Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

PISA

EXPRESSING THE CONCURRENT BEHAVIOUR OF ABSTRACT DATA TYPES WITH MESSAGE PASSING OR SHARED MEMORY: A FORMAL COMPARISON

R. De Nicola - A. Martelli - U. Montanari

Nota Interna

B80-3

Febbraio 1980

EXPRESSING THE CONCURRENT BEHAVIOUR OF ABSTRACT DATA TYPES WITH MESSAGE PASSING OR SHARED MEMORY: A FORMAL COMPARISON

by Rocco De Nicola*, Alberto Martelli* and Ugo Montanari**

Nota interna IEI B80-3, Feb. 1980

EXTENDED ABSTRACT

* Istituto Elaborazione dell'Informazione, CNR
Via S. Maria 46
I-56100 Pisa, Italy

** Istituto di Scienze dell'Informazione
Universita' di Pisa
Corso Italia 40
I-56100 Pisa, Italy

1. Introduction

An important issue on semantics of concurrent computation and on programming language design refers to the choice of synchronization primitives. In particular, an interesting discussion has recently arosen about primitives based on message passing and about primitives based on the existence of shared memory among processes /1,2/. The two classes of primitives are shown as largely equivalent, or dual, in /1/ and it is argued that the choice should depend on efficiency considerations with respect to the underlying hardware. In the case of distributed systems, where tightly coupled processes exist on the same node and where processes on different nodes are only loosely coupled, both classes of primitives should be available on different levels /2/, to reflect the phisical structure of the system. However, in particular situations like after system reconfiguration due to a failure, or when operating in debugging mode, it might be necessary, or convenient, to execute one class of primitives on a (virtual) machine providing only the other. Thus the problem arises of "translating" between the two classes.

A related issue is the high level description of "mutable" abstract data types exhibiting concurrent behaviour. As usual one would like to define such a behaviour as abstractly as possible, and to allow the implementor to choose the most convenient sinchronization mechanism. Then a synthesis problem must be solved: Given a behaviour, determine if it is implementable using some set of primitives; if yes, build a system (a program) with the required behaviour. Here we see an instance of an abstract data type essentially as a process, and thus the whole issue of the semantics of concurrent computation is brought in. We want to emphasize that most models presently known (see /3/ for a clean and suggestive survey of some of them) are based on an a priori, particular choice of syncrhonization primitives and thus cannot be considered sufficiently abstract from the above point of view.

Another setting where a similar problem arises is in giving the semantics of languages with pointers, or in general where objects with shared parts can be built. A possible semantics describes the heap in full detail /4/. An alternate point of view may be to represent the objects with shared parts as synchronized processes, which can experience simultaneus transitions. This approach is rather convenient in representing the state of the computation of a symbolic interpreter /5,6/. A similar technique may be used to represent the relation between an object and one of its parts as selected by an access function (for instance an array and one of its elements). If no other form of sharing or aliasing is present, the effect of a state change on a part may be extended to the main object only later, when the access function returns. This technique allows to give a simple applicative semantics of a language where aliasing and sharing are forbidden, but where access functions with side effects, and their composition, are possible /7,8/.

In this paper we address only a very small portion of the above problems. Our definition of process behaviour is very simple-minded and automata_like: A (possibly infinite) set of states and a set of transitions. States are assumed as directly observable, and synchronization between different processes is evidenced by the possibility of simultaneus transitions on both processes. To make the formalism simpler, we consider the case of a universe of only two processes. However we foresee no problems in extending our results to the general case.

In the message-passing model, synchronous transitions of different processes are labeled with the same symbol of a message alphabet. Thus we consider only unbuffered message passing primitives, i.e. primitives forcing process synchronization (hand shaking or rendez-vous primitives). A process may undergo a labeled transition only if the other process simultaneously executes an equally labeled transition.

In the shared memory model, the states of both processes are obtained by making the cartesian product of a private and of a shared set of states. Thus three set of states are present, two private set of states and a shared set of states. A transition on a private set of states cause a transition of only one process, while a transition on the shared set of states cause two simultaneous transitions on both processes.

The main question we address is as follow. Given a generic behaviour, can we find a suitable pair of message-passing processes and/or of shared memory processes exhibiting this behaviour? The answer is yes for message-passing, and no for shared memory processes. In the latter case we give necessary and sufficient conditions, and then we provide a synthesis procedure. Finally we define an abstraction operation as a homomorphism between behaviours, and we prove that we can express a generic behaviour as the abstraction of the behaviour of a pair of shared memory processes. A shared memory implementation of the dining philosophers problem is given as an example.

2. The two models

In this paper we consider only the case of two communicating processes. The global behaviour B of the two processes may be defined as follows:

$B = (S_1^B, S_2^B, Q^B, q_0, T)$

where S_1^B and S_2^B are the (possibly denumerably infinite) sets of states of the two processes

$Q^B \subseteq S_1^B \mathrm{X}\ S_2^B$ is the set of global states.

$q_0 \in Q^B$ is the initial state.

$T \subseteq Q^B \mathrm{X}\ Q^B$ is the set of transitions.

Furthermore we require that

$Q^B = \{ q \mid (q_0, q) \in T^* \}$

where T^* is the reflexive and transitive closure of T; i.e. every state of Q^B must be reachable from the initial state.

Given two behaviours

$B' = (S_1^B, S_2^B, Q^{B'}, q_0, T')$ and

$B'' = (S_1^B, S_2^B, Q^{B''}, q_0, T'')$

with the same sets of states S_1^B, S_2^B and the same initial state q_0, we say that $B' \subseteq B''$ (B'' is an <u>extension</u> of B') iff $Q^{B'} \subseteq Q^{B''}$, and $T' \subseteq T''$.

2.1 The message-passing model

A pair of message-passing processes is defined as follows

$M = (S_1^M, S_2^M, s_1^o , s_2^o, A, T_1^M, T_2^M, T_L^M)$

where

S_1^M and S_2^M are the (possibly denumerably infinite) sets of states of two processes,

$s_1^o \in S_1^M$ and $s_2^o \in S_2^M$ are the initial states,

$T_i^M \subseteq S_i^M \times S_i^M$ (i = 1,2) are the private transitions of the two processes,

A is a (possibly denumerably infinite) message alphabet,

$T_L^M \subseteq S_1^M \times S_1^M \times A \cup S_2^M \times S_2^M \times A$ are the labeled transitions.

A process may execute private transitions independently from the state of the other process, whereas labeled transitions must be executed simultaneously by the two processes. Thus the behaviour B of this model is the following

$$S_i^B = S_i^M \quad (i = 1,2)$$

$$q_0 = (s_1^o , s_2^o)$$

$$Q^B = \{ q \mid (q_0, q) \in \bar{T}^* \} \text{ where}$$

$$\begin{aligned} \bar{T} = {} & \{ ((s_1,s_2),(s_1',s_2)) \mid (s_1,s_1') \in T_1^M \} \cup \\ & \{ ((s_1,s_2),(s_1,s_2')) \mid (s_2,s_2') \in T_2^M \} \cup \\ & \{ ((s_1,s_2),(s_1',s_2')) \mid \exists\, a \in A \; (s_1,s_1',a),(s_2,s_2',a) \in T_L^M \} \\ & \text{where } s_i, s_i' \in S_i^M \quad (i = 1,2) \end{aligned}$$

$$T = \{ (q, q') \mid q, q' \in Q^B \text{ and } (q, q') \in \bar{T} \}$$

2.2 <u>The shared memory model</u>

A pair of shared memory processes is defined as follows

$$S = (S_1^S, S_2^S, S_3^S, s_1^o, s_2^o, s_3^o, P_1 , P_2 , T_1^S, T_2^S, T_3^S)$$

where

S_1^S and S_2^S are the (possibly denumerably infinite) private sets of states of the two processes

S_3^S is the (possibly denumerably infinite) set of common states

$s_1^o \in S_1^S$, $s_2^o \in S_2^S$, $s_3^o \in S_3^S$ are the initial states

$T_i^S \subseteq S_i^S \times S_i^S$ $(i = 1,2,3)$ are the transitions of the three sets of states above.

The states of each process are obtained by making the cartesian product of the private and the shared set of states. However it is possible to put some constraints on these pairs of states, by prohibiting some of them. (See /9/ for a formalism using constraints to describe the concurrent behaviour of loosely coupled processes)

$P_i^S \subseteq S_i^S \times S_3^S$ $(i = 1,2)$

are the sets of allowed states of the two processes. Of course we must have $(s_i^o , s_3^o) \in P_i^S$ $(i = 1,2)$.

The behaviour B of this model is obtained as follows

$S_i^B = P_i^S$ $(i = 1,2)$

The set of possible states $\overline{Q}^B$ consists of all pairs of states which have the same shared component

$\overline{Q}^B = \{ ((s_1, s_3), (s_2, s_3)) \mid (s_i , s_3) \in P_i^S, (i = 1,2) \}$, and all possible transitions are

$$\bar{T} \subseteq \bar{Q}^B \times \bar{Q}^B$$

$$\bar{T} = \{ (((s_1, s_3), (s_2, s_3)), ((s'_1, s'_3), (s'_2, s'_3))) \mid$$
$$(s_1, s'_1) \in T^S_1, s_2 = s'_2, s_3 = s'_3 \text{ or}$$
$$s_1 = s'_1, (s_2, s'_2) \in T^S_2, s_3 = s'_3 \text{ or}$$
$$s_1 = s'_1, s_2 = s'_2, (s_3, s'_3) \in T^S_3\}$$

Finally

$$q_0 = ((s^0_1, s^0_3), (s^0_2, s^0_3))$$
$$Q^B = \{ q \mid (q_0, q) \in \bar{T}^* \}$$
$$T = \{ (q, q') \mid q, q' \in Q^B \text{and } (q, q') \in \bar{T} \}$$

3. Comparison of the two models

In this section we show that the message-passing model is more general than the shared memory model. In fact we can prove that it is always possible to find a suitable pair of message-passing processes having a given behaviour, whereas this is not true for the shared memory model.

<u>Theorem 3.1</u> Given a global behaviour B of two processes, it is always possible to find a pair of message_passing processes having this behaviour.

<u>Proof outline</u> Let

$B = (S_1^B, S_2^B, Q^B, q_o, T)$

then we can construct a pair of message passing processes

$M = (S_1^M, S_2^M, s_1^o, s_2^o, A, T_1^M, T_2^M, T_L^M)$

where

$S_i^M = S_i^B, \ (i = 1,2)$

$s_i^o = q_o^i, \ (i = 1,2)$ where $q_o = (q_o^1, q_o^2)$

A is any (possibly infinite) alphabet

$T_i^M = \emptyset \quad (i = 1,2)$

T_L^M is obtained from T by associating with every transition $((s_1, s_2), (s_1', s_2'))$ of T two transitions (s_1, s_1', a) and (s_2, s_2', a) of T_L^M, with a different label for every pair of transitions of T_L^M.

In order to describe how to obtain a pair of shared memory processes with a given behaviour, we need some definitions.

Let

$B = (S_1^B, S_2^B, Q^B, q_o, T)$

be a global behaviour. We define a relation

$R_1 \subseteq S_1^B \times S_1^B$

such that

$(s_1, s_1') \in R_1$ iff $\exists s_2, s_2' \in S_2^B \ ((s_1, s_2), (s_1', s_2')) \in T$

and $s_2 \neq s_2'$

A relation $R_2 \subseteq S_2^B \times S_2^B$ can be defined in a similar way.

We can prove

Lemma 3.1 Let $(Q^{B})^{*S}$, R_1^{*S} and $R_2^{*S} \subseteq (S_1^B \cup S_2^B)^2$ be the reflexive, symmetric and transitive closures of the relations Q^B, R_1 and R_2 defined above. A necessary condition for the existence of a pair of shared memory processes with a given behaviour B is that

(3.1) $(Q^B)^{*S} \cap (R_1^{*S} \cup R_2^{*S}) = I$

where I is the identity relation.

Proof outline States of the behaviour which are in the relation $(Q^B)^{*S}$ correspond to states of the model with the same shared component, while states in the relation $(R_1^{*S} \cup R_2^{*S})$ correspond to states with the same private component. Thus condition (3.1) means that two states with the same shared component and the same private component are identical.

If a behaviour $B = (S_1^B, S_2^B, Q^B, q_o, T)$ satisfies condition (3.1) we can construct a pair of shared memory processes

$S = (S_1^S, S_2^S, S_3^S, s_1^o, s_2^o, s_3^o, P_1, P_2, T_1^S, T_2^S, T_3^S)$

whose behaviour $\bar{B} = (S_1^B, S_2^B, \bar{Q}^B, q_o, \bar{T})$ is an extension of B.

Outline of the construction The states in S_1^S and S_2^S correspond to the equivalence classes of $R_1^{*S} \cup R_2^{*S}$ in S_1^B and

S_2^B respectively, while the states in S_3^S correspond to the equivalence classes of $(Q^B)^{*S}$. In P_1 and P_2 we put all pairs, whose equivalence classes have nonempty intersection. Since the intersection contains at most one element (due to condition (3.1)), every pair in P_1 (P_2) corresponds to exactly one state s_1^B (s_2^B) of the behaviour B. Thus the sets of states $\bar{S}_1^B = P_1$ and $\bar{S}_2^B = P_2$ in the behaviour $\bar{B}$ of S can be identified with the sets S_1^B and S_2^B in the given behaviour B. Furthermore, we put a pair $(s_1^S, s_1'^S)$ in T_1^S iff there exists a transition $((s_1^B, s_2^B), (s_1'^B, s_2^B))$ in T with $s_1^B = (s_1^S, s_3^S)$ and $s_1'^B = (s_1'^S, s_3^S)$ for some s_2^B and s_3^S. Similarly for T_2^S. We put a pair $(s_3^S, s_3'^S)$ in T_3^S iff there exists a transition $((s_1^B, s_2^B), (s_1'^B, s_2'^B))$ in T with $s_1^B = (s_1^S, s_3^S)$, $s_1'^B = (s_1^S, s_3'^S)$, $s_2^B = (s_2^S, s_3^S)$ and $s_2'^B = (s_2^S, s_3'^S)$ for some s_1^S and s_2^S. It is easy to see that we have

$$Q^B \subseteq \bar{Q}^B \quad \text{and} \quad T \subseteq \bar{T}$$

by construction.

The pair of shared memory processes obtained with the construction above is in a sense <u>minimal</u> as specified by the following theorem.

<u>Theorem 3.2</u>

Given a behaviour B satisfying condition (3.1), let $\bar{B}$ be the behaviour of the pair of shared memory processes constructed as above. There exists no pair of shared memory processes with a behaviour $\bar{B}'$ such that

$$B \subseteq \bar{B}' \subset \bar{B}$$

Proof outline If a pair with behaviour $\bar{B}'$ exists, its sets of states $S_1^{\prime S}$, $S_2^{\prime S}$ and $S_3^{\prime S}$ must be obtained with the same construction above. To have $\bar{B}' \subset \bar{B}$, we must have $\bar{Q}'^{\beta} \subset \bar{Q}^{\beta}$ or $T' \subset \bar{T}$, but if we remove any element from $\bar{Q}^{\beta}$ or $\bar{T}$ we obtain a pair whose behaviour is not an extension of B.

We can now express our main result as a corollary.

Corollary 3.1

Given a behaviour B satisfying condition (3.1), let $\bar{B}$ be the behaviour of the pair of shared memory processes constructed as above. If $\bar{B}$ = B then this construction is a synthesis procedure. If $\bar{B} \neq B$ then behaviour B cannot be implemented with a pair of shared memory processes.

Proof Obvious.

Finally we can show that no condition is required if we allow abstraction, i.e. if we allow to map many states of the shared memory processes into the same state of the behaviour.

Theorem 3.3 Given a global behaviour B, it is always possible to find a pair of shared memory processes having it as abstract behaviour.

Proof outline

The private parts of the two processes are a singleton:

$S_1^S = \{ s_1^S \}$, $S_2^S = \{ s_2^S \}$,

whereas

$S_3^S = Q^B$, $T_3^S = T$.

All pairs of states of the two processes are allowed. The abstraction is defined by mapping every state (s_1^S, s_3^S), where $s_3^S = (s_1^B, s_2^B)$, of the first process into s_1^B , and every state (s_2^S, s_3^S) of the second process into s_2^B.

Note that, although the two processes of the previous proof have, in general, many more states than in the abstract behaviour, the two processes have as many global states as the abstract behaviour.

We point out that the pair of shared memory processes with abstract behaviour B described in the proof of Theorem 3.3, might be considered as dual of the pair of message passing processes with the same behaviour described in the proof of Theorem 3.1. In fact, both pairs of processes have the same global states and global transitions, and in both cases the processes have no private transitions.
Finally, it would be possible to investigate under which conditions we might derive from a given behaviour a pair of message passing processes with private transitions, or more than one transition labelled with the same symbol. Dually,

we might want to derive from a given behaviour a pair of shared memory processes with more than one private state.

4. An example: the dining philosophers

In this section we apply informally the ideas of the previous two sections to an example with more than two processes. There are six processes: the philosophers Ph^0, Ph^1, Ph^2, and the forks F^0, F^1, F^2, whose global behaviour is described as follows.

Every philosopher Ph^i has four states: t_i , l_i, r_i and e_i, meaning, respectively that the i-th philosopher is thinking, that he has picked up his left fork F^i, that he has picked up his right fork F^{i+1}, and that he has picked up both F^i and F^{i+1} (Of course the + operation is modulo 3). Every fork F^i has two states: d_i and u_i , meaning respectively, that the fork is down, and it has been picked up by an adjacent philosopher.

The initial global state is $< d_0 t_0 d_1 t_1 d_2 t_2 >$. The other states of Q^8 are all states reachable from the initial state by applying the following transitions

$$\ldots d_i\, t_i \ldots \longrightarrow \ldots u_i\, l_i \ldots$$

$$\ldots t_i\, d_{i+1} \ldots \longrightarrow \ldots r_i\, u_{i+1} \ldots$$

$$\ldots l_i\, d_{i+1} \ldots \longrightarrow \ldots e_i\, u_{i+1} \ldots$$

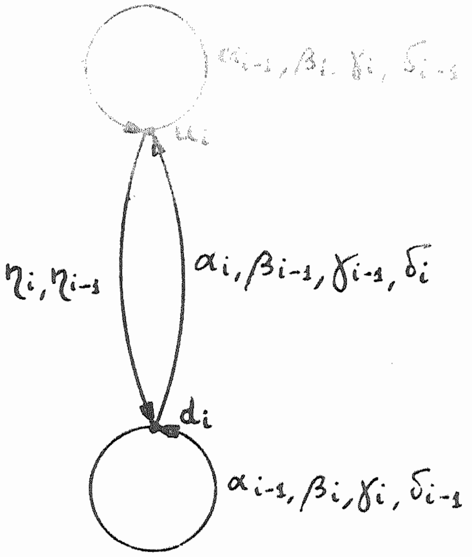
$$\ldots d_i\, r_i \ldots \dashrightarrow \ldots u_i\, e_i \ldots$$

$$\ldots u_i\, e_i\, u_{i+1} \ldots \dashrightarrow \ldots d_i\, t_i\, d_{i+1} \ldots$$

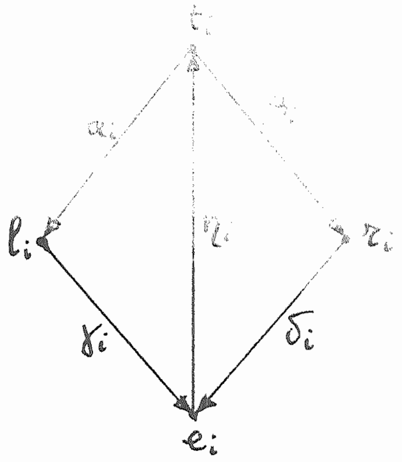
These transitions mean that a philosopher can pick up the two forks one at a time in any order, and when he has finished eating, he puts down both forks simultaneously.

Note that there are states such as $< u_0 l_0 u_1 l_1 u_2 l_2 >$ from which there is no outgoing transition, thus corresponding to a deadlock situation.

This global behaviour can be easily represented within the message passing model. We need three alphabets A^0, A^1, A^2, where alphabet A^i is common among the philosopher Ph^i and his two adjacent forks. Thus a philosopher will always have a simultaneus transition with his adjacent forks. The transitions for the i-th fork and philosopher are



$\alpha_{i-1}, \beta_i, \gamma_i, \delta_{i-1}$
u_i
η_i, η_{i-1}
$\alpha_i, \beta_{i-1}, \gamma_{i-1}, \delta_i$
d_i
$\alpha_{i-1}, \beta_i, \gamma_i, \delta_{i-1}$

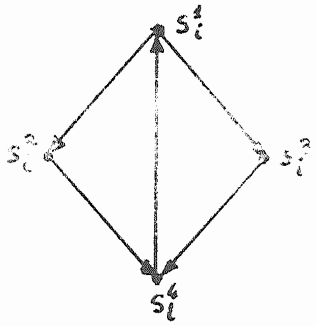


where $\alpha_i, \beta_i, \gamma_i, \delta_i, \eta_i \in A^i$ (i = 0,1,2), and a transition of F^i with multiple labels such as $\alpha_i, \beta_{i-1}, \ldots$ means that this transition can be executed if simultaneously the processes Ph^i and F^{i+1}perform a transition labeled with α_i, or F^{i-1} and Ph^{i-1} perform a transition labeled with β_{i-1}, and so on.

Note that, in extending to more than two processes the message passing model defined in the previous sections, we allow synchronization of several processes: in fact, a process may undergo a transition labeled with a symbol of an alphabet A only if simultaneously all other processes which share alphabet A with it undergo an equally labeled transition. In this example every transition involves three processes, however some transitions might involve only pairs of processes. For instance a philosopher might exchange a message only with his left fork in order to perform the transition t_i --> l_i or r_i --> e_i, since these transitions do not depend on the state of the right fork.

Going on to the shared memory model, we might show that the global behaviour does not satisfy some conditions similar to those given in the previous section, and thus no shared memory processes exist with this behaviour. Intuitively, since a fork has only two states, every transitions should always influence both adjacent philosophers.

It is possible however to define the shared memory processes in such a way that they have as abstract behaviour the behaviour given above. Every philosopher Ph^i shares with his two adjacent forks a process S^i with the following state transition diagram



A philosopher has no private part, and thus his abstract states are the states of his shared part S^i with the obvious mapping

s_i^1 --> t_i, s_i^2 --> l_i, s_i^3 --> r_i, s_i^4 --> e_i.

A fork F^i has two shared parts S^{i-1} and S^i (i.e. the two shared processes associated with the two adjacent philosophers) and no private part. The pairs of states of S^{i-1} and S^i are mapped into the abstract states as follows

(s_{i-1}^1 , s_i^1) --> d_i, (s_{i-1}^1 , s_i^2) --> u_i,
(s_{i-1}^1 , s_i^3) --> d_i, (s_{i-1}^1, s_i^4) --> u_i,
(s_{i-1}^2, s_i^1) --> d_i, (s_{i-1}^2, s_i^2) --> u_i,
(s_{i-1}^2 , s_i^3) --> d_i, (s_{i-1}^2, s_i^4) --> u_i,
(s_{i-1}^3 , s_i^1) --> u_i, (s_{i-1}^3 , s_i^3) --> u_i,
(s_{i-1}^4 , s_i^1) --> u_i, (s_{i-1}^4 , s_i^3) --> u_i

Missing pairs of states are not allowed.

Note that the global states and transitions of these processes with shared memory are isomorphic with the global states and transitions of the required behaviour. The same is true, of course, for the message passing processes defined above, and thus it is easy to put labels of the latter in corrispondence with shared transitions of the former. For instance, label α_i corresponds to transition (s_i^1, s_i^2), label β_i to (s_i^4, s_i^3), and so on.

References

1. Lauer H.C. and Needham, R.M., On the Duality of Operating System Structure, in Proc. Second International Symposium on Operating System, IRIA, Oct. 1978, reprinted in Operating System Review, 13,2 April 1979, pp. 3-19.

2. Svobodova, L., Liskov, B. and Clark, D., Distibuted Computer Systems: Structure and Semantics, Internal Report TR-215, Laboratory For Computer Science, M.I.T., March 1979.

3. MacQueen, D.B., Models for Distributed Computing, Rapport de Recherche N. 351, laboratoire de recherche en informatique et automatique, IRIA, April 1979.

4. Spitzen, J. and Wegbreit, B., The Verification and Synthesis of Data Structures, Acta Informatica, vol 4, pp. 127-144, 1975.

5. Cheatham, T.E., Townley,J.A. and Holloway, G.H., A System for Program Refinement, Proc. of 4th International Conference on Software Engineering, Munich, pp. 53-62, Sept. 1979.

6. Asirelli, P., Degano, P., Levi, G., Martelli, A., Montanari, U., Pacini, G., Sirovich, F., Turini, F., A Flexible Environment for Program Development Based on a Symbolic Interpreter, Proc. of 4th International Conference on Software Engineering, Munich, pp. 251-263, Sept. 1979.

7. Asirelli, P., Martelli, A. and Montanari, U., Language Constructs for Controlling Side Effects: A Proposal, Internal Report OL-78-3, IEI, Pisa, December 1978, submitted for publication to Transaction on Software Engineering.

8. De Nicola, R., Tipi di Dati Astratti Modificabili: Definizione e loro impiego in linguaggi di progammazione con effetti laterali controllati, Tesi di Laurea, Istituto Scienze dell'Informazione, Pisa, December 1978, in Italian.

9. Maggiolo Schettini, A., Wedde, H. and Winkowski, J., Modeling a Solution for a Control Problem in Distributed System by Restrictions, Proc. Symposium on Semantics of Concurrent Computation, Evian, France, pp 226-248, July 1979, Lecture Notes in Computer Science 70, Springer Verlag 1979.