Definition 4. A stream X-machine has the *output distinguishability property* [Ipate, Holcombe 97] when $\forall f_1, f_2 \in \Phi$, if $\exists m \in M, \sigma \in \Sigma$, such that $f_1(m, \sigma) = (g, m'_1)$, $f_2(m, \sigma) = (g, m'_2)$ for some $m'_1, m'_2 \in M$, $g \in \Gamma$, then $f_1 = f_2$.

Definition 5. A stream X-machine is called *minimal* [Holcombe, Ipate 98] if its associated automaton $A = (Q, \Phi, F, I, T)$, is minimal.

Definition 6. Let M be an arbitrary set and $\lambda \notin M$ an arbitrary element. For a natural number $n \geq 1$ we denote by $\mathcal{CM}$ the set of all matrices of order $n \times n$ with elements from $M \cup \{\lambda\}$. Let us consider a matrix $C \in \mathcal{CM}$, a value $v \in M$ and two indices $i \neq j, 1 \leq i, j \leq n$.

- If $C_{ij} = \lambda$, then we define an *output variant* of C, denoted by $(C_{ij} \Leftarrow v)$ as being the matrix with $(C_{ij} \Leftarrow v)_{ij} = v$ and $(C_{ij} \Leftarrow v)_{km} = C_{km}$ for $(k, m) \neq (i, j)$;
- If $C_{ij} = v$, then we define an *input variant* of C, denoted by $(v \Leftarrow C_{ij})$ as being the matrix with $(v \Leftarrow C_{ij})_{ij} = \lambda$ and $(v \Leftarrow C_{ij})_{km} = C_{km}$ for $(k, m) \neq (i, j)$;

Definition 7. *Communicating Stream X-machines Systems.* A Communicating Stream X-machines System (CSXMS) with n components is a system:

$$CSXMS_n = ((P_i)_{i=1,\ldots,n}, \mathcal{CM}, C^0), \text{where:}$$

- $P_i = (X_i, IN_i, OUT_i, in_i^0, out_i^0)$ is a component of the system, $i = 1, \ldots, n$;
- X_i are stream X-machines, $i = 1, \ldots, n$, where Σ_i, Γ_i, M_i, I_i, T_i, m_i^0 are as in the definition 1 and the set of states is $Q_i = Q_i' \cup Q_i''$ with Q_i' being the set of *processing states* and Q_i'' being the set of *communicating states* and $Q_i' \cap Q_i'' = \emptyset$;
- IN_i and OUT_i are two sets called the *input port* and the *output port* respectively of component i. The elements belonging to these sets are values from M_i or the undefined value λ, that is $IN_i, OUT_i \subseteq M_i \cup \{\lambda\}$ and $\lambda \notin M_i$;
- $in_i^0 \in IN_i$ and $out_i^0 \in OUT_i$ are the initial input and output port values, respectively;
- $\mathcal{CM}$ is the set of matrices of order $n \times n$, with elements from $M \cup \{\lambda, @\}$, where $M = \bigcup_{i=1}^n M_i$ and $@ \notin M$. These matrices are used for communication between the X-machines X_i.
- C^0 is the initial communication matrix and $C_{i,j}^0 = \lambda$ for $i \neq j$ and $C_{i,i} = @$;
- for each $i = 1, \ldots, n$, $\Phi_i = \Phi_i' \cup \Phi_i''$, where Φ_i' is the set of *processing functions* and Φ_i'' is the set of *communicating functions*; $\Phi_i' \cap \Phi_i'' = \emptyset$.
  - A processing function $f \in \Phi_i'$:

    $$f : M_i \times IN_i \times OUT_i \times \Sigma_i^\epsilon \longrightarrow \Gamma_i^\epsilon \times M_i \times IN_i \times OUT_i$$

    acts as in an ordinary stream X-machine; additionally it is defined on the IN_i and OUT_i sets too.
  - A communicating function $f \in \Phi_i''$:

    $$f : M_i \times IN_i \times OUT_i \times \Sigma_i^\epsilon \times \mathcal{CM} \longrightarrow \Gamma_i^\epsilon \times M_i \times IN_i \times OUT_i \times \mathcal{CM}$$

    is defined either as

    $$f(m, in, out, \epsilon, C) = (\epsilon, m, in, \lambda, (C_{ij} \Leftarrow out)), \text{for some } j, j \neq i \quad (1)$$

    or

    $$f(m, in, out, \epsilon, C) = (\epsilon, m, in', out, (in' \Leftarrow C_{ji})), \text{for some } j, j \neq i \quad (2)$$
- For each $i = 1, \ldots n$, $F_i : (Q_i' \times \Phi_i') \cup (Q_i'' \times \Phi_i'') \longrightarrow 2^{Q_i}$.

Note 8. The components P_i of the system communicate as described below. For $C \in \mathcal{CM}$ and each pair (i, j) with $i, j \in \{1, \ldots, n\}$, $i \neq j$, C_{ij} is a"message" from the component P_i to the component P_j. The messages are values from M_i. If there is no message, then $C_{i,j} = \lambda$. Initially $C_{ij}^0 = \lambda$, $i \neq j$. There is no need for a component to pass a message to itself; this will be denoted as $C_{ii}^0 = @$, $\forall i \in \{1, \ldots, n\}$. The symbol @ will be used only for this purpose (i.e. C_{ij} can not take the value @, for any $j \neq i$). λ and @ are not memory values for any X-machine of the system. Hence, the actual messages passed from an X-machine to another can not be λ or @. The symbol λ is used to describe an empty buffer C_{ij}.

Each X-machine P_i can read only from the i^{th} column and write only into the i^{th} row of the communication matrix. At any time, a location C_{ij}, $i \neq j$ contains a single piece of information, namely a memory value or the empty value λ.

Note 9. The actions (1) and (2) of a communicating function from Definition 7 enables any two components P_i, P_j to communicate.

- Using C_{ij} as a temporary buffer, P_i can send a message to P_j.The form 1 is used to send the message *out* from P_i to the buffer C_{ij}. This is possible only if $out \neq \lambda$ and $C_{ij} = \lambda$, i.e. the buffer is empty. After completion, an output variant C' of C is obtained, where $C' = (C_{ij} \Leftarrow out)$ with $C'_{kl} = C_{kl}$ for $(k,l) \neq (i,j)$ and $C'_{ij} = out$:
- Using C_{ji} as a temporary buffer, P_i can receive a message from P_j. The form 2 is used to move the value of the non empty temporary buffer C_{ji} to the component P_i. After completion, a new matrix $C' = (in' \Leftarrow C_{ji})$ is obtained, with $C'_{kl} = C_{kl}$ for $(k,l) \neq (j,i)$ and $C'_{ji} = \lambda$.

Note 10. For all $i = 1, \ldots, n$, a state $q \in Q_i$ may be a processing state or a communicating state. All functions emerging from a processing state are processing ones, and all functions emerging from a communicating state are communicating ones.

Note 11. For an uniform treatment, all the processing functions

$$f : M_i \times IN_i \times OUT_i \times \Sigma_i^\epsilon \longrightarrow \Gamma_i^\epsilon \times M_i \times IN_i \times OUT_i$$

will be extended to:

$$\overline{f} : M_i \times IN_i \times OUT_i \times \Sigma_i^\epsilon \times \mathcal{CM} \longrightarrow \Gamma_i^\epsilon \times M_i \times IN_i \times OUT_i \times \mathcal{CM}$$

even though the function does not depend in any way on the current communication matrix and does not change it.

In the sequel, f in place of $\overline{f}$, $f \in \Phi_i$, will be used. Let us note also that a communicating function can only observe the local memory M_i, but it does not change it.

Note 12. The operations concerning the same position (i,j) of the current matrix *C are supposed to be done under mutual exclusion.*

Note 13. If, in a processing state, several functions emerging from it may be applied, then one of them is arbitrarily chosen to act; *if no function can be applied the X-machine blocks,* and so does the entire system.

If, in a communicating state, several functions may be applied, then one of them is arbitrarily chosen; *if no function can be applied, the respective X-machine does not change the state, actively waiting for the moment when at least one function can be applied.*

Definition 14. A *configuration* of a $CSXMS_n$ system has the form

$$z = (z_1, \ldots, z_n, C)$$

where:

- $z_i = (m_i, q_i, in_i, out_i, s_i, g_i)$, $i = 1, \ldots, n$
- m_i is the current value of the memory M_i of P_i;
- q_i is the current state of P_i;
- $in_i \in IN_i$, $out_i \in OUT_i$ are the current port values of P_i;
- $s_i \in \Sigma_i^\star$ is the current input sequence of P_i;
- $g_i \in \Gamma_i^\star$ is the current output sequence of P_i.

– C is the current communication matrix of the system.

The *initial configuration* of the system is $z^0 = (z_1^0, \ldots, z_n^0, C^0)$, where $z_i^0 = (m_i^0, q_i^0, in_i^0, out_i^0, s_i^0, \epsilon)$ with $q_i^0 \in I_i$.

Passing from a configuration z to a new configuration z' ($z \models z'$) supposes that at least one of the X-machines changes its configuration, i.e. a function is applied. A change of configuration:

$$z = (z_1, \ldots, z_n, C) \models z' = (z_1', \ldots, z_n', C') \tag{3}$$

with $z_i = (m_i, q_i, in_i, out_i, s_i, g_i)$, $z_i' = (m_i', q_i', in_i', out_i', s_i', g_i')$, $s_i = \sigma_i s_i'$, $\sigma_i \in \Sigma_i^\epsilon$, $g_i' = g_i \gamma_i$, $\gamma_i \in \Gamma_i^\epsilon$ for any i, may be described as follows. Let $C_0 = C$. For i taking the values $1, 2, \ldots, n$ in this order, there are two possibilities:

– either $z_i = z_i'$, or
– there exists a function $f \in \Phi_i$ emerging from q_i and reaching q_i', $q_i' \in F_i(q_i, f)$ and $C_i \in \mathcal{CM}$ so that $f(m_i, in_i, out_i, \sigma_i, C_{i-1}) = (\gamma_i, m_i', in_i', out_i', C_i)$.

Then $C' = C_n$.

It should be noted that the order chosen above is not important, due to the assumption in note 12.

It is important to mention that such a change in the configuration of one X-machine does not necessarily mean that the other X-machines have done nothing, but rather that they have not entirely completed executing a function. Let us note that these partial actions do not influence the completed ones since the memories M_i and the ports IN_i, OUT_i, $i = 1, \ldots, n$ are local to the components P_i and the access to each location C_{ij} of the current matrix C is done under mutual exclusion.

We can think about a change of configuration $z \models z'$ as follows: let t be the time when the system reached the configuration z and t' the closest following moment of time at which a component terminates the execution of a function; then z' is the configuration of the system at time t'.

Let $\overset{\star}{\models}$ be the reflexive and transitive closure of $\models$.

A *configuration* is a *final* one if $z_i = (m_i, q_i, in_i, out_i, s_i, g_i)$ for any $i = 1, \ldots, n$, with $s_i = \epsilon$ and $q_i \in T_i$.

Definition 15. *The output of a CSXMS corresponding to an input sequence.* For any $s = (s_1, \ldots, s_n) \in \Sigma_1^\star \times \ldots \times \Sigma_n^\star$ we define:
$X(s) = \{g = (g_1, \ldots, g_n) \in \Gamma_1^\star \times \ldots \times \Gamma_n^\star | \exists z^0$ an initial configuration and z a final one, $z^0 \overset{\star}{\models} z$, with $z = (z_1, \ldots, z_n, C)$, $C \in \mathcal{CM}$ and $z_i = (m_i, q_i, in_i, out_i, \epsilon, g_i)$ for any $i = 1, \ldots, n\}$.

Definition 16. A $CSXMS_n = ((P_i)_{1 \leq i \leq n}, \mathcal{CM}, C^0)$ has the *output distinguishability property* when $\forall i, \forall f_1, f_2 \in \Phi_i$, if $\exists m \in M_i$, $in \in IN_i$, $out \in OUT_i$, $C \in \mathcal{CM}$, $\sigma \in \Sigma_i$ such that:
$f_1(m, in, out, \sigma, C) = (g, m_1, in_1, out_1, C_1)$
$f_2(m, in, out, \sigma, C) = (g, m_2, in_2, out_2, C_2)$
for some $m_1, m_2 \in M_i$; $in_1, in_2 \in IN_i$; $out_1, out_2 \in OUT_i$; $C_1, C_2 \in \mathcal{CM}$; $g \in \Gamma$ then $f_1 = f_2$.

3 Example

For a given natural number n, *a sequence of* n *letters* a *and* n *letters* b *has to be produced, so that in each prefix of the sequence the number of* b *does not exceed the number of* a.

We will use a CSXMS with three components P_1, P_2 and P_3. The input sequences for all of them are ϵ.

P_1 successively sends the value 0 to P_3 (in order to inform it that an a has to be written on the output tape of P_3), but from time to time chooses to send to P_2 the number of a it has sent to P_3 since the last transmission to P_2. Just before stopping (when n becomes 0), P_1 sends to P_2 and to P_3 the negative value -1, in order to inform them that no more values will be sent.

P_2 keeps in v the record of the number of b it has to output. At each step, P_2 sends (only if $v > 0$) the value 0 to P_3 (in order to inform it that a b has to be written on the output tape of P_3) or receives from P_1 a value that it adds to v. Just before stopping, P_2 sends to P_3 the negative value -1.

P_3 is the only X-machine that adds elements to its output tape. While P_1 and P_2 are both active, P_3 chooses between receiving a value from P_1 or from P_2 and adds respectively an a or a b to its output tape (of course if the received value is not equal to -1). P_3 stops when P_1 and P_2 have both stopped.

The output tapes of P_1 and P_2 will remain void. The output tape of P_3 will contain the produced sequence.

The state transition diagrams for the three components are presented in [Fig.1], [Fig.2] and [Fig.3] respectively. The communicating states are denoted by double underlined numbers.

The internal memory M_1 of P_1 is $\mathbf{N} \times \mathbf{N}$; its current content is referred as (n, k), where k corresponds to the number of a P_1 has sent to P_3 since the last transmission; initially $k = 0$. We have $q_1^0 = 1$ and $T_1 = \{5\}$. The functions referred to in the state transition diagram (Fig. 1) are defined as follows, where " $-$ " stands for no action.

f_1: **if** $n = 0$ **and** $k = 0$
then $out_1 \leftarrow -1$

f_2: $C_{12} \Longleftarrow out_1$

f_3: $out_1 \leftarrow -1$

f_4: $C_{13} \Longleftarrow out_1$

f_5: **if** $C_{13} = \lambda$ **then** $-$

f_6: $out_1 \leftarrow k\,;\ k \leftarrow 0$

f_7: **if** $n > 0$
then $out_1 \leftarrow 0$

f_8: $C_{13} \Longleftarrow out_1$

f_9: $n \leftarrow n - 1\,;\ k \leftarrow k + 1$

The internal memory of P_2 is $\mathbf{N}$ and its content will be referred as v, with the meaning mentioned above. Initially $v = 0$, $q_2^0 = 1$, $in_2^0 = out_2^0 = 0$ and $T_2 = \{3\}$. The functions referred to in the state transition diagram (Fig. 2) are defined as follows.

g_1: **if**$v = 0$ **and** $in_2 < 0$
then $out_2 \leftarrow -1$

g_2: $C_{23} \Longleftarrow out_2$

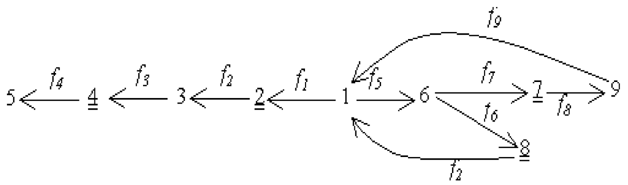

Figure 1: The state transition diagram for P_1

g_3: **if** $v > 0$ **or** $in_2 \geq 0$
then $-$

g_4: $in_2 \Longleftarrow C_{12}$

g_5: **if** $in_2 \geq 0$ **then** $v \leftarrow v + in_2$
else $-$

g_6: $-$

g_7: **if** $v = 0$
then $-$

g_8: **if** $v > 0$
then $v \leftarrow v - 1$; $out_2 \leftarrow 0$

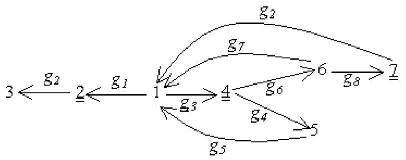

Figure 2: The state transition diagram for P_2

The internal memory M_3 of P_3 is $\mathbf{N}$ and is referred as nr. Initially $nr = 0$, but it will become 1 after P_1 stops and 2 after P_2 stops. $q_3^0 = 1$ and $T_3 = \{2\}$.

h_1: **if** $nr = 2$ **then** $-$

h_2: **if** $nr < 2$ **then** $-$

h_3: $in_3 \Longleftarrow C_{13}$

h_4: **if** $in_3 = 0$ **then** add a to O
else $nr \leftarrow nr + 1$

h_5: $in_3 \Longleftarrow C_{23}$

h_6: **if** $in_3 = 0$ **then** add b to O
else $nr \leftarrow nr + 1$

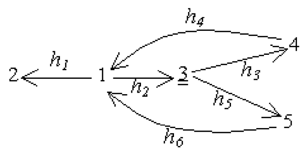

Figure 3: The state transition diagram for P_3

4 CSXMS systems versus stream X-machines

Theorem 17. *For any $CSXMS$, there exists a stream X-machine such that for any given input sequence, the same output will be produced.*

Proof. Let us consider a CSXMS with n components $P_1, \dots P_n$. We construct a stream X-machine

$$X = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m_0)$$

as follows:

- $\Sigma = \Pi_{i=1}^{n} \Sigma_i^{\epsilon}$. An input symbol $(\sigma_1, \dots \sigma_n) \in \Sigma$ has the following meaning: all the machines P_i such that $\sigma_i \in \Sigma_i$ use at the same moment their corresponding current input symbol σ_i.
- $\Gamma = \Pi_{i=1}^{n} \Gamma_i^{\epsilon}$.
- $Q = \Pi_{i=1}^{n} Q_i$.
- The set of initial states is $I = \{(q_i, \dots q_n,)| q_i \in I_i\}$.
- The final state set is $T = \Pi_{i=1}^{n} T_i$.
- $M = \Pi_{i=1}^{n}(M_i \times IN_i \times OUT_i) \times \mathcal{CM}$ is the global memory, containing current memory values, input and output port values of each component P_i and the current communication matrix. The initial memory value is $m_0 = (\Pi_{i=1}^{n}(m_i^0, in_i^0, out_i^0)) \times C^0$ and retains all the initial memory values as well as the initial communication matrix.
- $\Phi = \Pi_{i=1}^{n} \Phi_i^{\epsilon}$. A function $(f_1, \dots f_n) : M \times \Sigma \longrightarrow \Gamma \times M$ from Φ stands for all component functions $f_i \neq \epsilon$ acting in parallel. If $f_i = \epsilon$, then P_i does not change its configuration.
  These functions are defined as follows:
  $(f_1, \dots f_n)(((m_1, in_1, out_1), \dots, (m_n, in_n, out_n), C), (\sigma_1, \dots \sigma_n)) =$
  $((g_1, \dots, g_n), ((m'_1, in'_1, out'_1), \dots, (m'_n, in'_n, out'_n), C'))$
  if:
  1) For any i either $f_i = \sigma_i = g_i = \epsilon$ (no function is applied) or $f_i(m_i, in_i, out_i, \sigma_i, C) = (g_i, m'_i, in'_i, out'_i, C')$ (function f_i is defined for its current arguments and was actually applied).
  2) For any $i \neq j$ with $f_i \neq \epsilon$, $f_j \neq \epsilon$, f_i and f_j are not both communicating functions referring to the same location C_{ij} of the communication matrix.
  This condition assures the mutual exclusion on the same location of C.
  In this definition C' gathers all modification performed on C by the functions $f_1, \dots, f_n$ (see definition 14).

– The transition function $F : Q \times \Phi \longrightarrow 2^Q$ is defined as follows: $(r_1, \ldots, r_n) \in F((q_1, \ldots, q_n), (f_1, \ldots, f_n))$ if either $f_i = \epsilon$ and $r_i = q_i$, or $r_i \in F_i(q_i, f_i)$.

A configuration of the stream X-machine X is, according to definition 1, a tuple:

$$\overline{z} = (((m_i, in_i, out_i)_{i=1,\ldots,n}, C), (q_i)_{i=1,\ldots,n}, (s_i)_{i=1,\ldots,n}, (\gamma_i)_{i=1,\ldots,n}) \qquad (4)$$

From the above construction it follows directly that the stream X-machine works in the same way as the given CSXMS does because:
$z \models z'$ iff $\overline{z} \vdash \overline{z'}$
where z, z' are those from (3), $\overline{z}$ is given by (4) while $\overline{z'}$ is obtained from $\overline{z}$ by replacing each occurrence α by α'.

Note 18. Please note that the stream X-machine built in the proof of this theorem is a slightly extended version of that given in the definition 1. The empty input and the empty output are also allowed in this theorem.

5 Testing conditions

Various testing strategies have been applied to the problem of generating tests for systems which can be modeled by finite state machines. These have been developed for the deterministic case [Chow 78] and for nondeterministic machine models [Luo, von Bochman, Petrenko 94]. Conditions for extending these various testing strategies to the context of (stream) X-machines have also been defined and investigated, again both for the deterministic case [Ipate, Holcombe 97] and for non-deterministic machines [Ipate, Holcombe 99].

Here an analysis is developed for the application of some of the same testing methods to communicating (stream) X-machines. Only the case of deterministic (stream) X-machines is considered.

Lemma 19. *For a given CSXMS having all the components P_i, $1 \leq i \leq n$, deterministic and with the output distinguishability property, it follows that the stream $X-$machine X, constructed in the proof of the theorem 17, is also deterministic and satisfies the output distinguishability property too.*

Proof. In order to show that X is deterministic, it is enough to prove the second requirement of the definition 3. Let us consider the contrary possibility, that there exist

$$(q_1, \ldots, q_n), (q'_1, \ldots, q'_n), (q''_1, \ldots, q''_n) \in Q$$

and

$$(f_1, \ldots, f_n) \in \Phi \text{ such that } (q'_1, \ldots, q'_n) \neq (q''_1, \ldots, q''_n)$$

and

$$\{(q'_1, \ldots, q'_n), (q''_1, \ldots, q''_n)\} \subseteq F((q_1, \ldots, q_n), (f_1, \ldots, f_n)).$$

Then it follows that there exists $1 \leq i \leq n$ such that

$$q'_i \neq q''_i \text{ and } \{q'_i, q''_i\} \subseteq F_i(q_i, f_i).$$

This contradicts the deterministic property of the P_i component.

In order to show that X has the output distinguishability property (see definition 4), let us consider two arbitrary functions

$$f = (f_1, \dots, f_n), \psi = (\psi_1, \dots, \psi_n)$$

and suppose that there exist

$$m = ((m_1, in_1, out_1), \dots, (m_n, in_n, out_n), C) \in M, \sigma = (\sigma_1, \dots \sigma_n) \in \Sigma$$

such that

$$f(m, \sigma) = (g, m'),\ \psi(m, \sigma) = (g, m'')$$

for some
$m' = ((m'_1, in'_1, out'_1), \dots (m'_n, in'_n, out'_n), C')$,
$m'' = ((m''_1, in''_1, out''_1), \dots (m''_n, in''_n, out''_n), C'') \in M$ and
$g = (g_1, \dots g_n) \in \Gamma$. Then $f_i = \psi_i$, $1 \leq i \leq p$ (according to the output distinguishability property (see definition 16) of each component P_i), and so $f = \psi$.

It should be noted that, in the above proof, some of the elements occurring on the same positions in σ, g, f and ψ may be ϵ.

Remark. If for a word x, the set of distinct symbols occurring in x is denoted by $Alph(x)$, then for any sequences

$$x_1, \dots, x_p \text{ and } y_1, \dots, y_q, \text{ such that } x_1 \dots x_p \neq y_1 \dots y_q$$

it follows that for any sequences

$$z_1, \dots, z_{p+1} \text{ and } w_1, \dots, w_{q+1}, \text{ such that}$$

$$Alph(z_1 \dots z_{p+1} w_1 \dots w_{q+1}) \cap Alph(x_1 \dots x_p y_1 \dots y_q) = \emptyset$$

we have

$$z_1 x_1 \dots x_p z_{p+1} \neq w_1 y_1 \dots y_q w_{q+1}$$

Note 20. Paths in CSXMS. Let $CSXMS_n = ((P_i)_{1 \leq i \leq n}, \mathcal{CM}, C^0)$ be a CSXMS with n components, let $X = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, \overline{m^0})$ be the X-machine built in the proof of theorem 17 and let us consider a component P_i, $1 \leq i \leq n$. For $q, r \in Q_i$, $[q, r]$ denotes a path from q to r if there exist $s_j \in Q_i$, $0 \leq j \leq p$, $f_j \in \Phi_i$ $1 \leq j \leq p$, $s_j = F_i(s_{j-1}, f_j)$, $1 \leq j \leq p$; $s_0 = q$ and $s_p = r$.

For $q_1, r_1 \in Q$, $[q_1..r_1]$ denotes a path from q_1 to r_1 *associated* to $[q, r]$ if there exist $(s_{1,j}, \dots, s_{n,j}) \in Q$, $0 \leq j \leq t_p$; $(f_{1,j}, \dots, f_{n,j}) \in \Phi$, $1 \leq j \leq t_p$, $(s_{1,j}, \dots, s_{n,j},) \in F((s_{1,j-1}, \dots, s_{n,j-1}), (f_{1,j}, \dots, f_{n,j}))$, $1 \leq j \leq t_p$ and $s_{i,0} = q$; $s_{i,t_j} = s_j$, $1 \leq j \leq p$, $1 \leq t_1 \leq t_2 \leq \dots \leq t_p$; $q_1 = (s_{1,0}, \dots, s_{n,0})$; $r_1 = (s_{1,t_p}, \dots, s_{n,t_p})$.

Lemma 21. *Let us consider a $CSXMS_n$ with all the components P_i, $1 \leq i \leq n$, minimal and such that for any component P_i, $1 \leq i \leq n$ and any path $[q, r]$, $q \in Q_i$, $r \in T_i$, there exists in the X-machine X, considerd in the proof of theorem 17, the associated path $[q_1..r_1]$, $q_1, r_1 \in Q$, and there exists a path from r_1 to a final state, then X is minimal too.*

Proof. Let us consider two distinct states $q = (q_1, \dots q_n)$ and $q' = (q'_1, \dots q'_n)$. It follows that there exists $1 \leq i \leq n$ such that $q_i \neq q'_i$. P_i being a minimal $X-$machine, it results that there exist $r_i, r'_i \in Q_i$ such that the sequences of basic functions from Φ_i corresponding to $[q_i, r_i]$ and $[q'_i, r'_i]$, namely $f_{i,1} \dots f_{i,t}$, and $\psi_{i,1} \dots \psi_{i,m}$, respectively, accepted by the associated automaton A_i (definition 5) are distinct. In the $X-$machine built in the proof of theorem 17 there exist the associated paths $[q_1..r_1], [r_1..r_2]$ and $[q'_1..r'_1], [r'_1..r'_2]$; $r_2, r'_2 \in T$ (see note 20) and accordingly the sequences of functions from Φ

$$f_1 \dots f_a \text{ and } \psi_1 \dots \psi_b.$$

Then, there exist $1 \leq c_j \leq a$, $1 \leq j \leq t$ and $1 \leq d_j \leq b$, $1 \leq j \leq m$ such that each $f_{c_j} \in \Phi$ has the i^{th} component the basic function $f_{i,j} \in \Phi_i$, $1 \leq j \leq t$ and each $\psi_{d_j} \in \Phi$ has the i^{th} component the basic function $\psi_{i,j} \in \Phi_i$, $1 \leq j \leq m$. The other basic functions $f_j \in \Phi$ and $\psi_h \in \Phi$ do not contain as components any occurrence of $f_{i,g}$, $1 \leq g \leq t$ and $\psi_{i,g}$, $1 \leq g \leq m$, respectively. According to the remark following lemma 19, the strings $f_1 \dots f_a$ and $\psi_1 \dots \psi_b$ are distinct.

Theorem 22. *If a CSXMS has its components deterministic and the X-machine built in the proof of theorem 17 fulfills the constraints imposed by lemma 21, then this X-machine is deterministic and minimal.*

Proof. It is an immediate consequence of lemmas 19 and 21.

Remark. – The complexity of the testing algorithm and of test data set is provided in the case of a CSXMS with n components having the same number of states, input symbols, and basic functions. Also the estimated number of additional states for each component implementation is no larger than k. If $m \geq n$ is the number of states of the resulted X-machine and $s = card(Q_i)$, $p = card(\Sigma_i)$, $r = card(\Phi_i)$ and the complexity of each basic function is less than α, then, similar to [Holcombe, Ipate 98], [Ipate, Holcombe 97] the following values are obtained:

- the upper limit of the number m of states of the X-machine is given by s^n (see the proof of the theorem 17)
- the complexity of the algorithm that generates the test set will be

$$\alpha((p+1)^n - 1)((r+1)^n - 1)^{k+1} m^2 (m + 2k)$$

- the maximum number of test sequences required is less than

$$m^2((r+1)^n - 1)^{k+1}$$

- the total length of the test set is less than

$$m^2(m+k)((r+1)^n - 1)^{k+1}$$

For estimating the number of input symbols and basic functions the following obvious relation has been used

$$\binom{1}{n}x + \binom{2}{n}x^2 + \dots \binom{n}{n}x^n = (x+1)^n - 1$$

– Some hints for decreasing the complexity of the testing algorithm are provided in [Barnard, Whitworth, Woodward 96] in the case of the reachability tree.

6 Conclusions

While in [Barnard, Whitworth, Woodward 96] is presented the basic concept of assembling systems from communicating X-machines, they did not developed this to the point of deriving the input-output relationships for such systems. By extending their model, this paper has been able to derive this result, in the form of definition 15.

From this, it has been demonstrated in theorem 17 that the process of assembling a system in this way does not generate any additional computational power, in that if the individual components are each modeled as stream X-machines, then the complete system can also be modeled as a stream X-machine, which can be constructed as in the proof of that theorem. Furthermore, it has also been shown (in lemma 19 and theorem 22) that important properties of the individual component machines, such as determinism, minimality and output distinguishability, will under suitable conditions all carry over to the system as a whole.

The reason why these results are important is that the process of assembling a CSXMS from component X-machines in this way is intended to model precisely the process of constructing software systems from sub-systems which interact by exchanging data amongst themselves. Of course, the particular model of CSXMS that is presented here is not the only possible formulation, and other variants of it still need to be explored [Vertan 99]. One aim of this will be to find the best way of representing the various features of the assembly process that need to be captured in such a model. It will also be necessary to establish the conditions under which the results developed here will apply to these others versions of the model as well.

Given that the model considered here does represent the assembly process in this way, the significance of the results concerning the equivalence of computational power is that they give software designer the freedom to choose the level of detail at which to apply the X-machine model to any particular software system. Thus, for a system which is assembled from a number of components in this way, a designer can either choose to build an X-machine model of each component separately, and the assemble them into a CSXMS, or they can choose to build a single X-machine model of the whole system. In practice, of course, they are likely to adopt the first approach, as corresponding more closely to the actual software design process, but theorem 17 assures that models built by either of these two approaches will be equivalent, provided of course that they have been built correctly.

Indeed, the problem of ensuring the correctness of such models is a central issue in software design. Software testing is an important aspect of this issue, and this paper has been able to extend to the CSXMS model some of the testing results that have been established in [Holcombe, Ipate 98], [Ipate, Holcombe 97] for individual deterministic X-machines.

One eventual aim of this extension is to provide a basis for characterising the test set for a CSXMS in terms of the test sets for the component X-machines, so as to be able to distinguish the properties that must be tested for the CSXMS as a whole from those which can be established by testing the components separately. The results presented here provide a fundamental basis for future work aimed at extending the approach developed in [Holcombe 88] and [Holcombe, Ipate 98], so as to tackle the problem of defining rigorously the activity of integration testing for systems that are built in this way. This activity lies at the heart of any attempt to assemble software components into systems that will be reliable and correct, and the results that are presented here are crucial to it.

References

[Bălănescu, Georgescu, Gheorghe 98a] T. Bălănescu, H. Georgescu and M. Gheorghe, Guarded Aditive Valence Grammars as Models for Synchronization Problems, *Annals of Bucharest University, Mathematics-Informatics series*, 47, 1 (1998), 19-26.

[Bălănescu, Georgescu, Gheorghe 98b] T. Bălănescu, H. Georgescu, and M. Gheorghe, On Counting Derivation in Grammar Systems, *Romanian Journal of Information Science and Technology*, Romanian Academy Press, 1, 1 (1998), 23-42.

[Bălănescu, Georgescu, Gheorghe 98c] T. Bălănescu, H. Georgescu, and M. Gheorghe, Grammatical Models for Some Process Synchronizers, *Proceedings of the MFCS'98 Satellite Workshop on Grammar Systems*, Silesian University, Brno (1998), 117 - 137.

[Bălănescu, Georgescu, Gheorghe 99] T. Bălănescu, H. Georgescu, and M. Gheorghe, Stream X-Machines with Underlying Distributed Grammars, submitted to *Informatica* (1999).

[Barnard, Whitworth, Woodward 96] J. Barnard, J. Whitworth, M. Woodward, Communicating X-machines, Journal of Information and Software Tehnolog., 38, 6 (1996), 401-407.

[Chow 78] T.S. Chow, Testing software design modeled by finite-state machines, *IEEE Trans on SE*, 4, 3 (1978), 178-187.

[Eilenberg 74] S. Eilenberg, *Automata, languages and machines*, Academic Press (1974).

[Holcombe 88] M. Holcombe, X-Machines as Basis for Dynamic System Specification, *Software Engineering Journal*, 3, 2 (1988), 69-76.

[Holcombe, Ipate 98] M. Holcombe and F. Ipate, *Correct Systems: Building a Business Process Solution,* Springer Verlag, Berlin (1998).

[Ipate, Holcombe 96] F. Ipate and M. Holcombe, Another look at computability, *Informatica,* 20 (1996), 359-372.

[Ipate, Holcombe 97] F. Ipate and M. Holcombe, An Integration Testing Method That is Proved to Find all Faults, *Intern. J. Computer Math,* 63 (1997), 159-178.

[Ipate, Holcombe 99] F. Ipate and M. Holcombe, Generating test sequences from nondeterministic X-machines, submitted to *FACS* (1999).

[Luo, von Bochman, Petrenko 94] G. Luo, G. von Bochman and A. Petrenko, Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized Wp-Method, *IEEE Trans on SE*, 20, 2 (1994), 149-161.

[Vertan 99] C. Vertan: A New Approach to Communicating X-Machines Systems, submitted to *J. UCS* (1999).