



The United Nations  
University

**UNU/IIST**

International Institute for  
Software Technology

---

# An Algorithm for Maintaining Consistent View of Processes in Distributed Systems

---

Dang Van Hung

January 1994

## UNU/IIST

UNU/IIST enables developing countries to attain self-reliance in software technology by: (i) their own development of high integrity computing systems, (ii) highest level post-graduate university teaching, (iii) international level research, and, through the above, (iv) use of as sophisticated software as reasonable.

UNU/IIST contributes through: (a) advanced, joint industry-university advanced development projects in which rigorous techniques supported by semantics-based tools are applied in case studies to large scale software developments, (b) own and joint university and academy institute research in which new techniques for application domain and computing platform modelling, requirements capture, software engineering and programming are being investigated, (c) advanced, post-graduate and post-doctoral level courses which typically teach Design Calculi oriented software development techniques, (d) events [panels, task forces, workshops and symposia], and (e) dissemination.

Application-wise, the advanced development projects presently focus on software to support large-scale infrastructure systems such as transport systems (railways, airlines, air traffic, etc.), manufacturing industries, telecommunications, etc., and are thus aligned with UN and International Aid System concerns. UNU/IIST is a leading research centre in the area of Duration Calculi, i.e. techniques applicable to *real-time, reactive, hybrid & safety critical systems*. The research projects parallel and support the advanced development projects.

At present, the technical focus of UNU/IIST in all of the above is on applying, teaching, researching, and disseminating Design Calculi oriented techniques and tools for trustworthy software development. UNU/IIST currently emphasises techniques that permit proper development steps and interfaces. UNU/IIST also endeavours to promulgate sound project and product management principles.

UNU/IIST's primary dissemination strategy is to act as a clearing house for reports from research and technology centres in industrial countries to industries and academic institutions in developing countries. At present more than 200 institutions worldwide contribute to UNU/IIST's report collection while UNU/IIST at the same time subscribes to more than 125 international scientific and technical journals. Information on reports received (and produced) and on journal articles is to be disseminated regularly to developing country centres — which are then free to order a reasonable number of report and article copies from UNU/IIST.

Dines Bjørner, Director — 1.7.1992–31.6.1997

UNU/IIST Reports are either *R*esearch, *T*echnical, *C*ompendia or *A*dministrative reports:

$\mathcal{R}$  Research Report •  $\mathcal{T}$  Technical Report •  $\mathcal{C}$  Compendium •  $\mathcal{A}$  Administrative Report



The United Nations  
University

**UNU/IIST**

**International Institute for  
Software Technology**

P.O. Box 3058  
Macau

---

# An Algorithm for Maintaining Consistent View of Processes in Distributed Systems

---

**Dang Van Hung**

## **Abstract**

In the paper, the problem of determining the global properties of distributed systems is addressed. At each moment during the execution of a system, every process has its knowledge about the system. By message passing the processes can exchange their knowledge. We present a general algorithm for a process to synthesize the knowledge that it obtains, and to maintain its consistent view about the system. Depending on different interpretations the algorithm can be used for distributed snapshots, for verification and design of stabilizing protocols, etc.

Dang Van Hung is from the Institute of information Technology of National Center for Natural Science and Technology of Vietnam, where he is a researcher. He is a Fellow of UNU/IIST from April 1994 to July 1995. His research interest is in Formal Technique of Programming, Concurrent and Distributed systems. E-mail: [dvh@iist.unu.edu](mailto:dvh@iist.unu.edu).

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Labeled Transition System Model of Distributed Computations</b>	<b>1</b>
<b>3</b>	<b>General algorithm for maintaining consistent view of processes in distributed systems</b>	<b>7</b>
<b>4</b>	<b>Conclusion</b>	<b>11</b>



## 1 Introduction

A distributed system is a set of processors that share no memory and communicate only by passing message along bidirectional communication links. Because the relative processor speeds and message transmission time are not known, coordination between processors is difficult, and the design and proof of distributed algorithms are very complicated. Coordination between processors is based only on the causality between events in the system. Thus, ones have to use additional messages called control messages in order to observe a global state property, such as in snapshot algorithms.

Message passing is the only way for transferring knowledge in distributed systems. At each moment a process in a distributed system has a view about the system that is changing during the execution. When a process sends a message to another process, its knowledge is transferred to this process explicitly or implicitly. The recipient, when it has received the message can combine the obtained knowledge with its to get a new view about the system. Maintaining the knowledge about the system of each process in its local states may simplify determining global state properties of the system. By considering the interaction between knowledge and actions in distributed systems and the way that consistent global view of a process is changing during the execution of the systems, we propose a general algorithm for maintaining the amount of knowledge about the systems in the local states of processes. We found that many familiar algorithms, such as checkpointing, concurrent common knowledge attainment, partial logical time, etc. can be derived from our algorithm.

The paper is organized as follows. Section 2 contains our system model. In Section 3 we present our algorithm and some of its applications.

## 2 Labeled Transition System Model of Distributed Computations

The distributed computing systems considered in this paper are taken from [1, 6, 10, 11] as follows. A distributed computing system consists of multiple autonomous processors that do not share primary memory but cooperate by sending messages over a communication network. The communication network is assumed to be connected so that each processor can send message to any other, and to be reliable (i.e. messages are delivered error-free without ever being lost). Logically, between two ordered processors there exists an one way communication channel. We consider in this paper two kinds of message passing: synchronous and asynchronous one. With synchronous message passing, the sender of a message is blocked until the receiver has received the message. With asynchronous message passing, the sender does not wait for the receiver to be ready to accept its message. Conceptually, the sender continues immediately after sending the message. We start with the definition of labeled transition systems which is taken from [9].

**Definition 1:** A labeled transition system is a triple  $P = (Q, \rightarrow, A)$ , where  $Q$  is the set of configurations,  $A$  is the set of labels, and  $\rightarrow \subseteq Q \times A \times Q$  is the transition relation.

In the sequel, we write  $p \xrightarrow{a} q$  instead of  $(p, a, q) \in \rightarrow$ . The labeled transition  $P$  is said to be deterministic iff  $\forall p \in Q, a \in A \bullet |\{q \in Q | p \xrightarrow{a} q\}| \leq 1$ .

**Definition 2:** A distributed computing system  $D$  is a tuple  $(P_1, \dots, P_N, C, M)$ , where

1.  $P_i = (Q_i, \rightarrow_i, A_i)$  is a labeled transition system, called the process  $i$  of the system,  $Q_i$  and  $A_i$  are respectively the set of its local states and the set of its local actions,
2.  $C$  is the set of communication actions of the system,  $C = \{s_{ij}, r_{ij} | i, j = 1, \dots, N, i \neq j\}$ , where  $r_{ij}, s_{ij}$  are respectively the message receipt, the message send of the process  $i$  from/to the process  $j$ ,
3.  $M$  is a matrix the element  $M_{ij}$  of which is the set of messages sent by  $P_i$  to  $P_j$ ,
4.  $\rightarrow$  is a transition labeled over  $C$ ,  $\xrightarrow{c} \subseteq Q_i \times (M_{ij} \times Q_i)$  if  $c = s_{ij}$ , and  $\xrightarrow{c} \subseteq (Q_i \times M_{ji}) \times Q_i$  if  $c = r_{ij}$ ,  $i, j = 1, \dots, N, i \neq j$ .

For simplicity, we assume that the set  $C, A_i, Q_i, M_{ij}, i, j = 1, \dots, N, i \neq j$  are pairwise disjoint. Thus, the labeled transition relations in Definition 2 can be distinguished by their label sets. Consequently, we can omit the index of the relations without fear of confusion.

In order to represent the parallel execution of the processes in a distributed system, we use joint actions which has been introduced in [7, 8]. A joint action consists of actions performed simultaneously by all processes in the system including local actions, communication actions and the idle action.

**Definition 3:** Let  $D$  be a distributed system given as above. The set  $\Sigma_i = A_i \cup \{\lambda\} \cup \{r_{ij}, s_{ij} | j = 1, \dots, N\}$  is called set of actions performed by the process  $i$ , where  $\lambda$  denotes the empty word, the idle action. The set  $A = \Sigma_1 \times \dots \times \Sigma_N$  is called the set of joint actions of the system, while the set  $AS = \{(a_1, \dots, a_N) | a_i \in A_i \text{ and } a_i = s_{ij} \text{ iff } a_j = r_{ji}, i, j = 1, \dots, N\}$  is called the set of its synchronous actions.

Let  $VAS(A) = \{(h_1, \dots, h_N) | h_i \in \Sigma_i^*\}$ .  $VAS(A)$  is called the set of vectors of action sequences over  $A$ . In this paper we adopt the following conventions.  $\Lambda$  will denote the joint action for which all components are  $\lambda$ .  $\Lambda_i(a)$  denotes the joint action for which all components are  $\lambda$  except that the  $i$ th component is  $a$ . For  $W \in VAS(A)$ ,  $w_i$  will denote its  $i$ th component. For  $W, W' \in VAS(A)$ , the product  $WW'$  of  $W$  and  $W'$  is defined as  $WW' = (w_1 w'_1, \dots, w_N w'_N)$ . For a sequence  $w = a_1 a_2 \dots a_m$  over an alphabet  $B$ , we call  $m$  the length of  $w$  (denoted by  $|w|$ ), and the set  $\{(i, a_i) | i = 1, \dots, m\}$  set of letter occurrences of  $w$ . For a letter occurrence  $(i, a)$ ,  $a$  is said to be its label. A singleton set will be identified with its element. For a partial ordering  $(O, <)$ ,  $O' \subseteq O$  is called a left closed subset of  $O$  iff  $\forall x \in O' \forall y \in O \bullet y < x \Rightarrow y \in O'$ .

The computations of a distributed system are described by associating a labeled transition system is associated with it. A state of the system consists of local states of its processes, one for each, and a state of communication channels. A state of the latter will be the sequences of messages that have not been accepted.

**Definition 4:** Let  $D$  be a distributed system given as above. The state space of  $D$  is the set  $S = Q_1 \times \dots \times Q_N \times \Gamma$ , where  $\Gamma$  is the set of matrices  $G$  the element  $g_{ij} (\in M_{ij}^*)$  of which is a priori-queue of possible messages from the process  $i$  to the process  $j$ . Each element of  $S$  will be called a complete state of the system; the vector of the  $N$  first components of a complete state is simply called state of the system.

Unless otherwise stated, states of a channel are not assumed to be FIFO. The empty queue represents the state that there is no message on the channel and is denoted by  $\lambda$  also. For a queue  $g$ , the operation  $put(g, m)$  gives a new queue which is the queue  $g$  added with the element  $m$ , while the operation  $get(g)$  returns a element  $m$  of  $g$  and a new queue which is  $g$  with  $m$  being removed if  $g$  is not empty. In the case that  $g$  is a FIFO queue,  $put(g, m)$  means putting  $m$  at the front of  $g$  and  $get(g)$  means getting and removing the element at the gear of  $g$ .

**Definition 5:** The labeled transition system for  $D$  is  $T = (S, \rightarrow, A)$ , where  $S$  is the state space,  $A$  is the set of joint actions of  $D$ ,  $\rightarrow$  is defined below.

For  $\bar{q} = (q_1, \dots, q_N, G)$ ,  $\bar{q}' = (q'_1, \dots, q'_N, G')$  in  $S$ ,  $\bar{a} = (a_1, \dots, a_N)$  in  $A$ ,  $\bar{q} \xrightarrow{\bar{a}} \bar{q}'$  iff for  $i, j = 1, \dots, N$ :

1.  $a_i \in A_i \cup \{\lambda\}$ ,  $a_j \in A_j \cup \{\lambda\}$  and  $q_i \xrightarrow{a_i} q'_i$ ,  $q_j \xrightarrow{a_j} q'_j$ ,  $g'_{ij} = g_{ij}$ , or
2.  $a_i \in A_i \cup \{\lambda\}$ ,  $a_j = r_{ji}$  and  $q_i \xrightarrow{a_i} q'_i$ ,  $(q_j, m) \xrightarrow{a_j} q'_j$ ,  $(m, g'_{ij}) = get(g_{ij})$ , or
3.  $a_i \in A_i \cup \{\lambda\}$ ,  $a_j = s_{ji}$  and  $q_i \xrightarrow{a_i} q'_i$ ,  $q_j \xrightarrow{a_j} (m, q'_j)$ ,  $g'_{ji} = put(m, g_{ji})$ , or
4.  $a_j = r_{ji}$ ,  $a_j = s_{ji}$  and  $q_j \xrightarrow{a_j} (m, q'_j)$ ,  $(q_i, m') \xrightarrow{a_i} q'_i$ ,  $(m', g'_{ji}) = get(put(m, g_{ji}))$ .

Each process in a distributed system can perform its local computations independently of the others. The message passing is synchronous if the specified channel is empty and the receiver is ready to accept message.

The relation  $\rightarrow$  is extended to be labeled over  $VAS(A)$  as follows. For  $\bar{q}, \bar{q}' \in S$ ,  $W \in VAS(A)$ ,  $\bar{q} \xrightarrow{W} \bar{q}'$  iff there exists  $\bar{a}_1, \dots, \bar{a}_k \in A$ ,  $\bar{q}_1, \dots, \bar{q}_k \in S$  such that  $W = \bar{a}_1 \bar{a}_2 \dots \bar{a}_k$  and  $\bar{q} = \bar{q}_1 \xrightarrow{\bar{a}_1} \bar{q}_2 \xrightarrow{\bar{a}_2} \dots \xrightarrow{\bar{a}_k} \bar{q}_k = \bar{q}'$ . Furthermore, we call  $W_n = \bar{a}_1 \bar{a}_2 \dots \bar{a}_n$ ,  $n \leq k$  a prefix of  $W$  and denote by  $W_n \sqsubseteq W$ .

Let  $\bar{q}^0 = (q_1^0, \dots, q_N^0, \emptyset)$  be a fixed complete state, called initial state of  $D$  in which  $\emptyset$  denotes the matrix having the empty queue as its elements.

**Definition 6:** Let  $H = \{W | W \in VAS(A) \wedge \exists \bar{q} \in S \bullet \bar{q}^0 \xrightarrow{W} \bar{q}\}$ ,  $R = \{\bar{q} | \exists W \in VAS(A) \bullet \bar{q}^0 \xrightarrow{W} \bar{q}\}$ .  $H$  and  $R$  respectively are called the set of histories and reachable states of  $D$ .

The causality relation between action occurrences in the local computation and local states in an execution represented by  $W = \bar{a}_1 \bar{a}_2 \dots \bar{a}_k$  of  $D$  can be seen through the construction of an occurrence net from  $W$ . An occurrence net is a triple  $(B, E, F)$ , where  $B$  is the set of conditions,  $E$  is the set of events and  $F \subseteq (B \times E) \cup (E \times B)$  is the flow relation such that  $F^+$  is acyclic (hence,  $F^+$  is a partial order), and such that  $\forall b \in B \bullet (|\{e | (e, b) \in F\}| \leq 1 \wedge |\{e | (b, e) \in F\}| \leq 1)$  and  $E \subseteq \text{dom}(F) \cap \text{cod}(F)$ . Graphically, elements of  $B$  are represented by circles, the elements of  $E$  are represented by bars, an element  $(a, b)$  of  $F$  is represented by a directed arc from  $a$  to  $b$ . The readers are referred to [2] for the details of the notions of occurrence nets. An occurrence net  $\mathcal{N}$  is associated to  $W$  as follows. Each action occurrence of a process corresponds to an event of  $\mathcal{N}$  labeled by its name. Each local state of a process occurred during the execution corresponds to a condition of  $\mathcal{N}$  labeled by the state. Each message  $m$  sent by the process  $i$  to the process  $j$  corresponds to a condition labeled by  $(m, i, j)$ .  $\mathcal{N}$  is such that if  $\bar{q}^0 \xrightarrow{W} \bar{q} = (q_1, \dots, q_N, G)$ , then for  $i = 1, \dots, N$ ,  $q_i$  is a label of a condition having no outgoing arc.  $\mathcal{N}$  is defined by induction on the length of a sequence of joint actions of  $W$  as follows. The occurrence net associated to  $\Lambda$  is  $(B, \emptyset, \emptyset)$ , where  $B$  contains for each process  $i$  a condition labeled by  $q_i^0$ .

If  $(B, E, F)$  is an occurrence net associated to  $W \in H$ , and if  $\bar{a} = (a_1, \dots, a_N)$  is a joint action such that  $\bar{q}^0 \xrightarrow{W} \bar{q} = (q_1, \dots, q_N, G) \xrightarrow{\bar{a}} \bar{q}' = (q'_1, \dots, q'_N, G')$ , then an occurrence net associated to  $W\bar{a}$  is constructed from  $(B, E, F)$  as follows. For each  $a_i, i = 1, \dots, N$  such that  $a_i = s_{ij}$  with some  $j$ , and the transition  $q_i \xrightarrow{a_i} (m, q'_i)$  is used in the definition of the transition  $\bar{q} \xrightarrow{\bar{a}} \bar{q}'$ , a new event  $e$  labeled by  $s_{ij}$  is added to  $E$ . Let  $b$ , labeled by  $q_i \in B$  such that  $b$  has no outgoing arc. An element  $(b, e)$  is added to  $F$ . Then, we add new conditions  $b'$  and  $b''$  labeled by  $q'_i$  and  $(m, i, j)$ , respectively, to  $B$  and new elements  $(e, b'), (e, b'')$  to  $F$ . For each  $a_i$  such that  $a_i \in A_i \cup \{\lambda\}$ , a new event  $e$  labeled by  $a_i$  is added to  $E$ . Let  $b \in B$  be labeled by  $q_i$  and have no outgoing arc. An element  $(b, e)$  is added to  $F$ . Then, we add a new condition  $b'$  labeled by  $q'_i$  to  $B$  and a new arc  $(e, b')$  to  $F$ . For each  $a_i$  such that  $a_i = r_{ij}$  with some  $j$ , and the transition  $(q_i, m) \xrightarrow{a_i} q'_i$  is used in the definition of the transition  $\bar{q} \xrightarrow{\bar{a}} \bar{q}'$ , a new event  $e$  labeled by  $a_i$  is added to  $E$ . Let  $b, b' \in B$  be such that they have no outgoing arc, and such that  $b$  is labeled by  $q_i$  and  $b'$  is labeled by  $(m, j, i)$ . New arcs  $(b, e), (b', e)$  are added to  $F$ . Then, we add a new condition  $b''$  labeled by  $q'_i$  to  $B$  and a new arc  $(e, b'')$  to  $F$ . The correctness of the construction is obvious.

**Example:** Let  $D = (P_1, P_2, C, M)$ ,  $P_1 = (\{q_1, q_2, q_0\}, \rightarrow, \{b\})$ ,  $P_2 = (\{p_1, p_2, p_0\}, \rightarrow, \{a\})$ ,  $M_{21} = \{1, 0\}$ , where  $(q_0, 1) \xrightarrow{r_{12}} q_1$ ,  $(q_0, 0) \xrightarrow{r_{12}} q_2$ ,  $q_1 \xrightarrow{a} q_0$ ,  $p_0 \xrightarrow{s_{21}} (p_2, 0)$ ,  $p_1 \xrightarrow{s_{21}} (p_0, 1)$ ,  $p_0 \xrightarrow{b} p_1$ . Let  $W = (r_{12} a r_{12}, b s_{21}, s_{21}) = (\lambda, b)(r_{12}, s_{21})(a, b)(r_{12}, s_{21})(\lambda, s_{21})$ ,  $\bar{q}^0 = (q_0, p_0, \emptyset)$ . The occurrence net defined for  $W$  is pictorial in Fig. 1.

Unlike in the models in the literature [6, 10, 11], in our model the role of messages in the transition of states of the system is taken into account. Graphically, we use occurrence nets, in which both actions and states are represented. The relation  $F^+$ , when restricted on the set of events, is exactly the ‘‘happens before’’ relation defined by Lamport [3].

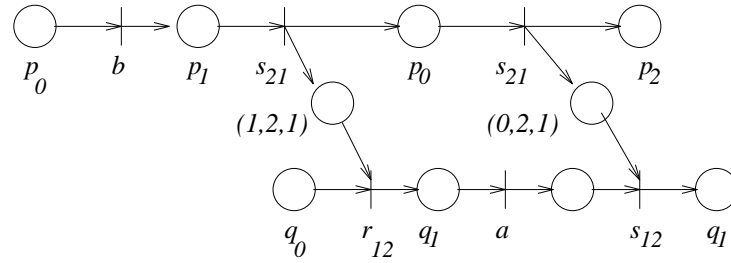


Fig. 1

**Lemma 1:** Let  $W \in H$ ,  $\mathcal{M}$  be an occurrence net associated to  $W$ . There is a one-to-one corresponding  $\varphi$  from the set of letter occurrences of the components of  $W$  to the set  $E$  of events of  $\mathcal{M}$  preserving labels such that:

1. for letter occurrences  $a, a'$  in  $w_i$ ,  $a$  follows immediately  $a'$  in  $w_i$  if and only if there exists a condition  $b$  with label in  $Q_i$  such that  $(\varphi(a'), b), (b, \varphi(a)) \in F$ ,
2. for a letter occurrence  $a$  with the name  $r_{ij}$  in  $w_i$ , there exists only one letter occurrence  $a'$  with the name  $s_{ji}$  in  $w_j$  and a condition  $b \in B$  with label  $(m, j, i)$  for some  $m \in M_{ji}$  such that  $(\varphi(a'), b), (b, \varphi(a)) \in F$ .

We call  $a'$  defined in (2) the corresponding send of  $a$ , and in both cases (1) and (2) we say that  $\varphi(a')$  happened immediately before  $\varphi(a)$ .

In general, the transition system of a distributed system need not be deterministic although the transition systems of its components are deterministic. The reason is that we do not specify the way that messages are delivered. Thus, the correspondence between sends and receipts is not unique. However, we have the following theorem which follows immediately from Lemma 1.

**Theorem 1:** Let  $D$  be a distributed system in which the states of the channels are FIFO queues of possible messages. If the transition relations of the components of  $D$  are deterministic then so is the transition system for  $D$ .

From the FIFO assumption and Lemma 1, it can be verified that for letter occurrences  $a, a'$ , where  $a$  is the  $h$ th occurrence of  $r_{ij}$  in  $w_i$ ,  $a'$  is the  $h$ th occurrence of  $s_{ji}$  in  $w_j$  if and only if there exists a condition  $b \in B$  with label  $(m, i, j)$  for some  $m$  such that  $(e', b), (b, e) \in F$ , where  $e$  is the event corresponding to  $a$  and  $e'$  is the event corresponding to  $a'$ .

Since each event in  $E$  represents a transition in the components of  $D$ , and since the transition relations of the components of  $D$  are deterministic, all occurrence nets associated to  $W$  with

respect to its different factorizations into a sequence of joint actions are isomorphic. Again, from FIFO assumption, the state of channels after the execution  $W$  are determined uniquely from its associated occurrence net. Thus, there exists uniquely only one complete state  $\bar{q}$  such that  $\bar{q}^0 \xrightarrow{W} \bar{q}$ .

For the simplicity, from now on in this paper, we restrict our attention to only the distributed systems for which the labeled transition systems are deterministic. This means that for every execution  $W$  of the systems, the occurrence net associated to  $W$  is defined uniquely (up to an isomorphism). Consequently, we can identify the events of the occurrence net associated to  $W$  with the letter occurrences in its components according to the corresponding in Lemma 1 and “immediately happens before” relation is defined uniquely for any execution represented by  $W$ . In the sequel, let  $D$  be a distributed system,  $W \in H$ ,  $\mathcal{M}$  be an occurrence net associated to  $W$ .

For an occurrence net  $\mathcal{M}$ , we call the conditions having no incoming arc initial conditions; each subgraph of  $\mathcal{M}$  forming an occurrence net is called subnet of  $\mathcal{M}$ . A subnet is left closed if the set of its events and conditions is a left closed subset of the set of events and conditions of  $\mathcal{M}$  w.r.t. the partial ordering generated by the flow relation.

**Theorem 2:** The occurrence net associated with each prefix  $W'$  of  $W$  is a left closed subnet of the occurrence net associated with  $W$ . Conversely, for each left closed subnet  $\mathcal{M}'$  of  $\mathcal{M}$  containing the initial conditions, there exists a prefix  $W'$  of  $W$  such that  $\mathcal{M}'$  is the occurrence net associated to  $W'$ .

*Proof:* The first part of the theorem is obvious from the assumption of  $D$  and the definition of  $\mathcal{M}$ . The second part is proven below.

Let  $\mathcal{M} = (B, E, F)$ , and let  $\mathcal{M}' = (B', E', F')$  be a left closed subnet of  $\mathcal{M}$ ,  $B' \subseteq B$ ,  $E' \subseteq E$ ,  $F'$  is the restriction of  $F$  on  $B' \cup E'$ . Since  $B' \cup E'$  is a left closed subset of  $B \cup E$  w.r.t.  $F^+$ , and since  $B \cup E$  is a finite set, there exists a topology sorting  $e_1 e_2 \dots e_m$  of  $E$  w.r.t.  $F^+$  such that  $e_1 e_2 \dots e_k$ ,  $k \leq m$ , is a topology sorting of  $E'$  w.r.t.  $F^+$ . For each  $e_i$ , let  $\bar{a}_i = \Lambda_j(a)$ , where  $a$  is the label of  $e_i$  and  $a \in A_j$ . Since the letter occurrence of each component of  $W$  are totally ordered by  $F^+$  by Lemma 1, we have  $W = \bar{a}_1 \bar{a}_2 \dots \bar{a}_m$ . For  $i = 1, \dots, m$  let  $\mathcal{M}_i$  be the subnet of  $\mathcal{M}$  containing the initial conditions of  $\mathcal{M}$  such that its set of events is  $\{e_1, \dots, e_i\}$ . Let  $\bar{q}_i$  be a state constructed by the labels of the conditions having no outgoing arc of  $\mathcal{M}_i$ . It can be seen by trivial induction on  $i$  that  $\bar{q}^0 \xrightarrow{\bar{a}_1} \bar{q}_1 \xrightarrow{\bar{a}_2} \dots \xrightarrow{\bar{a}_m} \bar{q}_m$ . Now, let  $W' = \bar{a}_1 \dots \bar{a}_k$ . We can prove that  $W' \subseteq W$  and  $\mathcal{M}_i$  is the occurrence net associated to  $W'$ . The details of the proof are left to the readers.

Theorem 2 establishes a one-to-one corresponding between the left closed subnets of the occurrence net associated to an execution  $W$  of the system and its prefixes. The following corollaries follow immediately from it.

**Corollary 1:**  $W' \in VAS(A)$  is an prefix of  $W$  if and only if  $w'_i$  is a prefix of  $w_i$  and the set of letter occurrences of the components of  $W'$  is a left closed set of the one of  $W$  w.r.t. the “happens before” relation.

*proof:* The subnet  $\mathcal{M}'$  containing the initial condition and the letter occurrences of  $W'$  is a left closed subnet of  $\mathcal{M}$ . From Theorem 2, there is a prefix  $W''$  of  $W$  such that  $\mathcal{M}'$  is its associated occurrence net. By the property of  $W'$ , it follows that  $W'' = W'$ .

**Corollary 2:** For each prefix  $w$  of  $w_i$ , there exists a prefix  $W'$  of  $W$  such that  $w'_i = w$ . Furthermore, the letter occurrences of the components of  $W'$  are those which happen before some letter occurrence in  $w$ .

*Proof:* Let  $a$  be the last letter occurrence of  $w$ . Denote  $a^- = \{e | e \text{ happens before } a\} \cup \{a\}$ .  $a^-$  is a left closed subset of the set of letter occurrences of  $W$ . Since if  $a^-$  contains a letter occurrence in  $w_i$  then it contains all letter occurrences preceding this letter occurrence, we can define  $w'_j$  to be the prefix of  $w_j$  containing only letter occurrences in  $a^-$ . Obviously,  $w'_i = w$ . From Corollary 1,  $W' = (w'_1, \dots, w'_N)$  is a prefix of  $W$ .

**Theorem 3:** Let  $W \in H$ ,  $W' = (w'_1, \dots, w'_N)$  and  $W'' = (w''_1, \dots, w''_N)$  be its prefixes. Define

$$\max(W', W'') = (u_1, \dots, u_N), \quad \min(W', W'') = (v_1, \dots, v_N),$$

where for  $i = 1, \dots, N$ ,

$$u_i = \underline{\text{if}} w''_i \text{ is a prefix of } w'_i \underline{\text{then}} w'_i \underline{\text{else}} w''_i,$$

$$v_i = \underline{\text{if}} w''_i \text{ is a prefix of } w'_i \underline{\text{then}} w''_i \underline{\text{else}} w'_i.$$

Then,  $\max(W', W'')$ ,  $\min(W', W'')$  are prefixes of  $W$  also.

*Proof:* Let  $\mathcal{M}$ ,  $\mathcal{M}'$  and  $\mathcal{M}''$  be associated occurrence nets of  $W$ ,  $W'$  and  $W''$  respectively. By Theorem 2,  $\mathcal{M}'$  and  $\mathcal{M}''$  are left closed subnets containing initial conditions of  $\mathcal{M}$ . Thus,  $\mathcal{M}' \cup \mathcal{M}'' = (B' \cup B'', E' \cup E'', F \cup F'')$ ,  $\mathcal{M}' \cap \mathcal{M}'' = (B' \cap B'', E' \cap E'', F \cap F'')$  are left closed subnets of  $\mathcal{M}$  containing the initial conditions. Also by Theorem 3, there exists prefixes  $U$ ,  $V$  of  $W$  such that the occurrence nets associated to them are  $\mathcal{M}' \cup \mathcal{M}''$  and  $\mathcal{M}' \cap \mathcal{M}''$  resp.. It is not difficult to see that  $U$  and  $V$  are respectively  $\max(W', W'')$  and  $\min(W', W'')$ .

**Corollary 3:** With the operations  $\min$ ,  $\max$  as above, the set of prefixes of  $W$  is a complete lattice.

### 3 General algorithm for maintaining consistent view of processes in distributed systems

The global view of a process in the distributed system may be local time, local states of processes, the truth value of a stable predicate, etc. We suppose that the view of a process at a moment

can contain only information extracted from the present or the past of the execution of the system. In general the global view of a process when it performs an action  $e$ , can contain only information extracted from those events and states that the event  $e$  causally depends on.

Let  $D$  be a distributed system,  $H$  be the set of its histories,  $W = (w_1, \dots, w_N) \in H$ . For a fixed index  $i$ , let  $w_i = a_1 a_2 \dots a_{l_i}$ . From Corollaries 1,2 and the proof of Theorem 3, for each  $j \in \{1, \dots, l_i\}$ , there exists a prefix  $W'$  of  $W$  such that the  $i$ th component of  $W'$  is the prefix of the length  $j$  of  $w_i$ . Such  $W'$  is called consistent global view of the process  $i$  at the point that it performs the  $j$ th action occurrence in  $w_i$ . The smallest prefix having this property that its existence is from Corollaries 1,2 is denoted by  $W_{i,j}$ .

**Theorem 4:** Let  $a$  be the name of the event  $e$ , which is the  $(l + 1)$ th letter occurrence in  $w_i$ . Then,

1. if  $a = r_{ij}$  for some  $j$ , then  $W_{i,l+1} = \max(W_{i,l}, W_{j,h})\Lambda_i(a)$ , where  $h$  is such that the  $h$ th letter occurrence in  $w_j$  is the corresponding send of  $e$ ,
2. if  $a \in A_i \cup \{s_{ij}\}$  for some  $j$ , then  $W_{i,l+1} = W_{i,l}\Lambda_i(a)$ .

*Proof:* If  $a = r_{ij}$  for some  $j$ , then every component of  $\max(W_{i,l}, W_{j,h})\Lambda_i(a)$  is a prefix of  $w_i$ . By the definition of  $W_{i,l}$ ,  $W_{j,h}$ , the set  $O$  of all letter occurrences in its components is a left closed subset of the one of  $W$  w.r.t. the ‘‘happens before’’ relation. Furthermore,  $O$  is the smallest set containing  $e$  with this property since the  $h$ th letter occurrence in  $w_j$  and the  $l$ th letter occurrence in  $w_i$  are all elements which happen immediately before  $e$ . By Corollaries 1 and 2,  $\max(W_{i,l}, W_{j,h})\Lambda_i(a)$  is the smallest prefix of  $W$  containing  $e$ . Hence,  $\max(W_{i,l}, W_{j,h})\Lambda_i(a) = W_{i,l+1}$ . The remaining case is proven in a similar way.

Theorem 4 is a foundation of our algorithm. We introduce a function  $f$  for representing the extraction of the information that we are interested in from global view of processes in the system. The function  $f$  is from the set  $\text{Pref}(W)$  of prefixes of  $W$  to a partial ordered lattice  $K$ . From Theorem 4, in order to maintain the values of  $f$ , we claim that  $f$  satisfies the following conditions:

$$f(\max(W', W'')) = s(f(W'), f(W'')), \quad f(W'\Lambda_i(a)) = c_i(f(W'), a),$$

where  $s$  is a given function from  $\text{Pref}(W) \times \text{Pref}(W)$  to  $K$ , and  $c_i$  is a given function from  $K \times A_i$  to  $K$ . The function  $s$  represents the synthesizing of global knowledge about the system of processes, while the function  $c_i$  represents the fact that the process  $i$  can compute the global information after it has performed an action from its consistent global information it has before.

Our algorithm is given as follows. The algorithm is presented as the construction of a distributed system  $D'$  from  $D$ , which is behaviorally equivalent to  $D$  with  $f(W_{i,j})$  being kept as a component of the local state of the process  $i$  when it performs the  $j$ th action occurrence in the execution

of the system resulting  $W$ . The messages of  $D'$  are messages of  $D$  stamped with current global view of the sending processes.

Let  $D$  be as in Definition 2. Then,  $D' = (P'_1, \dots, P'_N, C', \rightarrow, M')$ , where  $P'_i = (Q_i \times f(H), \rightarrow, A_i)$ ,  $M'_{ij} = M_{ij} \times f(H)$ , the relation  $\rightarrow$  for  $P'_i$  is defined below. Let  $q, q' \in Q_i$ ,  $a \in A_i$ ,  $W, W' \in H$ .

1. if  $q \xrightarrow{a} q'$  then  $(q, f(W)) \xrightarrow{a} (q', c_i(f(W'), a))$ ,
2. if  $q \xrightarrow{s_{ij}} (q', m)$  then  $(q, f(W)) \xrightarrow{s_{ij}} ((q', c_i(f(W), s_{ij})), (m, c_i(f(W), s_{ij})))$ ,
3. if  $(q, m) \xrightarrow{r_{ij}} q'$  then  $((q, f(W)), (m, f(W'))) \xrightarrow{r_{ij}} ((q', c_i(s(f(W), f(W'))), r_{ij}))$ .

The initial state of  $D'$  is  $\bar{q}^0 = ((q_1^0, f(\Lambda)), \dots, (q_N^0, f(\Lambda)), \emptyset)$ . It can be proven that:

**Theorem 5:**  $D$  and  $D'$  are behaviorally equivalent, i.e.  $H = H'$ , where  $H'$  is the set of histories of  $D'$ . Furthermore,  $\bar{q}^0 \xrightarrow{W} \bar{q}' = ((q_1, d_1), \dots, (q_N, d_N), G')$  if and only if  $\bar{q}^0 \xrightarrow{W} \bar{q} = (q_1, \dots, q_N, G)$  for some  $G$ , and  $d_i$  is  $f(W_{i,i})$  which is the global information of the process  $i$  after it has performed the last action occurrence in the  $i$ th component of  $W$ .

*Proof:* Let  $T$  and  $T'$  be labeled transition systems for  $D$  and  $D'$  respectively. The proof goes by direct induction on the length of derivation taking into account Theorem 4 and the construction of  $D'$ . The details are left to the readers.

Some applications of the algorithm derived from different interpretations of the functions  $f$ ,  $s$  and  $c_i$  are presented below.

**(1). Partial ordered logical time:** Let

$$f(W') = (|w'_1|, \dots, |w'_N|), K = \mathbf{N}^N,$$

$$s((n_1, \dots, n_N), (n'_1, \dots, n'_N)) = (\max(n_1, n'_1), \dots, \max(n_N, n'_N)),$$

$$c_i((n_1, \dots, n_N), a) = (n_1, \dots, n_i + 1, \dots, n_N),$$

where  $\mathbf{N}$  is the set of nonnegative integers,  $a \in A_i$ ,  $n_i \in \mathbf{N}$ ,  $i = 1, \dots, N$ .

If we consider the length of a local computation performed by a local process as the value of its local clock, then  $f(W)$  is the tuple of values of the local clock of all processes after the execution  $W$  of the system.  $f(W_{i,j})$  says that in any execution of the system resulting  $W$  when the process  $i$  performs the  $j$ th action occurrence, the tuple of values of local clock of all processes is at least  $f(W_{i,j})$ . Furthermore, the  $l$ th action occurrence in  $w_k$  happens before the  $j$ th action

occurrence in  $w_i$  if and only if  $f(W_{k,l}) < f(W_{i,j})$  (componentwise). So,  $f(W_{i,j})$  is called partial ordered logical time of the event corresponding to the  $j$ th action occurrence in  $w_i$  which preserves the happens before relation between events. In this case, our algorithm becomes Fridge's one presented in [4].

**(2). Checkpointing procedure:** In the system that failure can occur, reachable global states of the systems have to be recorded at some points during the computation of the systems for recovering from failure. Since  $W_{i,j}$  is a prefix of  $W$ , the global state of the system after the execution of  $W_{i,j}$  is a reachable state and can be used to be recorded as a check point. For this purpose, let

$$f(W') = ((|w'_1|, q'_1), \dots, (|w'_N|, q'_N)),$$

$$s(((n'_1, q'_1), \dots, (n'_N, q'_N)), ((n''_1, q''_1), \dots, (n''_N, q''_N))) = ((n'''_1, q'''_1), \dots, (n'''_N, q'''_N)),$$

where for  $i = 1, \dots, N$ ,  $a \in A_i$ ,  $(n'''_i, q'''_i) = \underline{\text{if}} n'_i < n''_i \underline{\text{then}} (n''_i, q''_i) \underline{\text{else}} (n'_i, q'_i)$ ,

$$c_i(((n'_1, q'_1), \dots, (n'_N, q'_N)), a) = ((n'_1, q'_1), \dots, (n'_i + 1, q''_i), \dots, (n'_N, q'_N)),$$

in which  $q''_i$  is the local state of the process  $i$  after it has performed the action  $a$  in the local state in  $((n'_1, q'_1), \dots, (n'_N, q'_N))$ . The tuple containing local states in  $f(W_{i,j})$  forms a reachable state of the system. Our algorithm in this case becomes the checkpointing procedure presented in [12]. The readers are referred to this paper for the discussion of the stabilization and the advantages of the algorithm.

**(3). Debugging distributed programs, state intervals:** In debugging distributed programs, ones have to determine which local states that a given local state during the execution of the programs can depend on. Since  $W_{i,j}$  contains all those action occurrences which happen before the  $j$ th action occurrence in  $w_i$ , the local state of the process  $i$  after it has performed this action can be derived from the occurrence net associated to  $W_{i,j}$ . Let, for this purpose,

$$f(W') = ((|w'_1|, q'_1), \dots, (|w'_N|, q'_N)),$$

where for  $i = 1, \dots, N$ ,  $q'_i$  is the local state of the process  $i$  before it performs the last action occurrence in  $w'_i$ . From the definition of  $W_{i,j}$ ,  $f(W_{i,j})$  contains all local states that the local state of the process  $i$  after it has performed the  $j$  action occurrence in  $w_i$  can depend on.

In order to reduce the cost of dependency tracking in debugging distributed programs (size of the function  $f$ ), we can take  $f$  to be  $f(W') = ((n'_1, q'_1), \dots, (n'_N, q'_N))$ , where  $n_i$  is the number of receipts in  $w'_i$ ,  $q'_i$  is the local state of the process  $i$  after its  $n_i$ th receipt.

The function  $s$  in both these cases are defined as in (2). The function  $c_i$ 's are defined by the local transitions of processes.

The  $f(W_{i,j})$  says that the local state of the process  $i$  after it has performed the  $j$ th action occurrence in  $w_i$  can depend on those local states of the process  $k$ ,  $k = 1, \dots, N$ , which occur between the  $n_k$ th receipt and the  $(n_k + 1)$ th receipt. The local states of the process  $k$  occurring between the  $n_k$ th and  $(n_k + 1)$ th receipts during its execution is called state interval. This technique has been used in [5]. The readers are referred to the paper for a debugging mechanism using the algorithm with this interpretation of the function  $f$ .

**(4). Detecting “possible property”:** Suppose that each process has a local variable  $x_i$ .  $\phi$  is a predicate of the variables  $x_i$ 's. The value of  $x_i$  is updated by the process  $i$ . Suppose that we wish to determine if there is an execution of  $D$  resulting  $W$  such that  $\phi$  is true at a point of the execution. This means that we have to determine if there is a reachable global state in which the tuple of values of  $x_i$ 's makes  $\phi$  true. Let, for this purpose,  $f(W') = ((n_1, v_1), \dots, (n_N, v_N))$ , where for  $i = 1, \dots, N$ ,  $n_i$  is the number of action occurrences in  $w'_i$  that update  $x_i$ ,  $v_i$  is the last value of  $x_i$ . The function  $s$  in this case is defined as in (2). The function  $c$  is defined as

$$c_i(((n_1, v_1), \dots, (n_N, v_N)), a) = \begin{array}{l} \text{if } a \text{ updates } x_i \text{ then} \\ ((n_1, v_1), \dots, (n_i + 1, v'_i), \dots, (n_N, v_N)) \\ \text{else } ((n_1, v_1), \dots, (n_N, v_N)) \end{array}$$

where  $v'_i$  is the new value of  $x_i$  (updated by  $a$ ). Then  $f(W_{i,j})$  contains a tuple  $(v_1, \dots, v_N)$  in a reachable state of  $D$  after the execution represented by  $W_{i,j}$ . Thus, if  $\phi(v_1, \dots, v_N)$  is true, then  $\phi$  is detected.

The algorithm work effectively if the processes inform a monitoring process when they update their local variable.

## 4 Conclusion

We have presented our approach to the distributed systems and our algorithm for maintaining consistent global view of processes in distributed systems. Our algorithm includes as a special cases some familiar algorithms such as partial ordered logical time ([4]), checkpointing procedure ([12]) and distributed debugging ([5]). One of the contributions of our work is to show how global view of a process is changing during the execution of a system, how processes can exchange their knowledge about the system. The system designers can base on this observation to design protocols for verifying the possible global state properties of the systems.

The more often the local processes of a system exchange messages, the more recent the global view is. Thus, by combining with introducing control messages and by incorporating knowledge into the model, we hope that efficient algorithms for attainment of some kind of common knowledge, such as concurrent common knowledge (see [11]) can be derived. This will be in our future work.

## References

- [1] H. E. Bal, J. C. Steiner, A. S. Tanenbaum, "Programming Languages for Distributed Computing Systems," *ACM Computing Surveys*, Vol. 21, No. 3, pp. 261-322, 1989.
- [2] E. Best, C. Fernandez, "Non-Sequential Processes," *A Petri Net View*, Springer-Verlag, 1988.
- [3] K. M. Chandy, L. Lamport, "Distributed Snapshots: Determining The Global State of Distributed Systems," *ACM Trans. Comput. Systems*, Vol. 3, No. 1, pp. 63-75, 1985.
- [4] C. Fridge, "Logical Time in Distributed Computing Systems," *Computer* Vol. 24, 8, pp. 28-33, 1991.
- [5] A. P. Goldberg, A. Gopal, A. Lowry, R. Strom, "Restoring Consistent Global States of Distributed Computations," *Proceedings of The ACM/ONR Workshop on Parallel and Distributed Debugging, ACM SIGPLAN Notices*, Vol. 26, No. 12, pp. 144-154, 1991.
- [6] J. I. Halpern, R. Fagin, "Modeling Knowledge and Actions in Distributed Systems," *Distributed Computing*, Vol. 3, pp. 159-177, 1989.
- [7] D. V. Hung, "A Model for Analyzing Distributed Systems," *Journal of Informatics and Cybernetics*, Hanoi, Vol. 1, No. 2, pp. 15-23, 1991.
- [8] D. V. Hung, "Labeled Transition Approach to Distributed Computing Systems," *Journal of Informatics and Cybernetics*, Hanoi, Vol. 8, No. 3, pp. 19-32, 1992.
- [9] R. Keller, "Formal Verification of Parallel Programs," *CACM*, Vol. 19, pp. 561-572, 1976.
- [10] L. Lamport, N. Lynch, "Distributed Computing: Models and Methods," *Chapter 18, Handbook of Theoretical Computer Science*, J. van Leenwen, editor, pp. 1157-1199, 1990.
- [11] P. Panangaden, K. Taylor, "Concurrent Common Knowledge: Defining Agreement for Asynchronous Systems," *Distributed Computing*, Vol. 6, pp. 73-93, 1992.
- [12] K. Saleh, I. Ahmad, K. Al. Saqabi, A. Agarwal, "Dynamic Checkpointing Procedure for the design of stabilizing Protocols," *Information and Software Technology*, Vol. 38, No. 8, pp. 479-485, 1993.