STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIII**, Number 1, 1998

A NEW APPROACH TO COMMUNICATING X-MACHINES SYSTEMS

CRISTINA VERTAN

Abstract. This paper presents a new model for the specification of communicating X-machines systems (CXMS). Each X-machine has its own local memory. An unique output tape is used. The X-machines act simultaneously. The states of each component of the system are partitioned into ordinary and communication states. Passing messages between the X-machines involves only communication states. It is shown that, taking advantage of the behaviour of X-machines, communication using channels may be implemented, thus providing a synchronized message passing.

Keywords: Communicating X-machines, concurrent processes, communication using channels

1. Introduction

Introduced by Eilenberg in 1974 ([4]), the X-machines received little further study until Holcombe ([6]) used them 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 new important 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. An input tape, an output tape and an internal memory are specified. 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 in a system and how they can communicate.

In [1], 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 generated by the system is the language of

1991 *Mathematics Subject Classification.* 68Q60, 68Q68, 68Q22.

terminal strings obtained as output. Relations between languages used as input and the corresponding generated languages are studied and results concerning the generative power of the grammars are obtained. This model simulates the concurrent behaviour of a system of grammars.

Barnard ([2]) 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. The communicating X-machine model encapsulates dynamic and functional behaviour, as well as the data model, in one process specification. Message passing using channels is not necessarily synchronous.

In this paper, the above ideas are continued. In the second section a more precise model of communicating X-machines is introduced. Each X-machine has its own local memory, while all components share the same output tape. Any X-machine may pass messages to any other one. The states of each component of the system are partitioned into ordinary and communication states. Passing messages between the X-machines involves only communication states; for functions emerging from a communication state, the local memory may only be observed, but never changed. In this way, internal behaviour and communication are separated. Links between components can be disabled. In the third section, (synchronous) channels are introduced as a way the X-machines can communicate. Basic operations for sending and receiving messages are implemented.

The suitability of the new approach is proved by a number of case studies specifying various problems occurring in the concurrent processing area.

2. Communicating X-Machines Systems

A Communicating X-Machines System ($CXMS$ for short) with n components is a 4-uple $CXMS_n = ((P_i)_{i=1,...,n}, C, C^0, O)$, where:

- **-:** P_i is the X-machine with number i;
- **-:** C is a matrix of order $n \times n$, used for communication between the X-machines;
- **-:** C^0 is the initial content of C;
- **-:** O is the output tape of the system, initially void.

For each pair (i, j) with $i, j \in \{1, ..., n\}$, $i \neq j$, C_{ij} is used as a temporary buffer for passing "messages" from the X-machine P_i to the X-machine P_j. Initially, $C_{ij} = C^0_{ij}$ has one of the values λ or @, as passing messages from P_i to P_j is intended or not. For all i, $C_{ii} =$ @ because an X-machine never passes a message to itself. The actual messages passed from an X-machine to another can not be λ, @ or \$, which are used for special purposes, further described. The mechanism of passing a value (message) will be explained in detail later. For the moment,

we mention only that each X-machine P_i can access (read from or write into) only the i^{th} column and i^{th} row of the communication matrix. We will denote this accessibility domain by $+_i$. A location C_{ij} may receive the value @, meaning that the connection from P_i to P_j is disabled. A disabled connection can not be enabled later. At any time, a location C_{ij} can contain a single piece of information.

Each X-machine P_i is a 9-uple:
$P_i = (Q_i, M_i, in_i, out_i, \Phi_i, F_i, q_i^0, T_i, M_i^0)$, where:

- Q_i is the finite set of states of P_i;
- M_i is the local memory of P_i. The local memories replace the input tape;
- in_i and out_i are two additional distinct memory locations used for receiving and transmitting messages; their initial value is λ; at any time they can contain a single piece of information, of the same type as those ones in C;
- q_i^0 is the initial state;
- T_i is the set of final states; there is no function emerging from a final state;
- M_i^0 is the initial content of the local memory;
- Φ_i is the set of functions applied when moving from one state to another one;
- F_i is the transition function $F_i : Q_i \times \Phi_i \to 2^{Q_i}$.

Remark 1. *For sake of simplicity, in the above definition we suppose that all messages passed from any X-machine to any other one have the same type. This does not restrict the generality of the model, since any message could begin with a flag indicating the type of the message. This flag could be used by the receiver in order to decode correctly the message.*

In each X-machine P_i there are two kinds of states: $Q_i = Q_i^{'} \cup Q_i^{"}, Q_i^{'} \cap Q_i^{"} = \emptyset$, where $Q_i^{'}$ contains *ordinary states* and $Q_i^{"}$ contains *communication states*. In the diagrams below, any state x will be represented as $\underline{x}$ (if it is an ordinary state), as $\underline{\underline{x}}$ (if it is a communication state) or as x (if it can be either an ordinary or a communication state). The final states are ordinary states.

Let $\underline{\underline{x}}$ be a communication state of the X-machine P_i, let $f_1, \ldots, f_k \in \Phi_i$ be the functions emerging from it and let $y_1, \ldots, y_k \in Q_i$ be their destinations, as in Fig. 1. Then any function f_s is defined as follows:

$$f_s : +_i \times M_i \times in_i \times out_i \to +_i \times in_i \times out_i$$

and may have one of the following meanings and forms:

1): a value is moved from out_i to C_{ij}, for some $j \neq i$:
if $condition_s \,\&\, C_{ij} = \lambda \& out_i \neq \lambda$

FIGURE 1. States and functions emerging from them.

then $C_{ij} \leftarrow out_i$, $out_i \leftarrow \lambda$
where $condition_s$ depends on M_i ;
2): a value is moved from C_{ji} to in_i, for some $j \neq i$:
if $condition_s \,\&\, C_{ji} \notin \{\lambda, @\}$
then $in_i \leftarrow C_{ji}, C_{ji} \leftarrow \lambda$
where $condition_s$ depends on M_i ;
3): under some condition, some elements of $+_i$ are modified:
if $condition_s$ **then** $modify +_i$,
where $condition_s$ involves elements in the domain of f_s and the modifications consist only in changing some elements of $+_i$ into $@, \lambda$ or \$.

Remark 2. *For functions emerging from a communication state, the local memory may be only observed, but not changed.*

If more than one of the functions $f_1, \ldots, f_k$ may be applied, one of them is chosen arbitrarily to act. If none of these functions may be applied, the X-machine does nothing (so it does not change the state).

Let now $\underline{x}$ be an ordinary state, which is not a final one, of the X-machine P_i, let $f_1, \ldots, f_k \in \Phi_i$ be the functions emerging from it and let $y_1, \ldots, y_k \in Q_i$ be their destinations, as in Fig. 1 b. Then any function f_s is defined as follows:

$$f_s : M_i \times in_i \times out_i \rightarrow M_i \times in_i \times out_i \times O$$

and is meant to (partially) change the content of M_i, in_i, out_i and possibly add some information to the output tape O. We will suppose that at any time at most one X-machine can write on the output tape, i.e. the writing operations are serialized. If more than one of the functions $f_1, \ldots, f_k$ may be applied, one of them is chosen arbitrarily to act. If none of these functions may be applied, the X-machine blocks and so does the entire system; the content of the output tape is not significant in this case.

The system starts with all X-machines in their initial states, $C = C^0$, $M_i = M_i^0$, $in_i = \lambda$ and $out_i = \lambda$ for all $i \in \{1, \ldots, n\}$. The X-machines act simultaneously. The system stops successfully when all X-machines reach final states; in

this case the result is the content of the output tape.

From the above definitions, it follows that a $CXMS$ is nondeterministic. The nondeterminism is provided in two ways:

-: by means of the communication states and the matrix C;
-: by means of the ordinary states' behaviour of each X-machine.

An X-machine P_i in a $CXMS$ is called *deterministic with respect to ordinary states* (for short *OS-deterministic*) if:

1): $F_i : Q_i \times \Phi_i \to Q_i \ \forall i = 1, \ldots, n$;
2): for any ordinary state, any content of the local memory and any content of the two additional memory locations (*in* and *out*), exactly one function can be applied.

Example 3. *For a given 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 two X-machines P_1 and P_2. P_1 successively adds a to the output tape O, but from time to time chooses to send to P_2 the number of a it has added to O since the last transmission. P_2 keeps in v the record of the number of b it has to output; at each step, P_2 outputs a b (if $v > 0$) or receives from P_1 a value that it adds to v.

The initial form C^0 of the communication matrix C is:

$$\begin{array}{cc} @ & \lambda \\ \lambda & @ \end{array}$$

In P_1, the internal memory M_1 contains the variables n and k, where k corresponds to the number of a P_1 has output since the last transmission; initially $k = 0$. We have $q_1^0 = \underline{1}$ and $T_1 = \{\underline{3}\}$. The state transition diagram appears in Fig. 2 a. The sign "$-$" in the description of functions stands for no action.

f_1: ***if*** $n = 0 \,\&\, k = 0$ ***then*** $-$ $\qquad$ f_2: $C_{12} \leftarrow @$

f_3: ***if*** $n > 0$ ***then*** $-$ $\qquad$ f_4 : *add a to O;* $n \leftarrow n - 1$; $k \leftarrow k + 1$

f_5: $out_1 \leftarrow k$; $k \leftarrow 0$ $\qquad$ f_6: ***if*** $C_{12} = \lambda \,\&\, out_1 \neq \lambda$ ***then*** $C_{12} \leftarrow out_1$; $out_1 \leftarrow \lambda$

f_7: ***if*** $C_{21} = \$$ ***then*** $C_{21} \leftarrow \lambda$

The internal memory of P_2 contains the variable v mentioned above; initially $v = 0$, $q_2^0 = \underline{1}$ and $T_2 = \{\underline{2}\}$. The state transition diagram is showed in Fig. 2 b.

g_1: ***if*** $v = 0 \,\&\, C_{12} = @$ ***then*** $-$ g_2: ***if*** $C_{12} \notin \{\lambda, @\}$ ***then*** $in_2 \leftarrow C_{12}$; $C_{12} \leftarrow \lambda$

g_3: $v \leftarrow v + in_2$ g_4: $C_{21} \leftarrow \$$

g_5: ***if*** $v > 0$ ***then*** $-$ g_6: *add b to O;* $v \leftarrow v - 1$

FIGURE 2. Example 3. The state transition diagrams for: a) P_1; b) P_2

Since no mutual exclusion or other synchronizations are assumed when working with the common memory C, the handling of this matrix has to be done carefully. For example let us suppose that in function f_7 we do not assign λ to C_{21}. Even for $n = 1$, the following two scenarios (interleavings) fail to produce the sequence ab to the output:
1) P_1 sends the value 0 to P_2. P_2 receives it, assigns \$ to C_{21}, so that the condition in the function f_7 will always be *true*; v is 0 and P_2 remains in state $\underline{\underline{1}}$ for a while. P_1 writes a and assigns $k = 1$ to C_{12}. n and k are now 0 and P_1 enters the final state, so that $C_{21} = @$. P_2 awakes and has a single possibility: to move to the final state. In this way a single a is output.
2) The second scenario resembles the first one, but P_2 awakes before P_1 sets C_{21} to @. P_2 chooses to receive a value (function g_2 begins to be executed), but meanwhile C_{21} becomes @, so that in_2 is set now to @. But incrementing v with @ is meaningless and may lead to unpredictable results.

The discussion above shows the necessity of introducing a more structured way to handle sending and receiving messages.

3. Communicating X-machines systems using channels

The mechanism introduced above assures only a low level of synchronization. In this paragraph we will introduce channels as a higher level of synchronization. The mechanism resembles that found in Occam (INMOS, 1984) and the formalism CSP (see [5]). The CXM systems prove to be a natural way for implementing intercommunication between the components, namely through *channels*.

The classical communication through channels is described further. It involves *send* and *receive* operations; the operations on each channel are synchronized. Each channel has a single sender and a single receiver. Whichever process reaches first a state where a channel operation is applied, will be blocked until the process at the other end of the channel reaches the complementary operation. When both processes are ready, a rendezvous is said to take place, with data passing from the output of the sender to the input of the receiver. Only after this message passing is complete can the two processes act further.

We will simulate this kind of communication with X-machines. The special symbol \$, mentioned at the beginning of section 2, will be used. Let us suppose that we intend to send the content of out_i to in_j (of course using C_{ij}) in the same way as a transmission through channels is done (see [3]). The functions emerging from a communication state are restricted to the following two forms:

a): when *condition* $=>$ $j\,!\,out_i$
for some $j \neq i$, where *condition* depends only on M_i;

b): when *condition*$'$ $=>$ $j\,?\,in_i$
for some $j \neq i$, where *condition*$'$ depends only on M_i;

In fact these are *macrofunctions*. Their diagrams are showed in Fig. 3 a and Fig. 3 b, where:

f_1: **if** *condition* $\&\ C_{ij} = \lambda\ \&\ out_i \neq \lambda$
then $C_{ij} \leftarrow out_i$; $out_i \leftarrow \lambda$

f_2: **if** $C_{ji} = \$$
then $C_{ji} \leftarrow \lambda$

g_1: **if** *condition*$'$ $\&\ C_{ij} \neq \lambda$
then $in_j \leftarrow C_{ij}$; $C_{ij} \leftarrow \lambda$

g_2: $C_{ji} \leftarrow \$$

g_3: **if** $C_{ji} = \lambda$
then $-$

It is important to stress the fact that the conditions appearing in a) and b) are included in f_1 and g_1. In this way even if the condition is fulfilled it does not mean that the function may be chosen without further checking. We will assume that it is the programer's duty to ensure, when the CXM system is working, that for each channel operation the complementary one is provided. The situation when two X-machines try simultaneously to send or simultaneously to receive messages between them has to be avoided.

FIGURE 3. State diagrams for implementing channels for inter-communication between the components of a $CXMS$.

Proposition 4. *Under the above assumptions the simulation of channels for X-machines works correctly.*

Proof. Let us suppose that the two X-machines involved in communication are P_i and P_j. Initially $C_{ij} = C_{ji} = \lambda$. We recall that only P_i and P_j can modify C_{ij} and C_{ji}. Let us assume that P_i chooses to send a value to P_j and $condition = true$. Then, the only possible sequence is: f_1, g_1, g_2, f_2. Two cases are possible:
I) If P_j executes g_3 then the send and receive operations are completed.
II) There is a delay in P_j before the execution of g_3 (P_j sleeps for a while). In this moment $C_{ji} = \lambda$.
According to the possible actions of P_i the following cases have to be studied:

1) P_i will not communicate again with P_j; when P_j awakes it will execute g_3.

2) P_i tries again to send a value to P_j and $condition = true$ and $C_{ij} = \lambda$. P_i is blocked on f_2, so when P_j awakes it will execute g_3. Following the assumption that to every *send* operation a *receive* is associated, P_j will try again to receive a value from P_i. This value will be received while executing the function g_1; then P_j will execute g_2, so there $C_{ji} \leftarrow \$$. P_i can now resume execution.

3) P_i tries to receive a value from P_j. Consequently it will try to execute the following sequence of functions:

g_1: **if** $condition'' \,\&\, C_{ji} \neq \lambda$ **then** $in_i \leftarrow C_{ji}$; $C_{ji} \leftarrow \lambda$

g_2: $C_{ij} \leftarrow \$$

g_3: **if** $C_{ij} = \lambda$ **then** $-$

As $C_{ji} = \lambda$, P_i will be blocked on g_1; when P_j awakes, it will execute g_3.

Example 5 (The Producer-Consumer problem with bounded queue)**.** *A producer produces items and places them into a buffer of finite length. The consumer takes items from the buffer and consumes them. The constraints are the following:*

- *produce must always precede consume;*
- *the consumer takes the items from the buffer in the same order they were placed, i.e. the buffer is a queue;*
- *reading from an empty buffer must be avoided;*
- *writing in a full buffer must be avoided too.*

We will suppose that these items are characters. The producer stops after sending the first character z, and the consumer stops after receiving z. The output tape

will contain the characters received by the consumer.

The problem will be modelled by means of a CXMS with 3 components: P_1, P_2 *and* P_3. *The initial form* C^0 *of the communication matrix* C *is:*

$$\begin{array}{ccc} @ & \lambda & @ \\ \lambda & @ & \lambda \\ @ & \lambda & @ \end{array}$$

P_1 *corresponds to the producer.* M_1^0 *contains the items that the producer places in the queue. We have* $q_1^0 = \underline{1}$ *and* $T_1 = \{\underline{5}\}$. *The state transition diagram for* P_1 *appears in Fig. 4 a.*

FIGURE 4. The Producer-Consumer problem. State transition diagrams for: a)P_1; b)P_2 c)P_3

f_1: $out_1 \leftarrow first(M_1)$;
$M_1 \leftarrow tail(M_1)$;

f_2: ***if*** $out_1 = z$ ***then*** $-$

f_3: ***if*** $out_1 \neq z$ ***then*** $-$

f_4: $2!out_1$

P_3 *models the activities of the consumer.* $M_3^0 = \emptyset$, $q_3^0 = \underline{\underline{1}}$ *and* $T_3 = \{\underline{4}\}$. *The state transition diagram is showed in Fig. 4 c.*

g_1: $1\,?\,in_3$ $\qquad$ g_2: *add* in_3 *to* O

g_3: ***if*** $in_3 = z$ ***then*** $-$ $\qquad$ g_4: ***if*** $in_3 \neq z$ ***then*** $-$

The X-machine P_2 implements the activities concerning the buffer. Let max be the size of the queue Q, and ok an integer variable initialized with 2. Variable ok will decrease to 1 after the character z is received from P_1 and will decrease to 0 when the same character is sent to P_3. The internal memory of P_2 includes Q, max, ok and nr, where nr is the current number of items in Q. We have $q_2^0 = \underline{\overline{1}}$ and $T_2 = \{\underline{5}\}$. "$\Leftarrow$" is the operator used for extracting an item from the queue Q, while "$\Rightarrow$" is the operator used for adding an item to the same queue. The state transition diagram appears in Fig. 4 b.

g_1: ***if*** $ok = 0$ ***then*** $-$ $\qquad$ g_2: ***if*** $ok \neq 0$ ***then*** $-$

g_3: ***when*** $ok = 2 \,\&\, nr < max$
$\quad => 1\,?\,in_2$ $\qquad$ g_5: ***when*** $out_2 \neq \lambda$
$\quad => 3\,!\,out_2$

g_4: $in_2 \Rightarrow Q;\ nr \leftarrow nr + 1$
$\quad$ ***if*** $in_2 = z$ ***then*** $ok \leftarrow ok - 1$
$\quad$ ***if*** $out_2 = \lambda$
$\quad$ ***then*** $out_2 \Leftarrow M_2;\ nr \leftarrow nr - 1$

g_6: ***if*** $out_2 = z$ ***then*** $ok \leftarrow ok - 1$
$\quad$ ***if*** $nr > 0$
$\quad$ ***then*** $out_2 \Leftarrow M_2;\ nr \leftarrow nr - 1$

Example 6 (Finding the first n prime numbers)**.** *We will introduce a CXM system with $n + 2$ components, in fact a pipeline of X-machines labeled with $P_0, P_1, \ldots, P_{n+1}$.*
The main activity of the X-machine P_0 is to pump the numbers 2,3,... to P_1. The complete activity of P_0 will be described below.
For $i = 1, \ldots, n$, the X-machine P_i does the following: the first number it receives from P_{i-1} is stored as a witness value and added to the output tape. For the following numbers it receives, it checks if these are primes "from its point of view", i.e. if the witness value does not divide them; if so, the number is passed to P_{i+1} (for further checking), otherwise it is discarded. Obviously, the witness values of $P_1, \ldots, P_n$ are the first n prime numbers.
The X-machine P_{n+1} acts as follows:
- *receives a number from P_n;*
- *sends the value -1 to P_0;*
- *successively receives numbers from P_n until the received value is -1.*

We describe now the complete activity of P_0. At each step, it chooses to send a number to P_1 or to receive, if possible, a value from P_{n+1}. When receiving the

value -1 from P_{n+1}, it sends it to P_1 and stops.
The X-machines $P_1, P_2, \ldots, P_n$ will stop after receiving the value -1.
The initial form C^0 of the communication matrix C is:

$$\begin{array}{cccccc} @ & \lambda & @ & \ldots & @ & \lambda \\ \lambda & @ & \lambda & \ldots & @ & @ \\ @ & \lambda & @ & \ldots & @ & @ \\ . & . & . & . & . & . \\ @ & @ & @ & \ldots & @ & \lambda \\ \lambda & @ & @ & \ldots & \lambda & @ \end{array}$$

For the X-machine P_0, M_0 contains a variable i initialized with 2, and a boolean variable ok initalized with $false$. We have $q_0^0 = \underline{1}$ and $T_0 = \{\underline{6}\}$. The state diagram appears in Fig. 5 a, where:

f_1: ***if*** *not ok* ***then*** $out_0 \leftarrow i$

f_2: $1\,!\,out_0$

f_3: $in_0\,?\,n+1$

f_4: $i \leftarrow i+1$

f_5 : $ok \leftarrow true$; $out_0 \leftarrow in_0$

f_6 : ***if*** ok ***then***$-$

f_7: $1\,!\,out_0$

Remark 7. *The conditions "**if** $C_{01} = \lambda$" in f_2 and "**if** $C_{n+1,0} \neq \lambda$" in f_3 are implicit, so that in fact the choice between these two functions is not completely non-deterministic.*

For each $i = 1, \ldots, n$, the internal memory M_i of the X-machine P_i contains two cells x (the "witness value") and y. $T_i = \{\underline{8}\}$ and $q_i^0 = \underline{\underline{1}}$. The state diagram is shown in Fig. 5 b, where:

g_1: $i-1\,?\,in_i$

g_2: $x \leftarrow in_i$; *add x to O*

g_3: $y \leftarrow in_i$

g_4: ***if*** $y \bmod x = 0$ ***then*** $-$

g_5: ***if*** $y \bmod x \neq 0$ ***then*** $out_i \leftarrow y$

g_6: $i+1\,!\,out_i$

g_7: ***if*** $y \neq -1$ ***then*** $-$

g_8: ***if*** $y = -1$ ***then*** $-$

The internal memory of the X-machine P_{n+1} is void. We have $q_{n+1}^0 = \underline{\underline{1}}$ and $T_{n+1} = \{\underline{6}\}$. The state transition diagram appears in Fig. 5 c, where:

h_1: $n\,?\,in_{n+1}$

h_2: $out_{n+1} \leftarrow -1$

h_3: $0\,!\,out_{n+1}$ $\quad$ h_4: $n\,?\,in_{n+1}$

h_5: ***if*** $in_{n+1} \neq -1$ ***then*** $-$ $\quad$ h_6: ***if*** $in_{n+1} = -1$ ***then*** $-$

FIGURE 5. Finding the first n prime numbers. The state transition diagrams for: a) P_0 ; b) P_i, $i = 1, \ldots, n$; c) P_{n+1}

4. Conclusions

In this paper we have presented a new formal specification for systems of communicating X-machines, as an extension of the X-machine model. The input tape is replaced by the initial contents of the local memories of the components. Each X-machine has its own internal memory and two additional memory locations used for sending and receiving messages. It enables us to distinguish between ordinary and communication states. In this way internal behaviour and external behaviour can be studied separately.

It is shown that the communication between the X-machines in the system can be achieved through channels, providing a synchronous message passing, so that most of the problems that appear in concurrent programming may be modeled by $CXMS$.

We are currently working on designing an automatic method which, for any deterministic $CXMS$, generates a concurrent program written in a Pascal-FC style language (see [3]).

Further work will include verification and testing. These have to be done separately for the internal and external behaviour of the components of the system. Techniques presented in [7] have to be adapted and developed. Reachability aspects have to be studied for both behaviours too.

References

[1] Bălănescu, T., Georgescu, H., Gheorghe, M. : Stream X-Machines with Underlying Distributed Grammars, Informatica (to appear)
[2] Barnard, J., Whitworth, J., Woodward, M. : Communicating X-Machines, Journal of Information and Software Technology, Vol. 38, no. 6, 1996
[3] Burns, A., Davies, G. : Concurrent Programming, Addison Wesley, 1993
[4] Eilenberg, S. : Automata, Languages and Machine, Vol. A, Academic Press, 1974
[5] Hoare, A. : Communicating Sequential Processes, Prentice Hall, 1985
[6] Holcombe, M. : X-Machines as a Basis for Dynamic System Specification, Software Engineering Journal 3 (1988), 69 - 76
[7] Holcombe, M., Ipate, F. : Correct Systems : Building a Business Process Solution, Springer Verlag, Berlin, 1998

Faculty of Mathematics, Bucharest University, Romania
E-mail address: cri@oroles.cs.unibuc.ro