Inevitable Global States: a Concept to Detect Unstable Properties of Distributed Computations in an Observer Independent Way

Eddy Fromentin, Michel Raynal

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Inevitable Global States: a Concept to Detect Properties of Distributed Computations in an Observer Independent Way

Eddy Fromentin and Michel Raynal

N° 2317

Aout 1994

PROGRAMME 1

ISSN 0249-6399

Inevitable Global States: a Concept to Detect Properties of Distributed Computations in an Observer Independent Way

Eddy Fromentin* and Michel Raynal*

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet Adp

Rapport de recherche n ° 2317 — Aout 1994 — 18 pages

Abstract: When analyzing, testing or debugging a distributed program, an important question one has to answer is: "Does this computation satisfy a given property ?". We are interested in this paper in answering such a question when the property is formulated as a general predicate on a global state of the computation, and more specifically when the property is unstable (i.e. once true the associated predicate is not guaranteed to remain true forever).

Notions such as abstraction level with respect to a predicate (user's level) and weak precedence between local states are first introduced. Then an abstraction called *inevitable global state* is defined, and a necessary and sufficient condition to detect such states is provided. With this abstraction a precise meaning is given to the previous question (independently of any particular perception one can have of the distributed computation). A detection algorithm for this question is finally presented.

Key-words: distributed computation, consistent global state, causality, concurrency, inevitable global state, unstable property, observation, observer-independent property.

(Résumé : tsvp)

Also in the proceedings of the 6^{th} IEEE Symposium on Parallel and Distributed Processing, *Inevitable global states : a Concept to Detect Unstable Properties of Distributed Computations in an Observer Independant Way*, Dallas, October 1994.

*IRISA – Campus de Beaulieu, 35042 RENNES cedex – FRANCE, e-mail:{fromenti, raynal}@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 84 71 71

États globaux inévitables : un concept pour la détection de propriétés des calculs répartis

Résumé :

Lors de l'analyse, du test ou du déverminage d'un programme réparti, il est important de pouvoir répondre à une interrogation du type : "Cette exécution satisfait-elle telle propriété ?". Ce rapport propose une façon de répondre à cette question quand la propriété en question, formulée commme un prédicat sur un état global de l'exécution, est instable (i.e. une propriété instable peut fugitivement être vraie lors de l'exécution).

Pour cela, nous introduisons des notions telles que niveau d'abstraction en regard d'un predicat donné (niveau de l'utilisateur) et précédence faible entre états locaux. Nous établissons ensuite un mécanisme d'abstraction que nous appellons *état global inévitable* ainsi qu'une condition nécessaire et suffisante en vue de la détection de tels états. Une signification précise à la question précédente est alors donnée, indépendemment de toute observation du calcul réparti. Un algorithme efficace pour la détection de ces propriétés est donné.

Mots-clé : calcul réparti, états globaux cohérents, causalité, concurrence, états globaux inévitables, propriétés instables, observation

1 Introduction

Properties of distributed computations can be expressed as "paths" their control flows have to follow, or as general predicates some of their global states have to verify, in order for the property to be satisfied. We adressed the first approach in [9, 13], and are here interested in the second one. Ideally a distributed computation $\widehat{R}$ satisfies a property Φ if it passes through a global state satisfying the predicate associated with Φ. Unfortunately, because of the uncertainty characterizing distributed computations, (due principally to the asynchrony of channels), we are incapable of determining the global states through which the computation actually passed.

If the property we want to detect is stable (once true in a global state the associated predicate remains true in all future global states) the previous remark is not a drawback. The Chandy-Lamport's snapshot algorithm [3] can be superimposed on the underlying computation to compute global states until the predicate is verified by a global state. If the property is unstable, the problem becomes more complicated as a snapshot algorithm cannot be used, because it can deliver a sequence of global states that has "gaps" corresponding exactly to those global states in which the unstable property holds. Cooper and Marzullo adressed this problem in [4] and provided two meanings to the sentence "Does a computation $\widehat{R}$ satisfy property Φ ?". They considered the set of all global states which could be generated by a distributed computation and the set of all possible sequential executions that can be associated with this computation. These executions –usually called observations– structure the set of all global states as a lattice. A distributed computation $\widehat{R}$ satisfies $POS\ \Phi$ (denoted $\widehat{R} \models POS\ \Phi$), if and only if the lattice has a global state Σ verifying Φ, whereas $DEF\ \Phi$ is satisfied by $\widehat{R}$ (denoted $\widehat{R} \models DEF\ \Phi$) if and only if each observation passes through a global state verifying Φ[1].

The Cooper-Marzullo's approach gives two clear answers to the initial question in the case of unstable properties. At the operationnal level, detection of $POS\ \Phi$ or $DEF\ \Phi$ requires first to build the lattice of global states and then to traverse it to look for one global state (in the case of POS) or one global state per observation (in the case of DEF) satisfying Φ. As, in the worst case, the size of the lattice can be exponential in the number of processes [1, 18], this approach can become unfeasible.

This paper presents another way to answer the question: "Does a computation $\widehat{R}$ satisfies Φ ?", which does not suffer from the previous limitations. The meaning of the question is based on a new and simple abstraction, termed *inevitable global state*. Basically a global state is inevitable if it is shared by all the observations of

[1] $POS\ \Phi$ stands for Possibly Φ; $DEF\ \Phi$ for Definitely Φ.

the computation, i.e. if it is observation-independent. A distributed computation $\widehat{R}$ satisfies the property formulated by $PROP\ \Phi$ (denoted $\widehat{R} \models PROP\ \Phi$)[2] if and only if there exists an inevitable global state Σ verifying Φ.

The paper has three main Sections. Section 2 presents the distributed computation model, notions such as user's level, weak precedence between local states, and the lattice associated with each distributed execution. Section 3 defines inevitable global states and states a necessary and sufficient condition for a global state to be inevitable. Section 4 presents a monitor-based algorithm to detect such states and to answer the question "$\widehat{R} \models PROP\ \Phi$?". The conclusion lists other problems that can be solved by using the inevitable global state abstraction.

This work is part of our current effort in designing and implementing a debugging facility for distributed programs [2, 12].

2 Distributed Computations and their Observations

2.1 Distributed programs

A distributed program is made of n sequential processes $P_1 \cdots P_n$ which communicate and synchronize by the only means of message passing. Such programs are executed by an underlying system composed (without loss of generality) of n processors (one per process) that have local memories and can exchange messages through reliable, non necessarily FIFO channels. Transmission delays are finite but unpredictable.

2.2 Basic events and Lamport's level

Execution of a distributed program produces a set H of *basic* events composed of communication events (send and receive) and internal events. These events are structured by Lamport's *happened before* relation, denoted $\stackrel{e}{\rightarrow}$ and sometimes called *causal precedence* [14].

Let $e_i^0 e_i^1 e_i^2 \cdots$ be the sequence of basic events produced by process P_i. Relation $\stackrel{e}{\rightarrow}$ is defined by:

[2] $PROP\ \Phi$ stands for Properly Φ (meaning in a proper or observation-independent manner).

$$e_i^x \xrightarrow{e} e_j^y \Leftrightarrow \begin{cases} i = j \text{ and } x+1 = y \\ \textit{or} \\ e_i^x \text{ is the sending of a message} \\ \quad \text{and } e_i^y \text{ its reception} \\ \textit{or} \\ \exists e_k^z \text{ such that } e_i^x \xrightarrow{e} e_k^z \textit{ and } e_k^z \xrightarrow{e} e_j^y \end{cases}$$

A distributed computation can be represented by a partially ordered set (poset) $\widehat{H} = (H, \xrightarrow{e})$. This poset defines the computation at some abstraction level; we call it *Lamport's level*. Figure 1 displays a distributed computation at Lamport's level: $\widehat{H} = (H, \xrightarrow{e})$. On process lines, from left to right, events are denoted by circles (white or black). An arrow from one process line to another represents a message; ends of the arrow are sending and receiving events; a message creates causal dependences on events belonging to several processes. It is important to note that this abstraction level includes all communication events.

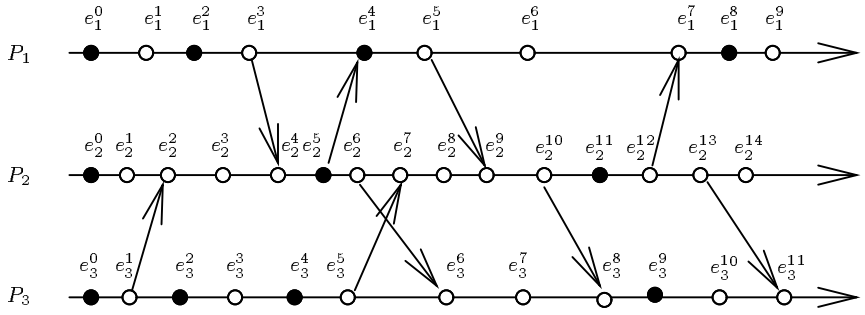


Figure 1: A distributed execution at Lamport's level

2.3 Relevant events and predicate (or user's) level

According to the problem he wants to solve (namely detection of some global predicate on processes local variables) only a subset of basic events (called *relevant events* [15]) are meaningful to the user (these events are the updates of variables used in the predicate; these variables are called *relevant variables*). Let R be the set of relevant events. The poset $\widehat{R} = (R, \xrightarrow{e})$ defines the computation at some abstraction level; we call it *predicate* or *user's level*; at this abstraction level only relevant

variables can be observed. Note that, due to transitivity of $\stackrel{e}{\rightarrow}$ on basic events, $\widehat{R}$ inherits causal precedence induced by communication events, even if communication events are not relevant.

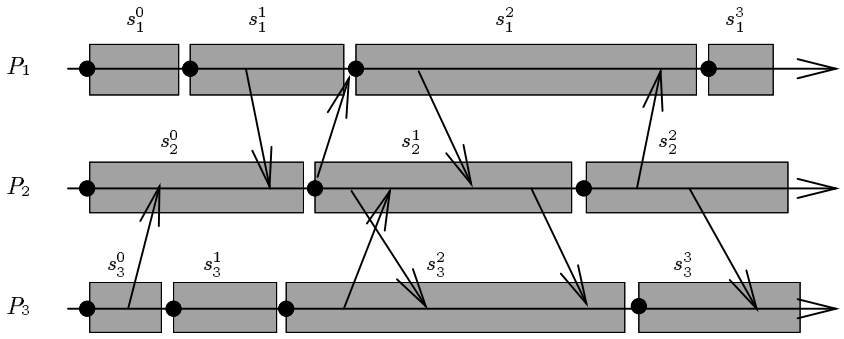


Figure 2: Local states of a distributed execution at user's level

Figure 2 considers the distributed computation exhibited in Figure 1 at some predicate (or user's level): $\widehat{R} = (R, \stackrel{e}{\rightarrow})$. Only relevant events, composing R, are represented (they are the black events of Figure 1). Arrows show causal dependences ($\stackrel{e}{\rightarrow}$) that will be used to define weak and strong precedence relations on local states.

2.4 Precedences between local states

Let r_i^x be the x^{th} relevant event produced by P_i; it provoked a local state change from s_i^{x-1} to $s_i^x = next(s_i^{x-1})$ (let s_i^0 be the initial local state of P_i). Contrary to events that are usually considered instantaneous, local states have a duration: s_i^x lasts from r_i^x (included) till r_i^{x+1} (not included). If P_i terminates s_i^{last} will denote its last local state.

As an example, consider a process P_i with two local variables X_1 and X_2 and only one of them, say X_1, is relevant i.e. appears in the global predicate we are interested in. Updates to X_1 are the only relevant events of P_i. At this abstraction level a local state of P_i is composed of the values of X_1 plus a value (timestamp) measuring P_i's progress at the time it modified X_1[3]. As we can see a local state

[3] When it produces r_i, progress of P_i from the distributed execution point of view, can be represented by the causal past of r_i, namely the couple: $(\{r \mid r : relevant : r \stackrel{e}{\rightarrow} r_i\} \cup \{r_i\}, \stackrel{e}{\rightarrow})$. The measure of such a progress is classicaly implemented with vector clocks (see [6, 16] and Section 3.3).

s_i^x does not change from r_i^x till r_i^{x+1} (independently of the basic events executed between r_i^x and r_i^{x+1} as these events are irrelevant).

Two precedence relations can be defined on local states:

- weak precedence: s_i^x weakly precedes s_j^y if s_i^x began (with respect to causal precedence) before s_j^y. Formally, $s_i^x \xrightarrow{w} s_j^y \Leftrightarrow r_i^x \xrightarrow{e} r_j^y$;
- strong precedence: s_i^x strongly precedes s_j^y if s_i^x was finished (with respect to causal precedence) when s_j^y began to exist. Formally, $s_i^x \xrightarrow{s} s_j^y \Leftrightarrow (r_i^{x+1} \xrightarrow{e} r_j^y$ *or* $s_j^y = next(s_i^x))$.

When considering Figure 2 we have:

$$s_1^0 \xrightarrow{s} s_2^1$$
$$s_1^1 \xrightarrow{w} s_2^1$$
$$\neg(s_1^1 \xrightarrow{s} s_2^1)$$

2.5 The lattice of consistent global states

A global state Σ is composed of n local states s_i, one from each process. Two local states s_i and s_j are mutually consistent (or independent), denoted $s_i \parallel s_j$, if $\neg(s_i \xrightarrow{s} s_j)$ *and* $\neg(s_j \xrightarrow{s} s_i)$. Intuitively they could have co-existed during the computation. A *consistent global state* $\Sigma = (s_1, \cdots, s_i, \cdots, s_n)$ is such that $\forall i \neq j : s_i \parallel s_j$; it could have been passed through by the distributed computation.

The set of all consistent global states that can be associated with a distributed computation $\widehat{R} = (R, \xrightarrow{e})$ has a lattice structure whose minimal (respt. maximal) element is the initial (respt. final) global state $\Sigma^0 = (s_1^0, \cdots, s_i^0, \cdots, s_n^0)$ (respt. $\Sigma^{last} = (s_1^{last}, \cdots, s_i^{last}, \cdots, s_n^{last})$) [1, 2, 4, 18]. There is an edge in the lattice from $\Sigma = (s_1, \cdots, s_i, \cdots, s_n)$ to $\Sigma' = (s_1, \cdots, next(s_i), \cdots, s_n)$ if and only if there is an event r_i of P_i that can be produced in global state Σ; r_i makes P_i progress from s_i to $next(s_i)$ and constitutes the label of this edge.

Figure 3 displays the lattice of consistent global states associated with $\widehat{R}$, the distributed computation of Figure 2. $\Sigma^0 = (s_1^0, s_2^0, s_3^0)$ and $\Sigma^{last} = (s_1^3, s_2^2, s_3^3)$.

A sequential observation O of the distributed computation $\widehat{R}$ represents a consistent view an external sequential observer could have. It is a sequence: $\Sigma^0 r^1 \Sigma^1 r^2 \Sigma^2 \cdots \Sigma^{i-1} r^i \Sigma^i r^{i+1} \cdots \Sigma^{last}$ of consistent global states and relevant events such that:

- all relevant events appear in an order consistent with $\widehat{R}$ (i.e. they form a linear extension of $\widehat{R}$);

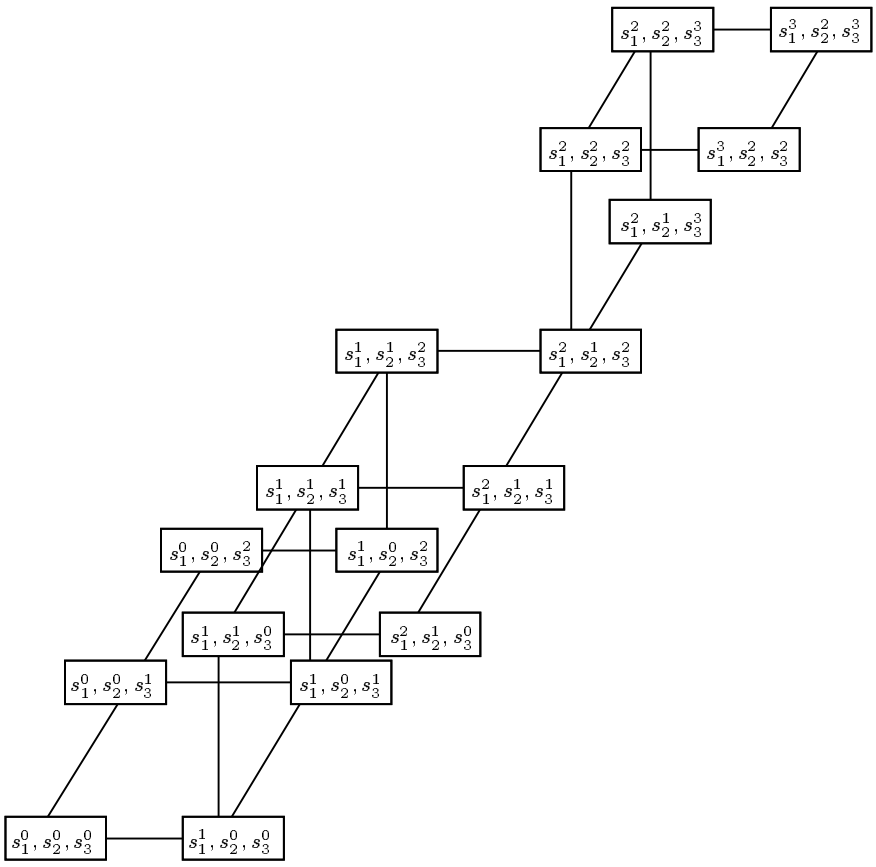


Figure 3: Lattice associated with the distributed computation of Figure 2

- Σ^i is the global state obtained from Σ^{i-1} by executing r^i.

As shown in [1, 18] all the sequential observations correspond exactly to all the paths of the lattice. In fact this lattice represents all the possible sequential executions of the distributed computation $\widehat{R}$.

A non-sequential observation [10] allows an observer to "see" several independent relevant events at the same time. Formally, it is a sequence of global states and sets of relevant events $\Sigma^0 sr^1 \Sigma^1 sr^2 \Sigma^2 \cdots \Sigma^{i-1} sr^i \Sigma^i \cdots \Sigma^{last}$ such that:

- sr^i is a non-empty set of independent events that do not precede the events of sr^k, for all $k < i$ (in terms of order theory sr is an antichain of $\widehat{R}$).
- Σ^i is the global state obtained from Σ^{i-1} by executing all events of sr^i.

2.6 The Cooper-Marzullo's approach

Cooper and Marzullo based on sequential observations the satisfaction, by a distributed computation $\widehat{R}$, of an unstable property Φ expressed as a predicate on a global state[4]. They defined two satisfaction rules. First $\widehat{R} \models POS\ \Phi$ if there exists a sequential observation O including a global state Σ such that $\Sigma \models \Phi$. Second $\widehat{R} \models DEF\ \Phi$ if, in each sequential observation O_x, there exists a global state Σ_x such that $\Sigma_x \models \Phi$.

As noted in the introduction such an approach has feasibility limitations as it requires to build the lattice (which can include $\Omega(k^n)$ global states, k being the maximum number of relevant events produced by a process).

The answer to $POS\ \Phi$ does not depend on the way an observation is modeled (the answer is positive if there is a state in the lattice verifying Φ). That is not the case for $DEF\ \Phi$. For two distinct observations, O_x and O_y, the global states satisfying Φ can be distinct. So if $\widehat{R} \models DEF\ \Phi$ we have no information about the states in which Φ is verified (this can be an inconvenience if the detection is done for debugging purposes, as in this case the user wants to know the global states in which Φ was verified). The next Section presents a new characterization, for a property to be satisfied, that does not have these drawbacks.

3 Inevitable Global States

3.1 Definitions

A global state Σ is inevitable if it belongs to all the observations of the distributed computation $\widehat{R}$.[5] In Figure 3 (s_1^2, s_1^1, s_3^2) is inevitable.

Relevant events and inevitable global states are the right abstractions to solve problems involving global states shared by all observations. Inevitability characterizes the greatest set of global states seen by all observations. Inevitable global states allow to state an interesting observation-independent definition for satisfaction of a

[4] Such as, for example, $x_1 + \cdots + x_n < k$, where x_i belongs to P_i.

[5] With the initial and final global states, all articulation points or vertex separators [5] of the lattice are inevitable global states.

property by a computation $\widehat{R}$. *Proper satisfaction* is defined as Φ being true at the same "point" for all observers:

$$(\widehat{R} \models PROP\ \Phi) \stackrel{def}{\equiv} (\exists \Sigma : \Sigma\ is\ inevitable \wedge \Sigma \models \Phi)$$

Both $POS\ \Phi$ and $PROP\ \Phi$ are observation-independent, but, as we will see, detection of $PROP\ \Phi$ can be done efficiently, without building the lattice.

3.2 A necessary and sufficient condition

Let $\Sigma = (s_1, \cdots, s_n)$ be a global state (distinct from Σ^0 and Σ^{last}).

Theorem IGS:
$\Sigma\ inevitable \Leftrightarrow \forall i, j : (s_i \stackrel{w}{\rightarrow} next(s_j)\ or\ s_j = s_j^{last})$

Proof:

This proof considers only sequential observations and supposes processes have infinite behaviors. It can easily be extended to non-sequential observations and finite behaviors of processes.

$\Rightarrow$

Let $\Sigma = (s_1, \cdots, s_i, \cdots, s_n)$ be an inevitable global state, r_i be P_i's event that produced s_i, and r'_j be P_j's event that produced $next(s_j)$.
Consider $\Sigma' = (\cdots, next(s_j), \cdots)$. Any sequential observation contains Σ (by hypothesis) and such a Σ' (because each observation includes all events and so r'_j) and Σ appears before Σ' (as s_j is produced by P_j before $next(s_j)$); consequently r_i appears before r'_j within this observation. Moreover this is true for any couple (i, j).
As Σ is inevitable the previous remark is true for all observations. It follows that for all observations: $\forall i, j$: r_i appears before r'_j.
From Szpilrajn's theorem [19] (which states that the intersection of all linear extensions –here the set of all observations– of a partial order –here $\widehat{R} = (R, \stackrel{e}{\rightarrow})$– is precisely this partial order) it follows that $\forall i, j : r_i \stackrel{e}{\rightarrow} r'_j$; that is to say in terms of local states (Section 2.4) $\forall i, j : s_i \stackrel{w}{\rightarrow} next(s_j)$.

$\Leftarrow$

Let a global state $\Sigma = (s_1, s_2, \cdots, s_n)$ be such that $\forall i, j : s_i \stackrel{w}{\rightarrow} next(s_j)$. First let us show that Σ is a consistent global state. Suppose that Σ is not consistent. Then $\exists i, j : s_i \stackrel{s}{\rightarrow} s_j$ (See Section 2.5 for the definition of a consistent global state); from which we can conclude (see Section 2.4) $\exists i, j : next(s_i) \stackrel{w}{\rightarrow} s_j$ which is a contradiction.

Now consider a sequential observation O that comprises two global states $\Sigma 1(i) = (\cdots, s_i, \cdots)$ and $\Sigma 2(j) = (\cdots, next(s_j), \cdots)$. We have to show that $\Sigma 1(i)$ precedes $\Sigma 2(j)$.

Let r_i' be P_i's event that produced s_i, r_i'' be P_i's event that produced $next(s_i)$ and r_j'' be P_j's event that produced $next(s_j)$. As O includes all events it includes r_i', r_i'' and r_j'' it follows that $\Sigma 1(i)$ and $\Sigma 2(j)$ exist.

$\Sigma 1(i)$ appears in O after r_i' and before r_i''; $\Sigma 2(j)$ appears after r_j''. Moreover $r_i' \stackrel{e}{\rightarrow} r_j''$ (because $s_i \stackrel{w}{\rightarrow} next(s_j)$); it follows that $\Sigma 1(i)$ precedes $\Sigma 2(j)$ in O.

As by hypothesis $\forall i, j : s_i \stackrel{w}{\rightarrow} next(s_j)$ we have $\forall i, j : r_i' \stackrel{e}{\rightarrow} r_j''$; consequently any sequential observation O is such that:

$$\begin{aligned} O = \quad & \Sigma^0 \cdots r'_{i_1} \Sigma 1(i_1) \cdots r'_{i_x} \Sigma 1(i_x) \cdots \\ & \cdots r'_{i_n} \Psi r''_{j_1} \Sigma 2(j_1) \cdots \\ & \cdots r''_{j_x} \Sigma 2(j_x) \cdots r''_{j_n} \Sigma 2(j_n) \cdots \end{aligned}$$

with n-uples $(i_1, \cdots, i_n)$ and $(j_1, \cdots, j_n)$ being permutations of $(1, \cdots, n)$. It follows that the global state Ψ is $\Sigma = (s_1, \cdots, s_i, \cdots, s_n)$.

□

3.3 Detecting weak precedence

The previous characterization (theorem **IGS**) of inevitable global states is purely theoretical. In order to use it in detection algorithms we need an equivalent operational characterization. This Section presents an appropriate vector clock mechanism which associates a timestamp with each local state; these timestamps allow to decide whether two local states are related by $\stackrel{w}{\rightarrow}$ or not.

Introduced simultaneously by Fidge [6] and Mattern [16], vector clocks constitute an operational tool to encode dependency and concurrency of events of a distributed computation. We associate here vector clock timestamps with local states in the following way:

- $v_i[1 \cdots n]$ is the vector clock of process P_i; it is initialized to $\mathbb{1}_i$ (a zero vector with 1 in the i^{th} position);
- each time P_i enters a new local state (execution of a relevant event) $v_i[i]$ is incremented by a positive value (e.g. 1) and the new value of the v_i constitutes the timestamp of the local state;
- all messages carry the current value of the vector clock of their senders;
- when a message m, carrying a timestamp $v(m)$, is delivered to P_i, v_i is updated to $max(v_i, v(m))$[6].

Let $v(s_i)$ and $v(s_j)$ be the timestamps associated respectively with local states s_i and s_j. From results of [6, 16] on the timestamping of events we can deduce the following formula (see the proof in the Appendix):

(F):
$s_i \xrightarrow{w} s_j \Leftrightarrow v(s_i)[i] \leq v(s_j)[i]$

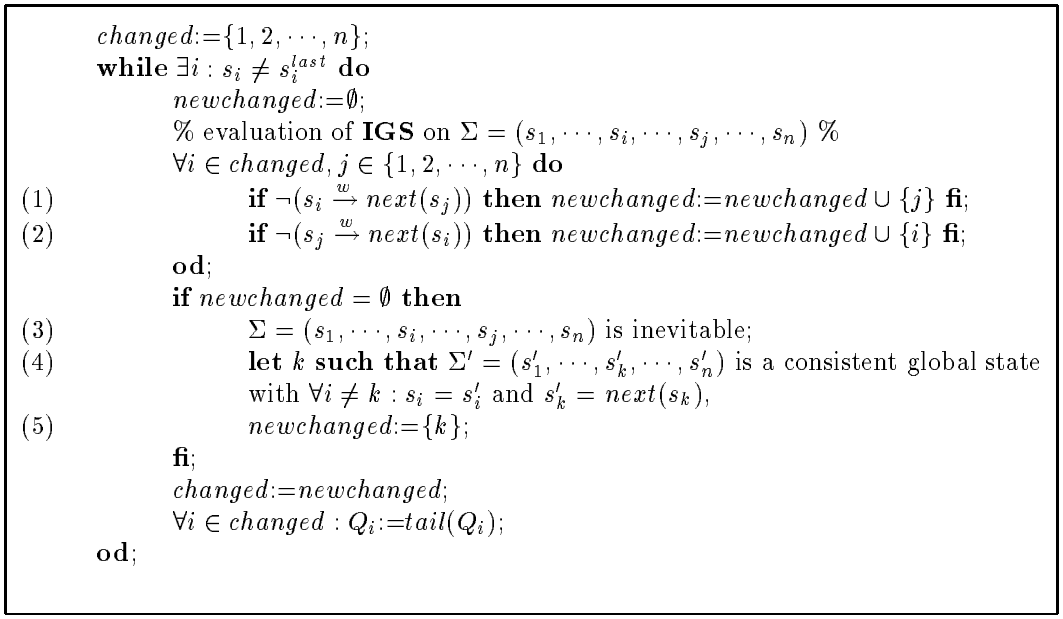
(Other relations and formulas concerning local states can be found in [7]; among them: $s_i \xrightarrow{s} s_j \Leftrightarrow v(s_i)[i] < v(s_j)[i]$).

4 A detection algorithm for $PROP\ \Phi$

Thanks to the formula (**F**) introduced in Section 3.3, it is easy to design an algorithm that detects inevitable global states. A FIFO channel is added between each process and a monitor M. Each time a new local state begins, P_i sends to M a control message composed of the local state and its timestamp. The monitor is equipped with n queues Q_i which store incoming messages from each process P_i. M uses **IGS** to detect inevitable global states.

The protocol executed by the monitor is an adaptation of an algorithm defined by Garg [11] to detect a largest anti-chain (here a n–uple of local states satisfying condition **IGS**) in a partially ordered set given its decomposition into its chains (here the sequences of control messages received from each P_i and stored in queues Q_i).

[6] $v_i := max(v_i, v(m)) \Leftrightarrow \forall k \in 1 \cdots n : v_i[k] := max(v_i[k], v(m)[k])$.



$changed$:=$\{1, 2, \cdots, n\}$;
while $\exists i : s_i \neq s_i^{last}$ **do**
  $newchanged$:=$\emptyset$;
  % evaluation of **IGS** on $\Sigma = (s_1, \cdots, s_i, \cdots, s_j, \cdots, s_n)$ %
  $\forall i \in changed, j \in \{1, 2, \cdots, n\}$ **do**
(1) **if** $\neg(s_i \xrightarrow{w} next(s_j))$ **then** $newchanged$:=$newchanged \cup \{j\}$ **fi**;
(2) **if** $\neg(s_j \xrightarrow{w} next(s_i))$ **then** $newchanged$:=$newchanged \cup \{i\}$ **fi**;
  od;
  if $newchanged = \emptyset$ **then**
(3) $\Sigma = (s_1, \cdots, s_i, \cdots, s_j, \cdots, s_n)$ is inevitable;
(4) **let** k **such that** $\Sigma' = (s'_1, \cdots, s'_k, \cdots, s'_n)$ is a consistent global state
    with $\forall i \neq k : s_i = s'_i$ and $s'_k = next(s_k)$,
(5) $newchanged$:=$\{k\}$;
  fi;
  $changed$:=$newchanged$;
  $\forall i \in changed : Q_i$:=$tail(Q_i)$;
od;

Figure 4: The algorithm to detect $PROP\ \Phi$

4.1 Underlying principles

In order to detect inevitable global states, Garg's algorithm is adapted in the following way. Q_i is the sequence of timestamped local states received in order from P_i; $head(Q_i)$ denotes the first local state of Q_i; $tail(Q_i)$ denotes the sequence Q_i without its first element; $next^*(s_i)$ denotes any successor of s_i including s_i itself.

Tests to decide whether two local states are related by a precedence relation are done on their timestamps, thanks to formula (**F**) introduced in Section 3.3. To make the algorithm easier to understand we suppose the queues Q_i have been filled up by processes. This version can easily be adapted to work on the fly, with processes filling their queues as they progress.

In Garg's algorithm heads of the queues are checked to see if they form a consistent global state (a largest antichain). Its adaptation to detect inevitable global states (that are always consistent, see Section 3.2) is based on the two following observations:

1. Let $\Sigma = (\cdots, s_i, \cdots, s_j, \cdots)$ be the global state under consideration candidate to inevitability. if $\neg(s_i \xrightarrow{w} next(s_j))$ we can conclude any global state $\Sigma' = (\cdots, next^*(s_i), \cdots, s_j, \cdots)$ is not inevitable. So in that case s_j is no longer considered and the algorithm considers the global state $\Sigma'' = (\cdots, s_i, \cdots, next(s_j), \cdots)$ as a candidate for inevitability. This observation allows to redefine appropriately the head of the queues (with the auxiliary variable *changed* in the algorithm).

2. After an inevitable global state $\Sigma = (s_1, s_2, \cdots, s_n)$ has been found, the next candidate Σ' for inevitability is defined in the following way (in order not to miss inevitable global states): Σ' is a consistent global state $(s'_1, s'_2, \cdots, s'_n)$ that is an immediate successor of Σ in the lattice, i.e.: $\exists k : (\forall i \neq k : s'_i = s_i \ and \ s'_k = next(s_k))$.

4.2 The algorithm

The algorithm is described in Figure 4. For any queue Q_i, s_i (respt. $next(s_i)$) is a synonymous of $head(Q_i)$ (respt. $head(tail(Q_i))$). Moreover to simplify the description of the algorithm we suppose $next(s_i^{last}) = s_i^{last}$.

4.3 Properties

The safety property indicates consistency of the detection: if the algorithm claims Σ inevitable then it is. The liveness property states that if a global state is inevitable then the algorithm will detect it. Proofs of these properties can be found in [8].

4.4 Time complexity

Let k_i be the number of local states (including s_i^0 and s_i^{last}) of process P_i and $K = max_i(k_i)$.

If we eliminate the statement **if** $newchanged = \emptyset$ **then** $\cdots$ **fi** we obtain an algorithm whose structure is the same as Garg's one. Garg showed, in [11], that the time complexity of this algorithm is $O(n^2K)$ comparisons. Each comparison is here on 2 integers.

Consider now the algorithm without the loop including lines 1 and 2. To advance the appropriate queue Q_k the algorithm has to find a consistent global state Σ' immediate successor of Σ. Obtaining such a global state $\Sigma' = (s_1, \cdots, next(s_k), \cdots, s_n)$ requires at most $O(n^2)$ comparisons of integers ($2(n-1)$ comparisons to test $next(s_k) \parallel s_i$ for $i \neq k$ and, at worst, n such sets of comparisons to find the appropriate k). Such tests are done each time an inevitable global state is found. $k_1 + k_2 + \cdots + k_n$ constitutes an upper bound on the number of inevitable global states (all elements of queues are examined without never backtracking). So the second part of the algorithm is upper bounded by $0(n^2 \Sigma k_i)$.

Consequently $0(n^3K)$ constitutes an upper bound on the number of comparisons of integers needed by the algorithm.

5 Conclusion

This paper presented the concept of *inevitability* for global states of a distributed computation $\widehat{R}$. As seen by all possible observations of $\widehat{R}$, an inevitable global state is observation-independent. Inevitability has be characterized by a necessary and sufficient condition; moreover deciding at run-time about inevitability can easily be done by associating vector clock timestamps with local states. Inevitability characterizes the maximal set of global states for which all possible observers agree they have been passed through by the distributed execution.

Inevitability has been used to give a precise meaning to the question "Does this computation satisfy property Φ ?". The interest of this meaning is in its being

observer-independent. Moreover an efficient algorithm to detect inevitable global states has be presented.

Inevitable global states can be used to solve other problems. Among them there are the definition of particular checkpoints [20] (as contrary to global states defined by a snapshot algorithm [3], an inevitable global state has necessarily been passed through by the actual execution) and the definition of concurrency measures [17] (as an inevitable global state results from synchronization constraints that can reduce potential parallelism).

Acknowledgements

We are grateful to Ö. Babaoğlu, E. Leu, F. Mattern, F. Schneider and S. Zaks for their comments on the inevitability concept. This work has been supported in part by the Commission of European Communities under ESPRIT Programme BRA 6360 (BROADCAST), by the French CNRS under the grant Parallel Traces and by a French-Israeli grant on distributed computing.

Appendix: Correctness Proof of Formula F

Let $v(s_i)$ and $v(s_j)$ be the vector timestamps associated respectively with local states s_i and s_j. Moreover let r_i and r_j the relevant events that produced local states s_i and s_j.

In order to use results of [6, 16] about timestamping of events we associate with each relevant event (r_i and r_j) the timestamp of the local state it produced. So $v(r_i) = v(s_i)$ and $v(r_j) = v(s_j)$.

$$\begin{aligned} s_i \overset{w}{\rightarrow} s_j &\Leftrightarrow r_i \overset{e}{\rightarrow} r_j \text{ (definition of } \overset{w}{\rightarrow}\text{, Section 2.4)} \\ &\Leftrightarrow v(r_i)[i] \leq v(r_j)[i] \text{ ([6, 16])} \\ &\Leftrightarrow v(s_i)[i] \leq v(s_j)[i] \end{aligned}$$

□

References

[1] Ö. Babaoğlu and K. Marzullo. *Consistent global states of distributed systems: fundamental concepts and mechanisms, in Distributed Systems*, chapter 4, pages 55–93. *ACM Press, Frontier Series*, (S.J. Mullender Ed.), 1993.

[2] Ö. Babaoğlu and M. Raynal. Specification and detection of behavioral patterns in distributed computations. In *Proc. of 4th IFIP WG 10.4 Int. Conference on Dependable Computing for Critical Applications*, Springer Verlag Series in Dependable Computing, San Diego, January 1994.

[3] K. M. Chandy and L. Lamport. Distributed snapshots : determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[4] R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proc. ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 167–174, Santa Cruz, California, May 1991.

[5] S. Even. *Graph theory.* Computer Science Press, 1979.

[6] J. Fidge. Timestamps in message passing systems that preserve the partial ordering. In *Proc. 11th Australian Computer Science Conference*, pages 55–66, February 1988.

[7] E. Fromentin and M. Raynal. Local states in distributed computations: a few relations and formulas. *ACM Operating Systems Review*, 28(2):65–72, April 1994.

[8] E. Fromentin and M. Raynal. *When all the observers of a distributed computation do agree.* Research Report 2194, INRIA, Mars 1994.

[9] E. Fromentin, M. Raynal, V.K. Garg, and A.I. Tomlinson. On the fly testing of regular patterns in distributed computations. In *Proc. of the* 23^{rd} *International Conference on Parallel Processing*, St. Charles, IL, August 1994. To appear.

[10] V. K. Garg and B. Waldecker. Detection of unstable predicates in distributed programs. In *Twelfth International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 253–264, Springer Verlag, LNCS 625, New Delhi, India, December 1992.

[11] V.K. Garg. Some optimal algorithms for decomposed partially ordered sets. *Information Processing Letters*, 44:39–43, 1992.

[12] M. Hurfin, N. Plouzeau, and M. Raynal. A debugging tool for distributed Estelle programs. *Journal of Computer Communications*, 16(5):328–333, May 1993.

[13] M. Hurfin, N. Plouzeau, and M. Raynal. Detecting atomic sequences of predicates in distributed computations. In *Proc. ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 32–42, San Diego, CA, May 1993. (Reprinted in SIGPLAN Notices, Dec. 1993).

[14] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[15] K. Marzullo and L. Sabel. *Using Consistent Subcuts for Detecting Stable Properties*. Technical Report 91-1205, Dpt. of Computer Science, Cornell Univerity, Ithaca, New York, May 1991. 11 pages.

[16] F. Mattern. Virtual time and global states of distributed systems. In Cosnard, Quinton, Raynal, and Robert, editors, *Parallel and Distributed Algorithms*, pages 215–226, North-Holland, October 1988.

[17] M. Raynal, M. Mizuno, and M.L. Neilsen. Synchronization and concurrency measure for distributed computations. In *Proc. of the 12th IEEE International Conference on Distributed Computing Systems*, pages 700–707, Yokohama, Japan, June 1992.

[18] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations : in search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[19] E. Szpilrajn. Sur l'extension de l'ordre partiel. *Fund. Math.*, 16:386–389, 1930.

[20] J. Xu and R. H. B. Netzer. Adaptative independent checkpointing for reducing rollback propagation. In *Proc. 5th IEEE Symposium on Parallel and Distributed Processing*, pages 754–761, Dallas, December 1993.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399