Tạp Chí Tin Học và Điều Khiển Học
Tập VIII(1992), Số 3, 19-32.

Labelled Transition System Approach to Distributed Computing Systems

Dang Van Hung
Institute of Informatics
National Centre for Scientific Research of Vietnam

I. Introduction

Labelled transition systems defined by Keller [15] in 1976 has been used widely for defining operational semantics for parallel programming languages as well as models for parallel systems [1,5,15]. One advantage of them is that they are very simple. However, semantics represented by them is interleaving one, i.e. they model independence by arbitrary interleaving. The situation that two events are independent is modeled by saying that the two events may occur in any order, which refers to a global ordering of the events. In order to overcome this shortcoming, Mazurkiewicz has introduced the concept of traces. Each Mazurkiewicz's trace contains sequences of events which are equivalent by an independence. One problem with traces is that the independence is fixed in the system, which restricts the expressive power of the models with traces. Vectors of firing sequences introduced by Shield [19] represents the behavior of concurrent and distributed systems more intuitively. This model has the same expressive power as traces [11], and is suitable for synchronous systems only.

In this paper we combined the mentioned concepts by considering an event of concurrent and distributed systems to contain atomic events executed in parallel by component processes. In this way we can define labelled transition systems, an intuitive model for distributed systems, which can expresses the concurrency explicitly and can be used in designing, implementing and verifying distributed algorithms.

In order to show the advantages of our model we relate it to partial orderings of events [5,6,16,18] and consider the problem of determining global states and the logical time of distributed systems. By keeping partial knowledge of the others processes in the states of each process in the systems, we propose a new algorithm for distributed snapshots. It is shown that the amount of global

information which can be maintained in the local states of the processes of the system by exchanging messages is not more than what derived from actions leading to them.

The paper is organized as follows. In the next section we introduce labelled transition systems for distributed computing systems and some behavioral properties. In the third section we relate our models to partial orderings of events, define the logical time in distributed systems and present a general algorithm for the maintenance of interesting global knowledge for solving some problems concerning the distributed systems.

2. Labelled Transition Systems for Distributed Computing

In this section we introduce a formal, general model for distributed computing systems and associate them with labelled transition systems in order to describe their behavior. In this model of the behavior, the semantics of the systems becomes easier to understand, and we can formulate precisely some problems with the systems. The distributed computing systems considered in this paper are taken from [2,10] 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 any 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 is blocked until the receiver has accepted 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 also assume that each message sent or accepted involves an explicit action.

We start with the definition of labelled transition systems which are essentially those given in [6,15,17].

Definition 1 (Labelled transition systems). 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 $\rightarrow$ is the transition relation.

In the sequel, we shall write $p - a \rightarrow q$ instead of $(p, q, a) \in \rightarrow$. P is said to be deterministic iff

$$|\{q \in Q | p - a \rightarrow q\}| \leq 1, \ \forall p \in Q, \ a \in A.$$

Definition 2 (Distributed Computing Systems). A distributed computing system with N processes is a tuple $D = (P_1, P_2, ..., P_N, C, \rightarrow C, M)$, where

(i) $P_i = (Q_i, \rightarrow i, A_i)$ is a labelled transition system, called the i^{th} process of the system, Q_i is the set of local states of the i^{th} process, and A_i is the set of its local computation actions,

(ii) C is the set of communication actions of the system, $C = \cup_{i,j=1}^{N} \{s_{ij}, r_{ij}\}$, r_{ij} is the message receive of the i^{th} process from j^{th} process, s_{ij} is the message send of the i^{th} process destinationed the j^{th} process,

(iii) M is a matrix, of which the i D row j D column element M_{ij} is a set of possible messages from the process i to the process j,

(iv) $\rightarrow C$ is a transition relation labelled over C, $-s_{ij} \rightarrow C \subseteq Q_i \times (M_{ij} \times Q_i)$, and $-r_{ij} \rightarrow C \subseteq (Q_i \times M_{ji}) \times Q_i$, $i, j = 1, 2, ..., N$

For the simplicity, we assume that the sets $C, A_i, Q_i, M_{ij}, i, j = 1, 2, ..., N$ are pairwise disjoint.

D is said to be deterministic iff each labelled transition system in the system is deterministic.

In order to describe the behaviour of distributed systems we first define the set of actions of the system at each point of discrete time. An action of the system at each moment consists of actions of all process in the system including local actions, communicating actions and the idle action.

Definition 3 (Action of distributed systems). Let D be a distributed system given as above. $\sum_i = A_i \cup \{r_{ij}, s_{ij}, \lambda | j = 1, 2..., N\}$ is called the set of actions executed by the process i, where λ denotes the idle action. The set $A = \sum_1 \times \sum_2 \times ... \times \sum_N$ is called the set of possible actions of the system, while the set $A_s = \{\overline{a} = (a_1, a_2, ..., a_N) | \overline{a} \in A \& a_i = s_{ij}$ iff $a_j = r_{ji}\}$ is called the set of its synchronous actions.

The behavior of the system now can be describe as a set of sequences of actions of the system, called vector of action sequences.

Definition 4 (Vector of action sequences). Let

$$VAS(A) = \Sigma_1^* \times \Sigma_2^* \times ... \Sigma_N^*.$$

Each element of $VAS(A)$ is called a vector of action sequences of D.

For the simplicity of our notation, in this paper we adopt the following conventions:

For $\overline{w} \in VAS(A)$, w_i will denote its i^{th} component. For $\overline{w}$, $\overline{w'} \in VAS(A)$, the product $\overline{ww'}$ (concatenation) of $\overline{w}$ and $\overline{w'}$ is defined as

$$\overline{ww'} = (w_1 w'_1, w_2 w_2, ..., w_N w'_N),$$

For $B \subseteq \sum$ and $w = a_1 a_2 ... a_n \in \sum^*$, $\#_B w$ will denote the number of occurrences of the elements of B in w; n is called the length (number of letter occurrences) of w and will be denoted by $|w|$. The elements of the set $\{(i, a_i) | i = 1, 2, ..., n\}$ is said to be letter occurrences of w. For a letter occurrence (i, a_i), we call a_i its label. A single set will be identified with its element. $\overline{\lambda}$ will denotes the element $(\lambda, \lambda, ..., \lambda) \in A$, while $\lambda_i(a)$ denotes the element in A of which all components is λ except that the ith component is a.

In order to associate a labelled transition system to a given distributed system, we have to specify the state space for it. A state of the distributed system will consist of states of its processes and a state of the communication channels. The states of the channels will be the sequences of messages which have not been accepted.

Definition 5 (State space of distributed systems). Let D be a distributed system given as above. The state space for D is the set

$$S = Q_1 \times Q_2 \times ... \times Q_N \times \Gamma,$$

where Γ is the set of $N \times N$ matrices $\gamma's$ of which the $i-$Drow j-column element is a priori-queue of possible messages from the process i to the process j. ($\gamma_{ij} \in M_{ij}^*$).

Each element of S will be called a complete state of the system, while its n first components is called a state of the system.

In this paper we consider a state of a channel as a FIFO queue of possible messages. The empty queue will represent the fact that there is no message on the channel and will be denoted by λ. For FIFO queue γ_{ij} (content of the channel from the process i to the process j) the operation $put(\gamma_{ij}, m)$ means putting the message m into the channel, resulting $m\gamma_{ij}$ as its new content, while the operation $getout(\gamma_{ij})$ returns the element at the front of the queue and takes it out from the queue if the queue is not empty. Otherwise, the operation is undefined.

Labelled transition system for the distributed system D now is defined as follows.

Definition 6 (Labelled transition system for D). Let S and A be as above. The labelled transition system

$$T = (S, \rightarrow, A)$$

is said to be the labelled transition system for D, where $\rightarrow$ is defined below.

Let

$$AL = (A_1 \cup \{\lambda\}) \times (A_2 \cup \{\lambda\}) \times ...(A_N \cup \{\lambda\}).$$

$$AR = (R_1 \cup \{\lambda\}) \times (R_2 \cup \{\lambda\}) \times ...(R_N \cup \{\lambda\}),\ R_i = \cup_{j=1}^{N}\{r_{ij}\},\ i = 1, 2, ..., N,$$

$$AS = (S_1 \cup \{\lambda\}) \times (S_2 \cup \{\lambda\}) \times ...(S_N \cup \{\lambda\}),\ S_i = \cup_{j=1}^{N}\{s_{ij}\},\ i = 1, 2, ..., N.$$

In the sequel, when labelled transition relations can be distinguished one from another by the set of labels, we omit the subscript if it does not cause any confusion.

For $\overline{q} = (q_1, q_2, ..., q_N, \gamma),\ \overline{q}' = (q_1', q_2', ..., q_N', \gamma') \in S, \overline{a} = (a_1, a_2, ..., a_N) \in A$, we say $\overline{q} - \overline{a} \rightarrow \overline{q}'$ iff

(i). $\overline{a} \in AL \& (\gamma = \gamma') \& (\forall i \leq N) :\ q_i - a_i \rightarrow q_i'$,

(ii). $(\overline{a} \in AS) \& (\forall i \in N)$:
$((a_i = \lambda) \Rightarrow ((q_i = q_i') \& (\forall j \leq N) :\ (\gamma_{ij} = \gamma_{ij}')) \& ((a_i = s_i k_i) \Rightarrow ((q_i - s_i k_i \rightarrow (m, q_i')) \&$
$(put(\gamma_i k_i, m) = \gamma_i' k_i) \&$
$((\forall j \leq N,\ j \neq k_i) :\ (\gamma_{ij}' = \gamma_{ij}))))$,

(iii). $(\overline{a} \in AR) \& (\forall i \leq N)$:
$((a_i = \lambda) \Rightarrow ((q_i = q_i') \& (\forall j \leq N) :\ (\gamma_{ij} = \gamma_{ij}'))$
$\& ((a_i = r_{ik_i}) \Rightarrow ((m = getout(\gamma_{ik_i})) \& \& ((q_i, m) - r_{ik_i} \rightarrow q_i') \& ((\forall j \leq N,\ j \neq k_i) :\ (\gamma_{ij}' = \gamma_{ij})))),$

(iv).
$E(\overline{a} = \overline{a}_1\overline{a}_2\overline{a}_3) \& (\overline{a}_1 \in AL, \overline{a}_2 \in AS, \overline{a}_3 \in AR) \& (\exists \overline{p}, \overline{p}' \in S)$:

$$(\overline{q} - \overline{a}_1 \rightarrow \overline{p} - \overline{a}_2 \rightarrow \overline{p}' - \overline{a}_3 \rightarrow \overline{q}').$$

(Notice that for each $\overline{a} \in A$, there exist uniquely $\overline{a}_1 \in AL$, $\overline{a}_2 \in AS$, $\overline{a}_3 \in AR$ such that $\overline{a}$ can be written as $\overline{a} = \overline{a}_1\overline{a}_2\overline{a}_3$.

The definition achieves the general behaviour of distributed systems. Each process in a system can perform its local computations independently from the others. The message passing is synchronous if the specified channel is empty and the receiver is ready to accept message. The accept action can be delayed until the receiver becomes ready to perform it.

The relation $\rightarrow$ is extended to the one labeled over $VAS(A)$ in the natural way as follows:

For $\overline{q}, \overline{q}' \in Q$, $\overline{w} \in VAS(A)$, $\overline{q} - \overline{w} \rightarrow \overline{q}'$ iff there exist $\overline{a}_1, \overline{a}_2, ..., \overline{a}_k \in A$, $\overline{q}_0, \overline{q}_1, \overline{q}_k \in S$, and s.t $\overline{q}_0 = \overline{q}$, $\overline{q}_k = \overline{q}'$, and $\overline{q}_{i-1} - \overline{a}_i \rightarrow \overline{q}_i$, $i = 1, 2, ..., k$.

Definition 7 (Transition system for synchronized computations). Let T be the labelled transition system for D. The restriction of T on the label set A_s

$$T_s = (S, \rightarrow, A_s)$$

is called the labelled transition system for synchronized computations of D, or the synchronized transition system for D.

Let $\overline{q}_0 = (q_1^0, q_2^0, ..., q_N^0, \Phi)$ be a fixed state, called initial state of D. (Φ denotes the matrix having empty queues as its elements).

Definition 8 (Behaviour and reachable states of distributed systems). Let

$$B = \{\overline{w} \in VAS(A) | \ \overline{q}_0 - \overline{w} \rightarrow \overline{q}, \ for \ some \ \overline{q} \in S\},$$

$$R = \{\overline{q} \in S | \ \overline{q}_0 - \overline{w} \rightarrow \overline{q} \ for \ some \ \overline{w} \in VAS(A)\}.$$

We call B the behaviour, R the set of reachable states of D (with the initial state $\overline{q}_0$).

With the substitution of $VAS(A)$ by $VAS(A_s)$ in this definition, we get the definition of the synchronized behavior and the set of synchronized reachable states of D.

Now, we relate our model of behaviour to the net theory by associating to each element of B an occurrence net. The readers, who are not familiar with net theory, are refered to [18] for their details.

From the definition of T it follows that if $\overline{q}_0 - \overline{w} \rightarrow \overline{q}$, $\overline{w}$ can be written as $\overline{a}_1\overline{a}_2...\overline{a}_K$ such that $\overline{q}_0 - \overline{a}_1 \rightarrow \overline{q}_1 \rightarrow \overline{a}_2 ... - \overline{a}_K \rightarrow \overline{q}_K$, $\overline{q}_K = \overline{q}$, where $\overline{a}_k, k \leq K$, is an element of A having only one component different from λ.

Let $\overline{q}_k = (q_1^k, q_2^k, ..., q_N^k, \gamma_k)$, $\overline{w}_k = \overline{a}_1\overline{a}_2...\overline{a}_k, k \leq K, \overline{w} = (w_1, w_2, ..., w_N)$. The occurrence net $\pi(\overline{w}) = (G, E, F)$ and the labelling function l: $G \cup E \rightarrow (\cup_{i=1}^N Q_i) \cup (\cup_{i,j=1}^N M_{ij})$ are constructed by induction on k as follows.

$k = 0$:

Let $\pi(\lambda) = (G_0, \emptyset, \emptyset), G_0 = \{g_1^0, g_2^0, ..., g_N^0\}, l_0(g_j^0) = q_j^0, j \leq N$. $I_0 = \emptyset$ (R is introduced for the construction of $\pi(\overline{w}_{k+1})$.

$k \to k+1$:

Let $\pi(\overline{w}_k)' = (G_k, E_k, F_k), l_k : G_k \cup E_k \to (\cup_{i=1}^N Q_i) \cup (\cup_{i,j=1}^N M_{ij})$ have been constructed, I_k be a partial function from G_k to $\{(i,j,n) | i,j \leq N, n \geq 0\}$ such that $I_k(g)$ is defined when $l(g) \in \cup_{i,j=1}^N M_{ij}$. Let $\overline{a}_k = \overline{\lambda}_i(a)$.

Then, $E_{k+1} = E_k \cup \{e\}, l_{k+1}(e) = a, l_{k+1}|_{E_k \cup G_k} = l_k$, where e is a new transition $\varphi|_\Omega$ denotes the restriction of φ on Ω. Depending on a, we have the following cases:

a). $a \in A_i$. By the definition of $\to, q_i^k - a \to q_i^{k+1}$. Then, $B_{k+1} = G_k \cup \{g\}$, where g is a new place, $l_{k+1}(g) = q_i^{k+1}$. Let g' be the place in B_k having no outgoing arc such that $l_k(g') = q_i^k$. (Such a place exists uniquely). Now, $F_{k+1} = F_k \cup \{(g', e), (e, g)\}$.

b). $a = s_{ij}$ for some $j \leq N$. By the definition of $\to$, $q_i - a \to (q_i^{k+1}, m)$. Then, $G_{k+1} = G_k \cup \{g, g'\}$, where g, g' is a new place, $l_{k+1}(g) = q_i^{k+1}, l_{k+1}(g') = m$, $I_{k+1}(g') = (i,j,n)$, where n is the number of the occurrences of s_{ij} in the i^{th} component of $\overline{w}_k$. Let g'' be the place in G_k having no outgoing arc such that $l_k(g'') = q_i^k$. (Such a place exists uniquely). Now, $F_{k+1} = F_k \cup \{(g'', e), (e, g), (e, g')\}$.

c). $a = r_{ij}$ for some $j \leq N$. By the definition of $\to, (q_i^k, m) - a \to q_i^{k+1}$, where $m = getout(\gamma_{ji}^k)$. Then, $G_{k+1} = G_k \cup \{g\}$, where g is a new place. $l_{k+1}(g) = q_i^{k+1}$. Let g' be the place with $I_k(g') = (j,i,n)$, where n is the smallest number such that g' has no outgoing arcs, g'' be the place in G_k having no outgoing arc such that $l_k(g'') = q_i^k$. (Such places exist uniquely). Now, $F_{k+1} = F_k \cup \{(g'', e), (e, g), (g', e)\}$.

Example 1. Let

$$D = (P_1, P_2, \to, C, M), \; M_{12} = \{1, 0\}$$

$$\begin{aligned} P_1 = (\{q_0, q_1, q_2\}, \to, \{b\}), \qquad & P_2 = (\{p_0, p_1, p_2\}, \to, \{a\} \\ (q_0, 1) - r_{12} \to q_1, \qquad & p_1 - s_{21} \to (p_0, 1), \\ (q_0, 0) - r_{12} \to q_2, \qquad & p_0 - s_{21} \to (p_2, 0), \\ q_0 - a \to q_1, \qquad & p_0 - b \to p_1, \end{aligned}$$

Let

$\overline{w} = (r_{12} a r_{12}, b s_{21} b s_{21} s_{21}) =$

$(\lambda, b)(\lambda, s_{21})(r_{12}, \lambda)(\lambda, b)(a, \lambda)(\lambda, s_{21})(\lambda, s_{21})(r_{12}, \lambda), q_0 = (q_0, p_0, \Lambda)$, where

$$\Lambda = \begin{bmatrix} \lambda & \lambda \\ \lambda & \lambda \end{bmatrix}$$

The occurrence net defined for $\overline{w}$ is represented by the Fig. 1

Lemma 1. $\pi(\overline{w})$ is defined. Furthermore, let

$$S^0 = \{g \in G | \text{ has no outgoing arc and } l(g) \in \cup_{i=1}^N Q_i\},$$
$$\Gamma_{ij}^0 = \{g \in B | \text{ has no outgoing arc and } l(g) \in M_{ij}\},$$

$a_{ij} = g_1 g_2 ... g_h$ be the sorting of its elements such that g precedes g' iff $n < n'$, where $I(g) = (i,j,n)$ and $I(g') = (i,j,n')$, $i,j \leq N$.

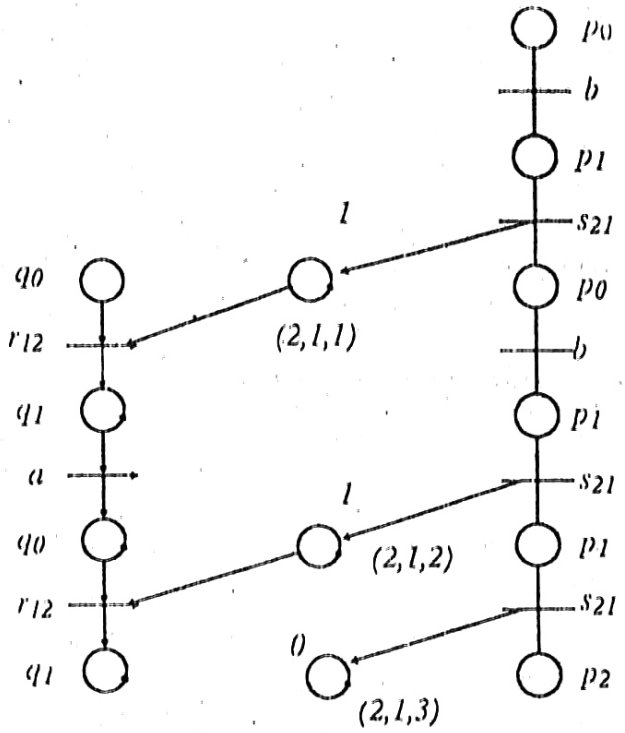

Fig. 1

Then, $q_i = l(S^0) \cap Q_i, \gamma_{ij} = l(\alpha_{ij})$

Lemma 2. There exists uniquely an one-to-one mapping τ between the set of letter occurrences of $\overline{w}$ and the set E of transitions, preserving labels such that:

- for letter occurrences a, b in w_i, a immediately follows b in w_i if and only if there is a place g with label in Q_i such that $(\tau(b), g), (g, \tau(a)) \in F$,
- for letter occurrences a, b, a is the h^{th} occurrence of r_{ij} in w_i, b is the h^{th} occurrence of s_{ji} in w_j for some h if and only if there is a place g in G with label in M_{ji} such that $(\tau(b), g), (g, \tau(a)) \in F$.

The lemmas 1 and 2 can be verified straightforward by induction on k. Details of the proofs are left to the reader.

From Lemmas 1 and 2, it follows that many interesting behavioural aspects of distributed systems can be formulated in terms of vectors of action sequences.

Theorem 1. *If distributed system is a deterministic, so is its transition system, i.e.*

$$|\{q \in S | p - \overline{w} \rightarrow q\}| \leq 1, \forall p \in S, \overline{w} \in VAS(A).$$

Proof. Since each transition in E represents a transition labelled over A_i $(i = 1, 2, ..., N)$ or C, it follows from Lemma 2 that if the component transition systems in D are deterministic then all occurrence nets constructed for $\overline{w}$ corresponding to its different factorizations into a sequence of

actions of D are isomorphic. Hence, by Lemma 1, the state $\overline{q}$ is determined uniquely independently from the way of factoring $\overline{w}$.

It follows from Theorem 1 that the synchronized transition system for a deterministic distributed system is a deterministic one, and if $\overline{p} - \overline{w} \rightarrow \overline{q}$, $\overline{w} \in VAS(A_s)$, then $\overline{q}$ does not depend on the way of factoring $\overline{w}$ into a concatenation of synchronous actions.

In practice, parallel programs are frequently designed to be synchronized (e.g., OCCAM, CSP programs), and then are implemented by asynchronous message passing distributed systems. Theorem 1 says that such implementation is correct.

3. Maintaining the amount of global information in local states of process

We will formulate some basic notions such as causality relation, logical time, of the system, and then propose an algorithm for maintaining the amount of global information of the system from which some properties of interest can be derived. The implementations of the algorithm may be expensive, but it shows the upper bound of what can be maintained in the states of processes by sending massages.

For simplicity, we present the above mentioned notions and results only for the case of asynchronous message passing distributed systems only. The results, however, can be modified and extended to the case of synchronous and asynchronous message passing distributed systems as well.

Let, in the sequel, D be a distributed system, $T = (S, \rightarrow, A)$ its labelled transition system with the initial state $\overline{q}_0$, B its behaviour.

For $\overline{w} \in B$, an initial part of $\overline{w}$ is defined to be a $\overline{w}' \in B$, such that $\overline{w} = \overline{w}'\overline{u}$ with some $\overline{u} \in VAS(A)$. Let $init(\overline{w})$ denote the set of all initial parts of $\overline{w}$.

For $\overline{w}' = (w'_1, w'_2, ..., w'_N), \overline{w}'' = (w''_1, w''_2, ..., w''_N) \in init(\overline{w})$, denote:

$$\max(\overline{w}', \overline{w}'') = (\max(w'_1, w''_1), \max(w'_2, w''_2), ..., \max(w'_N, w''_N)),$$
$$\min(\overline{w}', \overline{w}'') = (\min(w'_1, w''_1), \min(w'_2, w''_2), ..., \min(w'_N, w''_N)),$$

where for any u, v over an alphabet $\sum$,

$$\max(u, v) = \begin{cases} u, & \text{if } v \text{ is a prefix of } u, \\ v, & \text{if } u \text{ is a prefix of } v, \\ underfined & \text{otherwise,} \end{cases}$$

$$\min(u, v) = \begin{cases} u, & \text{if } u \text{ is a prefix of } v, \\ v, & \text{if } v \text{ is a prefix of } u, \\ underfined & \text{otherwise,} \end{cases}$$

Lemma 3. *With the oprerations max, min defined as above, $init(\overline{w})$ is a complete finite lattice.*

Proof. The lemma follows immediately from the fact that $\overline{w}' = (w'_1, w'_2, ..., w'_N)$ is an initial part of $\overline{w} = (w_1, w_2, ..., w_N)$ if and only if:

a) w_i' is a prefix of w_i, $i = 1, 2, ..., N$ b) $\forall i, j \leq N$, the number of sends to j in w_i' is not greater than the number of receives from i in w_j'.
The details of the proof are omitted.

$init(\overline{w})$ is intended to represent all the possible past parts of the execution resulting $\overline{w}$ of the system.

For a given $\overline{w} = (w_1, w_2, ..., w_N) \in B$, an event structure can be derived to represent the actions performed by local processors and the causality relation between them. Each letter occurrence in w_i's will be called a event. Let $EVENT(\overline{w})$ denote all events in $\overline{w}$.

Definition 9. Let $a, b \in EVENT(\overline{w})$. $a \leq b$ iff for any $\overline{w}' \in init(\overline{w})$, $b \in EVENT(\overline{w}')$ implies $a \in EVENT(\overline{w}')$.

It follows from Lemma 3 that $a \leq b$ iff the minimal initial part (according to the partial ordering relation generated by the operation min) of $\overline{w}$ containing b also contains a. Thus, the minimal initial part of $\overline{w}$ containing b contains all events which lead to the occurrence of b.

It can be seen easily that if $\overline{w}' \in init(\overline{w})$, then the poset $(EVENT(\overline{w}'), \leq)$ is a left closed subposet of $(EVENT(\overline{w}), \leq)$.

When a process i can perform an action b during an execution of the system resulting $\overline{w}$, what it can learn about the system is each predecessor of b has been occurred. Thus, the amount of global information that the processor i can have at this moment is the minimal initial part of $\overline{w}$ containing b. It follows that we can maintain only the knowledge of the system which is derived from this amount of global information. Denote by $K_i(\nu a)$ the minimal initial part of $\overline{w}$ having νa as its i^{th} component which is the minimal initial part of $\overline{w}$ containing a for an event a performed by the process i (a occurs in $\overline{w}$). When a is a fixed letter occurrence in w_i, we write $K_i(a)$ instead of $K_i(\nu a)$ if it does not cause any confusion.

Proposition 1. *Let $w_i = ua'a\nu$, where a is a receive, and b the corresponding send of a in w_j (i.e. $w_j = xby$ and $\#_{r_{ij}} ua' = \#_{s_{ji}} x$). Then*

$$K_i(a) = max\ (\ K_i(a'), K_j(b)\overline{\lambda}_i(a)$$

where $\overline{\lambda}_i(a)$ denotes the element of A, of which all components is λ except that its i^{th} component is a.

Proof. max $(K_i(a'), K_j(b))$, is an initial part of $\overline{w}$ containing a'. Since b is the corresponding send of a, if $\overline{q}_0$ - max $(K_i(a'), K_j(b)) \rightarrow \overline{q} = (q_1, ..., q_N, \gamma)$ then $\gamma_{ji} = \lambda$ and the message m generated by b is the first element of γ_{ji}. Thus, $\overline{w}' =$ max $(K_i(a'), K_j(b))\overline{\lambda}_i(a)$ is in B, and hence, $\overline{w}'$ must be the minimal initial part of $\overline{w}$ containing a. To see why, we suppose in contrast that $\overline{w}'$ is not so. This means that at least one component of $K_i(a)$ must be a proper prefix of $\overline{w}'$, which is the corresponding component of either $K_i(a')$ or $K_j(b)$. It follows that either min $(K_i(a'), K_i(a))$ is a

proper initial part of $K_i(a')$ containing a' or min $(K_i(a), K_j(b))$ is a proper initial part of $K_j(b)$ containing b. This is a contradiction to the definition of either $K_i(a')$ or $K_j(b)$. The proof is complete.

Proposition 1 is a foundation of the algorithm for maintaining the amount of global information of the system in the processes.

In order to keep our algorithm as general as possible, we introduce a function f for representing the knowledge of global properties that can be derived from $K_i(a)$'s. Let P be a partially ordered lattice, f be a function from $init(\overline{w})$ to P satisfying:

$$f(\max(\overline{w}', \overline{w}'')) = \max(f(\overline{w}'), f(\overline{w}''))$$
$$f(\overline{w}'\overline{\lambda})(a)) = g_i(f(\overline{w}'), a),$$

where g_i is a given function from $init(\overline{w}) \times A_i$ to P, $i = 1, 2, ..., N$.

The operation max represents synthesizing global knowledge of the system of a process, and the function g_i represents the fact that if the process i has the amount of global knowledge K of the system before it performs the action a, then it has the amount of global knowledge $g_i(K, a)$ of the system after it performs the action a.

Let

$\overline{q}_0 - \overline{w}' \rightarrow \overline{q}', \overline{q}_0 - \overline{w}'' \rightarrow \overline{q}'', \overline{q}' = (q_1', q_2', ..., q_N', \gamma'), \overline{q}''$
$= (q_1'', q_2'', ..., q_N'', \gamma''), \overline{w}', \overline{w}'' \in init(\overline{w})$. The following instances of f are of interest.

a).

$$f(\overline{w}') = (|w_1'|, |w_2'|, ..., |w_N'|), \overline{w}' \in init(\overline{w}), \ P = \mathbf{N}^N,$$
$$g_i(j_1, j_2, ...j_i, ...j_N) = (j_1, j_2, ..., j_i + 1, ..., j_n).$$

In this case, $f(\overline{w}')$ is considered as partially ordered logical time of the system after the execution represented by $\overline{w}'$. Then, $f(K_i(a))$ will be a timestamp assigned to a to ensure that $K_i(a) \leq K_j(b)$ iff a is a predecessor of b. An implementation of this can be found in [7].

b).

$$f(\overline{w}') = (|w_1'|, q_1', C_1'), (|w_2'|, q_2', C_2), ..., (|w_N'|, q_N', C_N)), \overline{w}' \in init(\overline{w}),$$

where $C_i' = ((n_{i1}', c_{i2}'), ..., (n_{iN}', c_{iN}'))$, n_{ij} is the number of sends from i to j in w_i', $c_{ij}' = \gamma_{ij}'$ is the queue of messages having not received by j from i after the execution represented by $\overline{w}'$.

$$g_i(((j_1, q_1', C_1'), ..., (j_i, q_i', C_i'), ..., (j_N, q_N', C_N')), a)$$
$$((j_1, q_1', C_1''), ..., (j_i + 1, p_i, C_i''), ..., (j_N, q_N', C_N'')),$$

where

$$C_j'' = ((n_{j1}'', c_{j1}''), ..., (n_{ji}'', c_{ji}''), ..., (n_{jN}'', c_{jN}'')), j \leq N,$$
$$(q_1', ..., q_i', ..., q_N', \gamma') - \overline{\lambda}_i(a) \rightarrow (q_1', ..., p_i, ..., q_N', \beta)), \beta_{ij} = c_{ij}'', i, j \leq N,$$
$$n_{ji}'' = n_{ij}' + 1 \text{ if } a \text{ is } s_{ij} \text{ and } n_{ij}'' = n_{ij}' \text{ otherwise.}$$

$f(\overline{w}')$ is considered as the global state of the system after the execution represented by $\overline{w}'$ which is added with message counters and local time clocks. Thus, $f(K_i(a))$ will give a possible global state when the processor i performs the event a. The operation max in this interpretation is defined as

$$\max(f(\overline{w}'), f(\overline{w}'')) = ((\max(|\overline{w}_1'|, |w_1''|), q_1, C_1),$$
$$(\max(|w_2'|, |w_2''|, q_2, C_2), \ldots, (\max(|w_N'|, |w_N''|), q_N, C_N),$$

where

$$q_i = \begin{cases} q_i' & \text{if } \max(|w_i'|, |w''^N) = |w_i'|, \\ q_i'' & \text{if } \max(|w_i'|, |w''^N) = |w_i''|, \end{cases}$$

if $\max(|w_i'|, |w''^)) = |w_i'|$, $\max(|w_j'|, |w''^)) = |w_j'|$, then

$$n_{ij} = n_{ij}', \quad c_{ij} = c_{ij}',$$

if $\max(|w_i'|, |w''^)) = |w_i'|$, $\max(|w_j'|, |w''^)) = |w_j'|$, then $n_{ij} = n_{ij}'$, c_{ij} is the concatenation of the first $n_{ij}' - n_{ij}''$ elements of c_{ij}' and c_{ij}''.

The remaining cases of c_{ij} are obtained by changing the roles of $\overline{w}'$ and $\overline{w}''$ in the above definition.

Now, we can give an algorithm for maintaining $f(K_i(a))$ in the local state of the process i after it has performed the event a. The algorithm is presented as a distributed system D', which is behaviourally equivalent to D' with $f(K_i(a))$ being kept as a component of the local state of the process i after it performs a.

$$D' = (P_1' P_2', \ldots, P_N', C, \rightarrow' C, M'$$

where

$$P_i' = (Q_i \times f(VAS(A)), \rightarrow i', A_i), M_{ij}' = M_{ij} \times f(VAS(A)),$$

$\rightarrow C'$ and $\rightarrow i'$ is defined as follows.

Let $q, q' \in Q_i, a \in \sum_i, \overline{w}, \overline{w}' \in VAS(A)$.

If $a \in A_i, q - a \rightarrow \overline{q}'$, then $(q, f(\overline{w}) - a \rightarrow i'(q', g_i(f(\overline{w}, a)))$.

If $a = s_{ij}, q - a \rightarrow (q', m)$, then $(q, f(\overline{w})) - a \rightarrow '((q', g_i(f(\overline{w}), a)), (m, g_i(f(\overline{w}), a))$.

If $a = r_{ij}, (m, q) - 'a \rightarrow q'$, then $((m, f(\overline{w}')), (q, f(\overline{w}))) - a \rightarrow' ((q', \max(f(\overline{w}), f(\overline{w}')))$.

Let $q_0' = ((q_1^0, f(\overline{\lambda})), (q_2^0, f(\overline{\lambda})), \ldots, (q_N^0, f(\overline{\lambda})), \Phi)$ be the initial state of D'.

Theorem 2. *D and D' are behaviourally equivalent, i.e. $B = B'$. Furthermore, if $(q, d), q' \in Q_i$, $d \in f(VAS(A))$ is the local state of the process i of D' after it performs the event a in an execution resulting $\overline{w}$, then $d = f(K_i(a))$.*

Proof. Let T and T' be labelled transition systems for D and D' respectively. The proof goes by induction on the length of derivation that

$$\overline{q}_0 - \overline{w} - \overline{q} = (q_1, q_2, \ldots, q_N, \gamma)$$

if and only if

$$\overline{q}_0' - \overline{w} \rightarrow' \overline{q} = ((q_1, f(K_1(\overline{w}_1))), (q_2, f(K_2(\overline{w}_2))), \ldots, (q_N, f(K_N(\overline{w}_N))), \gamma'),$$

using Proposition 1 and the property of the function f, where $\gamma_{ij} = h(\gamma_{ij}')$ with h being the extended homomorphism of the projection on the first component.

By choosing a suitable interpretations of f, D' can be used in debugging distributed programs as in [14,15], studying invariant properties and reasoning of the systems [8], distributed snapshots [4], etc. With f being the global state function, a global state of D can be determined in only one local process. If the determined global state is too old to be considered, some of local states of some another processes may be accessed as well. Some of these uses of the algorithm will be presented in separate papers.

Example 2. Let $D, \overline{w}$ be as in Example 1, f and $g_i, i = 1,2$ be defined in b). Since only the process 2 sends messages to process 1, a channel state γ can be written as a queue of messages from the process 2 to the process 1 (matrix γ contains one element, and hence, $C_i, i = 1,2$ contains one element). Then, the derivation represented by $\overline{w}$ in D is as follows.

$$((q_0,((0,q_0,(0,\lambda)),(0,p_0,(0,\lambda))),(p_0,((0,q_0,(0,\lambda)),(0,p_0,(0,\lambda))),\lambda) - (\lambda,b) \rightarrow$$

$$((q_0,((0,q_0,(0,\lambda)),(0,p_0,(0,\lambda))),(p_1,((0,q_0,(0,\lambda)),(1,p_1,(0,\lambda))),\lambda) - (\lambda,s_{21}) \rightarrow$$

$$((q_0,((0,q_0,(0,\lambda)),(0,p_0,(0,\lambda))),(p_0,((0,q_0,(0,\lambda)),(2,p_0,(1,1))),$$

$$(1,((0,q_0,(0,\lambda)),(2,p_0,(1,1)))) - (r_{12},b) \rightarrow$$

$$((q_1,((1,q_1,(0,\lambda)),(2,p_0,(1,\lambda))),(p_0,((0,q_0,(0,\lambda)),(3,p_1,(1,1))),\lambda) - (a,s_{21}) \rightarrow$$

$$((q_0,((2,q_0,(0,\lambda)),(2,p_0,(1,\lambda))),(p_1,((0,q_0,(0,\lambda)),(4,p_0,(2,11))),$$

$$(1,((0,q_0,(0,\lambda)),(4,p_1,(2,11)))) - (r_{12},s_{12}) \rightarrow$$

$$((q_1,((3,q_1,(0,\lambda)),(4,p_1,(2,\lambda))),(p_2,((0,q_0,(0,\lambda)),(5,p_2,(3,110))),$$

$$(0,((0,q_0,(0,\lambda)),(5,p_2,(3,110))))).$$

From the state $((q_1,((3,q_1,(0,\lambda)),(4,p_1,(2,\lambda)))$ of the process 1, we know that one global state of D is (q_1,p_1,λ).

Conclusion.

The vectors of sequential words generated by distributed labelled transition systems are an intuitive model for describing the behaviour of concurrent and distributed systems. The notions of causality relation, knowledge of global information of processes, logical time are formulated accurately in term of this model of behavior. It has been proved to be a good model for proving the equivalence and verification of distributed systems. Thus, an extension of the model to the case of systems of the processes created dynamically and an algebra of processes in this model should be studied in support of design and understanding of concurrent and distributed systems.

References

1. America H.M. & and Rutten J.J.M.M., A Parallel Object-Oriented Language: Design and Semantic Foundations, Languages for Parallel Architectures, J.W. Bakker, Ed., Wiley Series in Parallel Computing, 1989, 1-50.
2. Bal H.E ; Steiner J.G. & Tanebaum A.S., *Programming Languages for Distributed Computing Systems, ACM Computing Surveys*, Vol. **21**, (1989), 261-322.
3. Casavant T.L. & Kuhl J.G., *A Communicating Finite Automata Approach to Modeling Distributed Computation and Its Application to Distributed Decision-Making*, IEEE Transactions on Computers, Vol. **39**, (1990), 628-639.
4. Chandy K.M. & Lamport L., *Distributed Snapshots: Determining the Global State of Distributed Systems*, Ann. Rev. Computer Science, Vol. **2**, (1987), 37-68.
5. Degano P. & Montanari U., Distributed Systems, Partial Orderings of Events, and Event Structures, Control Flow and Data Flow: Concepts of Distributed Programming, M. Broy, ed., NATO ASI Series, Vol. **F14**, 1985, 7-106.
6. Degano P., Gorrieri R. & Marchetti S., *An Exercise in Concurrency: A CSP Process as a Condition/Event System*, Lecture Notes in Computer Science, Vol. ?, 1989, 85-105.
7. Fridge C., *Logical Time in Distributed Computing Systems*, Computer, Vol. **24**, (1991), 28-33.
8. Halpern J.I, *Using Reasoning about Knowledge to Analyze Distributed Systems*, Ann. Rev. Computer Sci.II **2**, (1987), 37-68.
9. Hoare C.A.R., Communicating Sequential Processes, CACM, 21, 1978, 666-677.
10. Hung D.V., *A Model for Analyzing Distributed Systems*, Journal of Informatics and Cybernetics, Vol. **1**, No. 2, 1991, 15-23 (in Vietnamese).
11. Hung D.V., *Notes on Projection Products, Trace Languages and Synthesized Concurrent Computation Systems*; MTA-SZTAKI Kozlemenyek, Vol. **32**, (1985), 87-104.
12. Hung D.V. & Knuth E., *Semi-Commutations and Petri Nets*, Theoretical Computer Science, n^0. 64, (1989), 67-81.
13. Hung D.V. & Knuth E., *A Non-interleaving Semantics for Communicating Sequential Processes: A fixed point approach*, Acta Cybernetica, Szeged, Hungary, Tom. **8**, f.3, (1988), 293-311.
14. Joseph T.A., Rauchle T. & Toueg S., *State Machines and Assertions: Integrated Approach to Modelling and Verification of Distributed Systems*, Science of Computer Programming, Vol. **7**, (1987), 22.
15. Keller R., *Formal Verification of Parallel Programs*, CACM, **19**, 1976, 561-572.
16. Mazurkiewicz A. Ochmanski E. & Peuczek W., *Concurrent Systems and Inevitability*, Theoretical Computer Science, n^0. **64**, (1989), 281-304.
17. De Nicola R., *Extensional Equivalences for Transition Systems*, Acta Informatica, **24**, (1987), 211-237.
18. Nielsen M., Plotkin G. & Winskel G., *Petri Nets, Event Structures and Domains*, Theoretical Computer Science, **13**, (1981), 85-108.

19. Shield M.W., Nonsequential Behaviour, Part I, Tech. Rept.CSR-120-82 Dept. of Computer Science, University of Edinburgh, 1982.

Abstract

Labelled Transition System Approach to the Distributed Computing Systems

The present paper introduces the notion of distributed transition systems for modelling, designing and understanding distributed computing systems. The concurrency can be expressed explicitly in the model. Some of the global properties of the systems are discussed and determined. It is shown in the paper that by keeping knowledge of other processes in each process of a sysrem, some of its global properties can be synthesized from only few local process states.