UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1 9 9 2

25ème anniversaire

N° 1720

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

REACHABILITY ANALYSIS ON DISTRIBUTED EXECUTIONS

Claire DIEHL
Claude JARD
Jean-Xavier RAMPON

Juillet 1992

RR-1720

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Reachability Analysis on Distributed Executions

Claire DIEHL, Claude JARD, Jean-Xavier RAMPON
e-mail: jard@irisa.fr

Publication Interne n°661 - Juin 1992 - 18 pages - Programme 1

abstract

The work described in this paper belongs to the general context of distributed program debugging. For the goal of verification, the expected behavior or the suspected errors of the system under test, is described by a global property: basically a global predicate on variables, or the possible occurrences orders of observable events. The problem is to verify whether this property is satisfied or not during the execution. In the context of asynchronous parallelism on distributed architectures that we are considering, the correct evaluation of global properties requires a careful analysis of the causal structure of the execution. The basis of the method is a reachability analysis of the state space associated to a distributed execution. We study on-line algorithms dedicated to trace checking which are efficient both in time and memory. We show that the state space can be linearly built due to its algebraic structure of lattice.

Treillis des états accessibles au cours d'une exécution répartie

résumé

L'étude de cet article appartient au domaine du déverminage de programme répartis. Pour faire de la vérification, on décrit le comportement attendu ou les erreurs suspectées par une propriété globale : par exemple un prédicat sur des variables ou les ordres possibles d'occurrence d'événements observables. Le problème est de vérifier si cette propriété est satisfaite ou non durant l'éxecution. Notre méthode repose sur une analyse d'accessibilité du graphe d'états associé à une exécution répartie. Nous étudions des algorithmes dédiés à la vérification de traces à la volée qui sont efficaces à la fois en temps et en mémoire. Pour la construction du graphe des états, nous nous appuyons sur sa structure de treillis.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE (U.R.A. 227) UNIVERSITE DE RENNES I INSA DE RENNES
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE (UNITE DE RECHERCHE DE RENNES)

1 Introduction

1.1 Problem statement

Progress in computer technology brings up parallelism and data distribution to an unavoidable level. However this is not a painless way: parallel and distributed programs are still complex objects. All the aspects of their development are not well mastered; observing their behaviors often reveals unexpected situations.

Our work belongs to the general context of distributed program debugging. We deal with verification techniques based on execution traces that we call "trace checking". For the goal of verification, the expected behavior or the suspected error of the system under test, is described by a global property (basically a global predicate on variables, or the possible occurrence orders of observable events). The problem is to verify whether this property is satisfied or not during the execution.

In the general context of asynchronous parallelism on distributed architectures that we are considering, the correct evaluation of global properties requires a careful analysis of the causal structure (which is a partial order) of the execution. The basis of the method is a reachability analysis of the state space associated to a distributed execution. We study on line algorithms dedicated to trace checking which are efficient both in time and memory.

1.2 Proposed approach

Trace checking rises several problems that must be solved:

- At the lowest level the runtime must provide the basic services of timestamping the communication actions. This gives information to decide causality between particular observable events. We use the classical Lamport's definition of message causality [12] and its "on the fly" coding given by Mattern and Fidge's vector clocks [8, 15]. Although all the communication events are modified at runtime, just a few significant observable events have to be traced for the goal of analysis. We slightly modify the timestamping mechanism to deal with observable events only.

- Deciding causality between events is not the most convenient way to represent the causality order. We show that in fact the covering relation (*i.e.* the transitive reduction of the order) can also be computed on the fly. For the goal of producing the immediate predecessors of an event when it occurs, we extend the vector clock with a bit array.

- Finally we show that the graph structure of the reachable states can be computed step by step at each event occurrence.

The last two algorithms are new. They allow to perform a reachability analysis in parallel with the considered computation. The time complexity is linear in the size of the state graph. Moreover, provided that observation preserves message causality, the construction is performed strictly on the fly: event by event with no additional delay.

The method used is able to code the state graph with the same size that the covering graph. This should be exploited by verification methods, thanks to the theory of orders which provides a good basis to deal with these problems.

1.3 Related work

Message causality is fundamental to many problems on distributed traces. In that way it has been studied for different goals:

- determining consistent snapshots or consistent recovery points in the field of distributed debugging or distributed database management [4, 15];
- execution replay [13, 14, 10];
- verifying logical properties in order to detect unexpected situations [6];
- getting performance measurements on global indicators [9, 5].

Most work in progress on the fundamentals of traces are based on the partial order defined by the causality relation [12]. To our knowledge, Cooper and Marzullo [6] were the first to perform a reachability analysis on the state space associated to a distributed execution. Their work however gives rise to the problem of the parallelism between the analysis process and the distributed computation. They require events to be considered level by level (the "level" is the number of predecessors). The analysis must then be blocked awaiting an event: in the worst case, where an isolated event occurs at the end of the trace, the analysis is postponed to the end of the trace. The basis of their algorithm is to enumerate all the possible nodes of the largest state space (p^n where p is the number of events per processor and n the number of processors), and then to remove nodes that are not reachable for the considered trace (by considering vector timestamps associated to the events).

We considerably improve the technique in allowing the analysis event by event: any linear extension can be processed. It can be referred as the "on the fly reachability analysis". This is made possible by actually computing on line the covering relation, rather than considering only vector stamps and makes best use of the lattice structure by a direct construction.

2 Abstract causality order

2.1 Message causality between observable events

From an abstract point of view, a distributed program consists of n sequential processes $P_1, ..., P_n$ communicating solely by messages. The behavior of each process is completely defined by a local algorithm which determines its reaction towards incoming messages: local state changes and sending of messages to other processes. A distributed computation is the concurrent and coordinated execution of all these local algorithms. A standard way to deal with distributed computations is to consider that local actions are defined as events. Only a few of them are significant for the purpose of verification.

We will denote by $E = X \uplus \bar{X} \uplus O$ the finite set of events occurring during a computation. X contains all the sending events, $\bar{X}$ the corresponding receipts and O the internal events defined as observable by the user. We also consider that E is partitioned into disjoint subsets E_i of local events occurred on process P_i: $E = \biguplus_{1 \leq i \leq n} E_i$.

Arguing from the fact that the only mean to gain knowledge for a process in a general distributed system is to receive messages from outside, one considers the receipt of a message as causally related to the corresponding sending. The causality between local events is defined by the local algorithm: a simple way is to consider the total ordering induced by the local sequentiality (denoted by $\prec_i$). The causality relation (defined by Lamport in [12]) in E^2 is the smallest relation $\prec$ satisfying:

1. $\forall i \in \{1..n\}, \forall x, y \in E_i, \quad x \prec_i y \Longrightarrow x \prec y$
2. $\forall x \in X, \quad x \prec \bar{x}$ ($\bar{x}$ the corresponding receipt of the sending event x)
3. $\prec$ is transitive

Notations and definitions

Now we give some general definitions and notations related to partially ordered sets.
A set P associated with an order relation $\leq_{\tilde{P}}$ is called a partially ordered set or **poset**.
Such a poset is denoted by $\tilde{P} = (P, \leq_{\tilde{P}})$. If the order relation is irreflexive, we write it $\tilde{P} = (P, <_{\tilde{P}})$.
Let $\tilde{P} = (P, \leq_{\tilde{P}})$, be a poset and x and y two elements of P.
If $x <_{\tilde{P}} y$ and $\forall z \in P$, $(x <_{\tilde{P}} z \leq_{\tilde{P}} y) \Rightarrow (z = y)$ we say that x is covered by y or x is immediate predecessor of y and we write $x \text{-}<_{\tilde{P}} y$. We obtain the **covering relation** of $\tilde{P}$.
Let A be a subset of P.
The **subposet** of $\tilde{P}$ on A, denoted by $\tilde{P}/_A$, is the poset induced by $\tilde{P}$ on A.
In that way we shall write :$\tilde{E} = (E, \prec)$ and $\tilde{O} = \tilde{E}/_O = (O, \prec)$.
A is a **chain** (resp. an **antichain**) in $\tilde{P}$ iff all elements of A are pairwise comparable (resp. incomparable) in $\tilde{P}$.
$Max(A, \tilde{P}) = \{a \in A, \forall x \in A, (a \leq_{\tilde{P}} x) \Rightarrow (a = x)\}$ is the set of **maximal** elements of A in $\tilde{P}$.
$Min(A, \tilde{P}) = \{a \in A, \forall x \in A, (x \leq_{\tilde{P}} a) \Rightarrow (a = x)\}$ is the set of **minimal** elements of A in $\tilde{P}$.
$\downarrow_{\tilde{P}} A] = \{x \in P, \exists a \in A, x \leq_{\tilde{P}} a\}$ is the **predecessor set** of A in $\tilde{P}$.
$\downarrow_{\tilde{P}} A[= \downarrow_{\tilde{P}} A] \setminus A$ is the **strictly predecessor set** of A in $\tilde{P}$.
$\downarrow^{im}_{\tilde{P}} A = Max(\downarrow_{\tilde{P}} A[, \tilde{P})$ is the **immediate predecessor set** of A in $\tilde{P}$.
If the set A is a singleton $\{x\}$, we shall simply write $\downarrow_{\tilde{P}} x]$, $\downarrow_{\tilde{P}} x[$ and $\downarrow^{im}_{\tilde{P}} x$.
A is an **ideal** of $\tilde{P}$ iff $\forall a \in A, \quad \downarrow_{\tilde{P}} a] \subseteq A$.

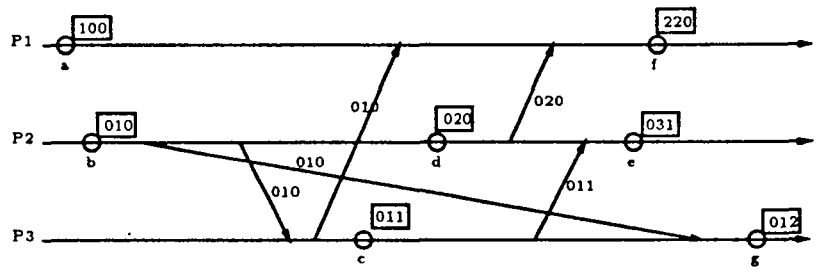

Figure 1: Computation of message causality

2.2 On the fly computation of causality

In order to characterize on the fly the message causality, Fidge and Mattern [8, 15] have developed a mechanism of logical clocks. Each event is stamped by a vector of $\mathbb{N}^n$ and the stamps ordering exactly codes the causality: it is an embedding of the causality order in $(\mathbb{N}^n, <_{\mathbb{N}^n})$[1]. Formally, the timestamping is defined by the map [7]:

$$\begin{aligned} \delta \ : \ & E \longrightarrow \mathbb{N}^n \\ & e \longmapsto (|\downarrow_{\tilde{E}} e] \cap E_i|)_{1 \leq i \leq n} \end{aligned}$$

and we have the fundamental property:

$$\forall e, f \in E, e \prec f \iff \delta(e) <_{\mathbb{N}^n} \delta(f).$$

We modify the algorithm proposed by Fidge and Mattern to stamp only observable events. We compute the map:

$$\begin{aligned} \delta \ : \ & O \longrightarrow \mathbb{N}^n \\ & e \longmapsto (|\downarrow_{\tilde{O}} e] \cap E_i|)_{1 \leq i \leq n} \end{aligned}$$

which obviously codes $\tilde{O}$. Stamps are growing slower because they count less events and, as we will see in the next section, computing the covering graph of $\tilde{O}$ also simplifies our on the fly algorithm. The timestamping mechanism follows:

- Each processor P_i owns a logical clock $c_i \in \mathbb{N}^n$. Each c_i is initialized to $(0...0)$.
- Each message sent by P_i is stamped by the current value of c_i.
- When P_i receives a message stamped by c_m, P_i updates its clock[2]: $c_i := max(c_i, c_m)$.
- When an observable event e occurs on P_i, P_i increments the i^{th} component of its clock: $c_i := c_i + (0..1_i..0)$ (only the i^{th} component is incremented), and e is stamped by c_i: $\delta(e) := c_i$.

The figure 1 shows an application of this algorithm. We only put the event stamps and the message stamps. This execution is used throughout the paper.

[1] $<_{\mathbb{N}^n}$ is the canonical order on $\mathbb{N}^n$: $\forall x, y \in \mathbb{N}^n$, $x <_{\mathbb{N}^n} y \iff \forall i \in \{1..n\}$, $x[i] \leq y[i]$ and $\exists j \in \{1..n\}$, $x[j] < y[j]$

[2] $\forall x, y \in \mathbb{N}^n \ \forall i \in \{1..n\}$, $max(x, y)[i] = max(x[i], y[i])$

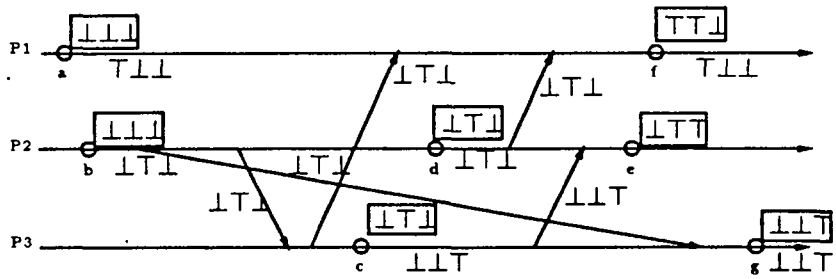

Figure 2: On the fly computation of the covering

2.3 On the fly computation of the covering

We now present an algorithm based on the vector clock mechanism which computes on the fly the covering relation of the message causality order. It avoids an expensive computation stage: the computation of the covering relation from the vector stamp trace.

This new algorithm is based on the following remarks :

1. $\forall i \in \{1..n\}$, $\forall j \in I\!N$, $j \leq |O \cap E_i| \Longrightarrow \exists! e \in O \cap E_i$, $\delta(e)[i] = j$
   ($\forall f \in O \cap E_i$, f is the $\delta(f)[i]^{th}$ observable event on P_i, hence it's unique.)

2. $\forall e \in O \cap E_i$, $\forall f \in O \setminus E_i$, $e \dashv\prec f \Longrightarrow \delta(f)[i] = \delta(e)[i]$
   (When f occurs on P_j, the last event which P_j knows on P_i is necessary the $\delta(f)[i]^{th}$ on P_i.)

Therefore, in order to know the immediate predecessors of an event e, we only have to know both its stamp $\delta(e)$ and the processors where they occurred. Thus, in addition to δ, we have to compute the map:

$$\begin{array}{rcl} \mu & : & O \longrightarrow \{\top, \bot\}^n \\ & & e \longmapsto \mu(e) \end{array}$$

verifying: $\mu(e)[i] = (\downarrow_{\widetilde{O}}^{im} e \cap E_i \neq \emptyset)$

Computation of μ goes with computation of logical clocks c_i. This is performed on the fly according to the following rules (see figure 2 for an example):

- Each processor P_i owns a boolean vector $m_i \in \{\top, \bot\}^n$ which indicates where the events currently covered occurred. For instance, if $m_i[j] = \top$ then an observable event currently covered by P_i has occurred on P_j. Each m_i is initialized to $(\bot..\bot)$.

- Each message sent by P_i is stamped by the current value of m_i.

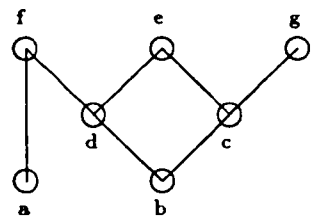

Figure 3: The execution covering graph

- When P_i receives a message stamped by $c_m \in I\!N^n$ and $m_m \in \{\top, \bot\}^n$, P_i updates m_i. The new value of m_i depends on m_i, c_i, m_m and c_m.

$$\begin{array}{ll} \forall j \in \{1..n\} \text{ if } & c_m[j] > c_i[j] \text{ then } m_i[j] := m_m[j] \\ & \text{else if } c_i[j] > c_m[j] \text{ then } m_i[j] := m_i[j] \\ & \text{else } m_i[j] := (m_i[j] \wedge m_m[j]) \ (\wedge\text{: logical and}) \end{array}$$

- When an observable event e occurs on P_i, e is stamped first ($\mu(e) := m_i$) and then P_i updates m_i: $m_i := (\bot ... \top_i ... \bot)$ (only the i^{th} component is equal to $\top$: the only covered event is e).

In practice, each process informs a "global observer" when an observable event occurs: it "traces" its identifier e and its two stamp values $\delta(e)$ and $\mu(e)$. The observer maintains a sequence of events for each process. When an event $e \in E_i \cap O$ is traced, the observer determines its immediate predecessor set according to the relation:

$$\forall j \in \{1..n\}, \forall f \in O \cap E_j, \begin{cases} \text{if } j \neq i & f \prec e \iff \mu(e)[j] \wedge (\delta(f)[j] = \delta(e)[j]) \\ \text{if } j = i & f \prec e \iff \mu(e)[j] \wedge (\delta(f)[j] = \delta(e)[j] - 1) \end{cases}$$

For each $j \neq i$ for which $\mu(e)[j]$ is true, the $\delta(e)[j]^{th}$ event in the sequence corresponding to the j^{th} process is an immediate predecessor of e. If $\mu(e)[i]$ is true, the $(\delta(e)[i]-1)^{th}$ event of P_i is also covered by e. Thus we have to do the realistic assumption that all predecessors of e have been already traced. (If we are not in this case, there exist algorithms constructing a linear extension of the causality.) Therefore to calculate the covering relation without delay, we have to process the events according to any linear extension of the causality order and we have to memorize all sequences of events. The figure 3 shows the covering graph of our example.

3 Associated state graph

3.1 State graph and the ideal lattice

Building the state graph associated to a distributed trace consists in "replaying" the trace, recording the changes of a global state vector. The only constraint during the

replay is the causality preservation. We adopt the standard interleaving semantics for parallelism which considers only one move at a given time. The local state of a process P_i is picked up between two local events. In order to capture in the state all the messages in transit, we can identify the local state with the set of all the local events which have been already considered in the past of the process. To define the initial state, one can consider a minimal event $\emptyset$ in the past of all the observable events. The global state is the union of the local states for each process.

Notice that the causality constraint only produces global states being closed by the causality relation: the set of global states is isomorphic to the ideal lattice of the causality order. See figure 5 to have a look of the state graph of our example.

3.2 Fundamentals

Since our goal is to compute on the fly the state graph of a distributed execution, we studied correlations between ideal lattices yielded by a poset and by one of its subposet when the missing vertex is a maximal one. We obtain a complete characterization of correlations between these two lattices assuming that the initial poset has at least a maximal element.

We denote by $I(P)$ the set of all ideals of $\tilde{P}$ and by $\widetilde{I(P)}$ this set ordered by inclusion.

Remark 1 *For any I_1, I_2 in $I(P)$, $I_1 \cup I_2$ and $I_1 \cap I_2$ belong to $I(P)$. Moreover, $I_1 \cup I_2$ (resp. $I_1 \cap I_2$) is the smallest (resp. greatest) element of $I(P)$ including (resp. included in) both I_1 and I_2. Thus this gives to $\widetilde{I(P)}$ a lattice structure (a poset $\tilde{T}$ is a lattice iff any of its two elements subset has a supremum and an infimum in $\tilde{T}$).*

In order to establish our characterization theorem, we have to introduce some definitions and we need a result from Bonnet and Pouzet [2] extending to the infinite case the well known result of Birkhoff [1] on finite distributive lattices and posets.

Let $\tilde{T}$ be a lattice, and for any subset A of T let us denote by $\bigvee_{\tilde{T}} A$ (resp. $\bigwedge_{\tilde{T}} A$) the supremum (resp. the infimum) of A in $\tilde{T}$. Then:

* $\tilde{T}$ is **complete** iff $\forall A \subseteq T$, $\bigvee_{\tilde{T}} A \in T$.
* $\tilde{T}$ is **completely join distributive** iff $\forall A \subseteq T, \forall x \in T, (\bigvee_{\tilde{T}} A) \bigwedge_{\tilde{T}} x = \bigvee_{\tilde{T}} \{y \bigwedge_{\tilde{T}} x, y \in A\}$.
* $\tilde{T}$ is **join generated** by a subset A of T iff $\forall x \in T, \exists B \subseteq A$, $x = \bigvee_{\tilde{T}} B$ and $\forall B \subseteq A$, $\bigvee_{\tilde{T}} B \in T$.
* an element x of T is **completely join irreducible** in $\tilde{T}$ iff $\forall A \subseteq T$, $(x = \bigvee_{\tilde{T}} A) \Rightarrow (x \in A)$.

Theorem 1 **[2]**

1. *Let $\tilde{P}$ be an order, then $\widetilde{I(P)}$ is:*

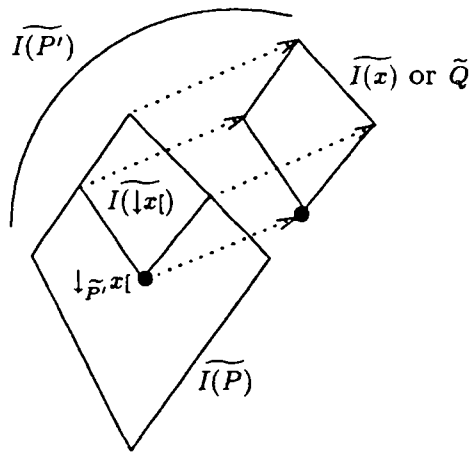

Figure 4: Illustration of theorem 2

(a) a complete, completely join distributive lattice,

(b) join generated by its completely join irreducible elements.

Moreover, $\tilde{P}$ is isomorphic by the map $i(x) = \{y \in X,\ y \leq x\}$ to the suborder of $\widetilde{I(P)}$ on its completely join irreducible elements.

2. *Let $\tilde{T}$ be an order satisfying conditions (1a) and (1b), then $\tilde{T}$ is isomorphic by the map $j(x) = \{y \in T,\ y \subseteq x\} \bigcap \{$completely join irreducible element of T in $\tilde{T}\}$ to the order $\widetilde{I(P)}$ where $\tilde{P}$ is the suborder of $\tilde{T}$ on its completely join irreducible elements.*

For the proof of the following theorem, we need the following Lemma:

Lemma 1 *Let $\tilde{P}$ be an order, $\forall I, J \in I(P)$, $I \text{-}<_{\widetilde{I(P)}} J \Longleftrightarrow I \subset J$ and $\mid J \setminus I \mid = 1$*

Proof: (i) Assume that $I \subset J$, then $I <_{\widetilde{I(P)}} J$. The result follows directly from the fact that $\forall Z, K \in I(P)$, $Z <_{\widetilde{I(P)}} K \Longrightarrow Z \subset K$.
(ii) Assume that $I \text{-}<_{\widetilde{I(P)}} J$, then $I \subset J$. If $\mid J \setminus I \mid = 0$ then $I = J$ which is in contradiction with $I \text{-}<_{\widetilde{I(P)}} J$. Assume that $\mid J \setminus I \mid \geq 2$ and let $x, y \in J \setminus I$ with $x \neq y$. W.l.o.g. we can assume that $y \not<_{\tilde{P}} x$, then $I <_{\widetilde{I(P)}} I \cup \downarrow_{\tilde{P}} x] <_{\widetilde{I(P)}} J$ which is in contradiction with $I \text{-}<_{\widetilde{I(P)}} J$. □

Theorem 2 *Let $\widetilde{P'}$ be a poset with a least one maximal element. Let x be a maximal element in $\widetilde{P'}$ and let $\tilde{P}$ be a poset such that $P' \setminus P = \{x\}$ and $\widetilde{P'}/_P = \tilde{P}$.*
Let $I(\downarrow x[) = \{I \in I(P),\ \downarrow_{\widetilde{P'}} x[\leq_{\widetilde{I(P)}} I\}$ (we have $\downarrow_{\widetilde{P'}} x[\in I(P)$).
Let $\tilde{Q}$ be a poset isomorphic to $I(\downarrow x[)$ by a map ϕ.
The poset $\tilde{Z}$ defined by: $Z = Q \uplus I(P)$, $\tilde{Z}/_{I(P)} = \widetilde{I(P)}$, $\tilde{Z}/_Q = \tilde{Q}$ and
$\forall p, q \in I(P) \times Q \ \ (p -<_{\tilde{Z}} q \iff q = \phi(p))$
is isomorphic to $\widetilde{I(P')}$ by the map:

$$\varphi : z \longmapsto \begin{cases} z & \text{if } z \in I(P) \\ \phi(z) \cup \{x\} & \text{otherwise} \end{cases}$$

For an illustration of the theorem, see figure 4.

Proof: Let $I(x) = \{I \in I(P'),\ x \in I\}$.

We are going to show that $\widetilde{I(P')}/_{I(P') \setminus I(x)}$ is isomorphic to $\widetilde{I(P)}$. Let $A \subseteq I(P') \setminus I(x)$, it is then clear that $\bigcap A \in I(P') \setminus I(x)$ and that $\bigcup A \in I(P') \setminus I(x)$, thus $\widetilde{I(P')}/_{I(P') \setminus I(x)}$ is a complete completely join distributive lattice (since $\widetilde{I(P')}$ is a join distributive lattice). Since x is a maximal element in $\widetilde{P'}$, we have:
(i) any completely join irreducible element in $\widetilde{I(P')}/_{I(P') \setminus I(x)}$ is a completely join irreducible element in $\widetilde{I(P')}$. Indeed, $\forall I \in I(P') \setminus I(x)$, $\{J \in I(P),\ J \leq_{\widetilde{I(P')}} I\} \subset I(P') \setminus I(x)$.
(ii) any completely join irreducible element in $\widetilde{I(P')}$ belonging to $I(P') \setminus I(x)$ is a completely join irreducible element in $\widetilde{I(P')}/_{I(P') \setminus I(x)}$. By definition of $I(x)$.
Thus $\widetilde{I(P')}/_{I(P') \setminus I(x)}$ is join generated by the completely join irreducible element in $\widetilde{I(P')}$ which does not contains $\{x\}$ as subset. Using Theorem 1, we can immediately deduce that $\widetilde{I(P')}/_{I(P') \setminus I(x)}$ is isomorphic to $\widetilde{I(P)}$.

Now, we are going to show that $\widetilde{I(P)}/_{I(\downarrow x)}$ is isomorphic to $\widetilde{I(P')}/_{I(x)}$ by the map ψ from $I(\downarrow x[)$ to $I(x)$ such that $\psi(I) = I \cup \{x\}$. Since ψ is a one to one mapping (ψ is injective and since x is a maximal element in $\widetilde{P'}$, $\forall I \in I(x), I \setminus \{x\} \in I(\downarrow x[)$) and since $\forall I, J \in I(\downarrow x[), I \subset J \iff I \cup \{x\} \subset J \cup \{x\}$, ψ is an isomorphism between $\widetilde{I(P)}/_{I(\downarrow x)}$ and $\widetilde{I(P')}/_{I(x)}$.

In order to conclude the proof, it remains to show that $\forall I \in I(x), \exists! I' \in I(P') \setminus I(x)$ such that $I' -<_{\widetilde{I(P')}} I$ and that $I' = I \setminus \{x\}$. Since x is a maximal element in $\widetilde{P'}$, $\forall I \in I(x)$, $I \setminus \{x\} \in I(P') \setminus I(x)$, thus by Lemma 1 $I \setminus \{x\} -<_{\widetilde{I(P')}} I$. Since $\forall I \in I(P') \setminus I(x)$, $x \notin I$ and $I -<_{\widetilde{I(P')}} I \cup \{x\}$, the remaining follows directly from Lemma 1. □

Remark 2 *When $\tilde{P}$ is assumed to be anti well founded (i.e. any non empty set has a maximal element), we can denote $\tilde{Z}$ by $\tilde{P} *^x_{D(x)}$ with $D(x) = \downarrow^{im}_{\widetilde{P'}} x$. Thus we obtain a coding of the ideal lattice of such a poset from any of its linear extensions. Moreover, when $\tilde{P}$ is assumed finite, we get an efficient and effective coding mechanism.*

3.3 On the fly computation of the state graph

Assuming that the number of processes involved in a distributed computation is finite, using our previous theorem, we are now able to give an algorithm for an on the fly computation of a distributed execution state graph (assuming the knowledge of the covering causality relation).

Its **width**, denoted $w(\tilde{P})$, is the length of one of its largest antichains.
A **chain decomposition** of $\tilde{P}$ is a partition $\{P_i\}_{i\in I}$ of P, where each P_i is a chain in $\tilde{P}$.

The "granulation algorithm"

Input:

1. The transitive reduction of $\widetilde{P'}$ with a chain decomposition $\{P'_i\}_{1\leq i\leq k}$.
2. For any $y \in P'$, $\delta(y) = (|\downarrow_{\widetilde{E}} y] \cap P'_i|)_{1\leq i\leq k}$ (same definition as in paragraph 2.2) and $i(y)$ such that $y \in P'_{i(y)}$.
3. A vertex $x \in Max(P', \widetilde{P'})$ and the set $D(x) = \downarrow^{im}_{\widetilde{P'}} x$.
4. When $P = P' \setminus \{x\}$ and $\widetilde{P'}/_P = \tilde{P}$, the transitive reduction of $\widetilde{I(P)}$, such that:
   (a) Each $y \in P$ is directly related to its corresponding $\downarrow_{\tilde{P}} y]$ in $\widetilde{I(P)}$.
   (b) Each edge yz in the transitive reduction of $\widetilde{I(P)}$ is labeled by $I_z \setminus I_y$ where I_y (resp. I_z) is the ideal of $\tilde{P}$ corresponding to the vertex y (resp. z) in $\widetilde{I(P)}$.
   (c) Outgoing edges of any vertices in the transitive reduction of $\widetilde{I(P)}$ are stored ordered by increasing index of the chain their label belong to.

Body:

1. Find $\downarrow_{\tilde{P}} D(x)]$ in $\widetilde{I(P)}$.
2. Build a poset $\tilde{Q}$ isomorphic to $\widetilde{I(\downarrow x[)}$, when $I(\downarrow x[) = \{I \in I(P), \downarrow_{\tilde{P}} D(x)] \leq_{\widetilde{I(P)}} I\}$ by a map ϕ.
3. For any $I \in I(\downarrow x[)$, create the edge $(I, \phi(I))$ with label x and store this edge according to the storage order.
4. Create a link between x and $\phi(\downarrow_{\tilde{P}} D(x)])$.

Output:

The transitive reduction of $\widetilde{I(P')}$, such that:

(a) Each $y' \in P'$ is directly related to its corresponding $\downarrow_{\tilde{P'}} y']$ in $\widetilde{I(P')}$.
(b) Each edge $y'z'$ in $\widetilde{I(P')}$ is labeled by $I_{z'} \setminus I_{y'}$ where $I_{y'}$ (resp. $I_{z'}$) is the ideal of $\widetilde{P'}$ corresponding to the vertex y' (resp. z') in $\widetilde{I(P')}$.
(c) Outgoing edges of any vertices in the transitive reduction of $\widetilde{I(P)}$ are stored ordered by increasing index of the chain their label belong to.

In order to prove the time complexity of the "granulation algorithm", we need some

more properties between a chain decomposition of an order and its ideal lattice.

Let $\widetilde{P}$ be an order, let $\{P_i\}_{1\leq i\leq k}$ be a chain decomposition of $\widetilde{P}$ and let Δ be the map defined by:

$$\begin{aligned}\Delta \; : \; & I(P) \longrightarrow I\!N^k \\ & I \longmapsto (\mid I \cap P_i \mid)_{1\leq i\leq n}\end{aligned}$$

Proposition 1 *$\forall I, J \in I(P)$, the following properties are equivalent:*

(i) $I \leq_{\widetilde{I(P)}} J$

(ii) $\Delta(I) \leq_{I\!N^k} \Delta(J)$

Proof: $(i) \Longrightarrow (ii)$: Since I and J are ideals, $\forall i \in \{1,\ldots,k\}$ we have $I \cap P_i \subseteq J \cap P_i$. Then, $\forall i \in \{1,\ldots,k\}$ we have $\Delta(I)[i] \leq_{I\!N} \Delta(J)[i]$ and thus $\Delta(I) \leq_{I\!N^k} \Delta(J)$.
$(ii) \Longrightarrow (i)$: For any ideal $A \in I(P)$ and for any $i \in \{1,\ldots,k\}$, if $\mid A \cap P_i \mid = \alpha_i \neq \emptyset$ then $A \cap P_i$ is a maximal subchain in $\widetilde{P_i}$ with α_i elements and containing the smallest element of $\widetilde{P_i}$. Thus, for any $i \in \{1,\ldots,k\}$ we have: $\Delta(I)[i] \leq_{I\!N} \Delta(J)[i] \Longrightarrow I \cap P_i \subseteq J \cap P_i$. Consequently, $I = (I \cap (\bigcup_{1\leq i\leq k} P_i)) \subseteq (J \cap (\bigcup_{1\leq i\leq k} P_i)) = J$. □

Proposition 2 *$\forall I, J \in I(P)$, the following properties are equivalent:*

(i) $I \leq_{\widetilde{I(P)}} J$

(ii) All maximal chains from I to J in $\widetilde{I(P)}$ have length $lg(I,J) = \sum_{i=1}^{k}(\Delta(J)[i] - \Delta(I)[i])$. And there is at least one maximal chain.

Proof: As the proposition holds clearly for $I = J$, assume that $I \neq J$. Since $(ii) \Longrightarrow (i)$ is obvious, we are going to show that $(i) \Longrightarrow (ii)$: Since $I \leq_{\widetilde{I(P)}} J$, there exists a maximal chain from I to J in $\widetilde{I(P)}$. Let $(x_0 = I, x_1, \ldots, x_{\alpha-1}, x_\alpha = J)$ be such a chain. From Lemma 1 we have $\mid J \mid = \mid I \mid + \alpha$ and from Proposition 1, we known that $\forall i \in \{1,\ldots,k\}$, $\Delta(J)[i] - \Delta(I)[i] \geq 0$, then since $\{P_i\}_{1\leq i\leq k}$ is a partition of P, it is clear that $lg(I,J) = \alpha$. Moreover, since $\widetilde{I(P)}$ is a distributive lattice, it is modular and and thus graded. So all maximal chains have the same length. □

Theorem 3 *The "granulation algorithm" runs in time complexity:*

$$O(((\mid I(P') \mid - \mid I(P) \mid) + \mid P \mid) w(P)).$$

Proof: The correctness of the "granulation algorithm" is clearly achieved through Theorem 2.

The time complexity analysis of the "granulation algorithm" can be perform step by step:

For step 1): Choose any y in $D(x)$, it is clear that $\Delta(\downarrow_{\tilde{P}} y]) = \delta(y)$ and $\Delta(\downarrow_{\tilde{P}} D(x)]) = \delta(x) - (0..1_{i(x)}..0)$. Then from proposition 2, we know there exists a chain in the transitive reduction of $\widetilde{I(P)}$ from $\downarrow_{\tilde{P}} y]$ to $\downarrow_{\tilde{P}} D(x)]$ with exactly $\Delta(\downarrow_{\tilde{P}} D(x)])[i] - \Delta(\downarrow_{\tilde{P}} y])[i]$ edges belonging to P_i. Thus starting from $\downarrow_{\tilde{P}} y]$, we choose an outgoing edge with label z and chain index j belonging to $Ind(y, D(x))$ where $Ind(y, D(x)) = \{i, \gamma(i) > 0 \; with \; \gamma(i) = \Delta(\downarrow_{\tilde{P}} D(x)])[i] - \Delta[193z(\downarrow_{\tilde{P}} y])[i]\}$. Let $\gamma(j) = \gamma(j) - 1$, by induction on z we arrive in $\downarrow_{\tilde{P}} D(x)]$ when $Ind(y, D(x)) = \emptyset$. Since the choice of such an z can be done in $O(w(P))$, thus step 1) can be achieved in $O((\mid P \mid)w(P))$.

For step 2) and 3): Since the number of outgoing edge of any $I \in I(P)$ is bounded by $w(P)$, steps 2) and 3) can be achieved in $O((\mid I(P') \mid - \mid I(P) \mid)w(P))$ (for example through a breadth first search algorithm).

Step 4 can be done in constant time during steps 2) and 3).

□

As consequence of this theorem, we are able to achieve the computation of the ideal lattice of a poset from any of its linear extensions.

Corollary 1 *Let $\tilde{P}$ be a poset, $\widetilde{I(P)}$ can be computed in time complexity $O((\mid I(P) \mid + \mid P \mid^2)w(P))$.*

Remark 3 *When $\mid I(P) \mid$ is in $\Omega(\mid P \mid^2)$, the computation of the ideal lattice of a poset from any of its linear extensions can be performed with the same time complexity than with the algorithm given by Bordat [3]. This last algorithm, which is up to our knowledge the most efficient one, is not accurate for the on the fly case (it is based on a depth first search of the all poset).*

This algorithm is illustrated in Figure 5. However, in the sake of simplifying the picture, ideals are only labeled with their maximal elements in $\tilde{P}$ and edges are not labeled. Black nodes denote the nodes containing $D(x)$ and which will be duplicated when incorporating a new event x. Doted edges denote the edges added between the duplicated subposet of $\widetilde{I(P)}$ and its corresponding copy. Assume that the first three steps have been performed. We have a lattice $\widetilde{I(P)}$ on $\{\emptyset, c, b, d, cd\}$ and the new incoming vertex for $\tilde{P}$, labeled "a", has for immediate predecessor set $D(a) = \emptyset$. Thus we have to:

1. duplicate the subposet of $\widetilde{I(P)}$ on all elements in $I(P)$ containing $D(a)$ (here the whole lattice). A new vertex "y" obtained from a vertex "x" has for label: $label(y) = (label(x) \cup \{a\}) \setminus D(a)$. New vertices are :$\{a, ca, ba, da, cda\}$

2. add a new edge xy between unconnected vertices x and y checking that y was obtained from x. New edges are :$\{(\emptyset, a), (c, ca), (b, ba), (d, da), (cd, cda)\}$

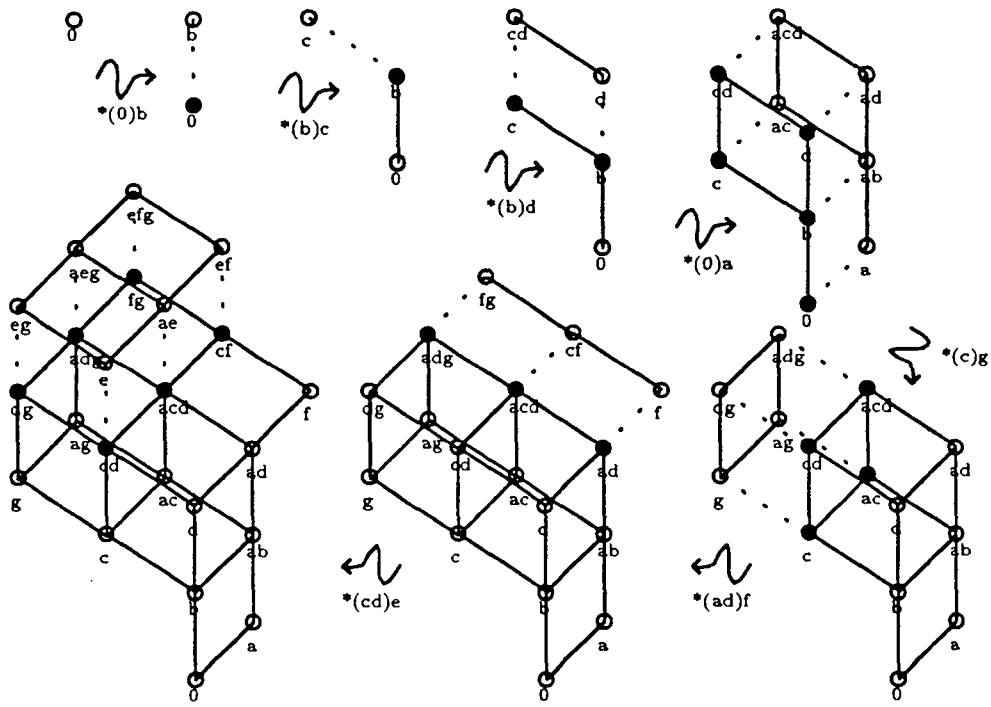

Figure 5: On the fly computation of the state graph

4 Conclusion and prospects

Trace checking for distributed programs is an important aspect of distributed debugging. The problem is complex since it requires a careful analysis of the causal structure of executions.

The use of the partial order theory is unavoidable to design efficient algorithms. As in classical verification methods for concurrent systems, the basis is a reachability analysis, *i.e* an exhaustive enumeration of the state space associated to the considered distributed execution. And we have naturally to face the combinatorial problem of state explosion. Our response to that problem is the development of on the fly techniques which allows the trace analysis in parallel with its execution: thus there is no need to store all the trace before checking properties.

In this paper, we have presented such algorithms to build the states of a distributed execution. Obviously, for the purpose of verification, our algorithm must be coupled to a verifier which will attribute the states according to the properties that have to be checked. In this phase, the classical methods used in distributed software verification can be applied [16].

Our proposal consists in two new algorithms. The first one builds the covering relation of causality between observable events: when an observable event occurs, we immediately know what are the observable events that just precede.

The second algorithm takes as input the covering relation event by event (*i.e.* any

linear extension of the causality order) and gives a way to build (or search dependingly on the verification method) the state graph. This algorithm is based on theoretical results on lattices and orders. The regular structure of the graph makes it possible to build it with a linear complexity. Its on the fly characteristic and also its time complexity substantially improve the Cooper and Marzullo's contribution for detecting global predicates.

Our main prospects are to really implement these algorithms in our favorite distributed environment Echidna [11] and first visualize the covering relation and the state graph. In second place, we will couple the building of the graph to a verifier of properties expressed by automata and temporal logic formula.

Acknowledgments

Our understanding of distributed computation has been considerably enlighten by the study of the order theory. We would like to thanks those who stimulated our thoughts: B.Caillaud, B. Charron-Bost, M. Habib, F. Mattern and M. Raynal. This work has received a financial support from the french national project C^3 on concurrency.

References

[1] G. Birkhoff. Rings of sets. *Duke Math J-3*, 311–316, 1937.

[2] R. Bonnet and M. Pouzet. Linear extension of ordered sets. In I.Rival (ed.), editor, *Ordered Sets*, pages 125–170, D.Reidel Publishing Company, 1982.

[3] J.P. Bordat. Calcul des idéaux d'un ordonné fini. *Recherche opérationnelle/Operations Research*, 25(3):265–275, 1991.

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

[5] B. Charron-Bost. Combinatorics and geometry of consistent cuts: application to concurency theory. In Bermond and Raynal, editors, *Proceedings of the international workshop on distributed algorithms*, pages 45–56, Springer–Verlag, LNCS 392, France, Nice 1989.

[6] R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proc. ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 163–173, Santa Cruz, California, May 1991.

[7] C. Diehl and C. Jard. Interval approximations of message causality in distributed executions. In Finkel and Jantzen, editors, *STACS 92 : Symposium on Theoretical Aspects of Computer Science*, pages 363–374, Springer–Verlag, LNCS 577, Cachan, february 1992.

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

[9] M. Habib, M. Morvan, and J.X. Rampon. Remarks on some concurrency measures. In *Graph–Theoretic Concepts in Computer Science*, pages 221–238, LNCS 484, june 1990.

[10] M. Hurfin, N. Plouzeau, and M. Raynal. Implementation of a distributed debugger for Estelle programs. In Paulo Verissimo, editor, *Proceeding of the ERCIM workshop*, Lisboa, November 1991.

[11] C. Jard and J.-M. Jézéquel. ECHIDNA, an Estelle-compiler to prototype protocols on distributed computers. *Concurrency Practice and Experience*, May 1992.

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

[13] T. Leblanc and J. Mellor-Crummey. Debugging parallel programs with instant replay. *IEEE Transactions on Computers*, C-36(4), April 1987.

[14] E. Leu, A. Schiper, and A. Zramdini. Efficient execution replay techniques for distributed memory architectures. In Arndt Bode, editor, *Proc. of the Second European Distributed Memory Computing Conference, Munich*, pages 315–324, April 1991.

[15] F. Mattern. Virtual time and global states of distributed systems. In Cosnard, Quinton, Raynal, and Robert, editors, *Proc. Int. Workshop on Parallel and Distributed Algorithms Bonas, France, Oct. 1988*, North Holland, 1989.

[16] C. H. West. An automated technique of communication protocol validation. *IEEE Transactions on Communications*, COM-26:1271–1275, 1978.

LISTE DES DERNIERES PUBLICATIONS INTERNES PARUES A L'IRISA

PI 653 MADMACS : UN OUTIL DE PLACEMENT ET ROUTAGE POUR LE DESSIN DE MASQUES DE RESEAUX REGULIERS
Eric GAUTRIN, Laurent PERRAUDEAU, Oumarou SIE
Avril 1992, 16 pages.

PI 656 CAUSALITY ORIENTED SHARED MEMORY FOR DISTRIBUTED SYSTEMS
Michel RAYNAL, Masaaki MIZUNO, Mitch NEILSEN
Avril 1992, 8 pages.

PI 657 ALGORITHMES PARALLELES POUR L'ANALYSE D'IMAGE PAR CHAMPS MARKOVIENS
Etienne MEMIN, Fabrice HEITZ
Mai 1992, 74 pages.

PI 658 MULTISCALE SIGNAL PROCESSING : ISOTROPIC RANDOM FIELDS ON HOMOGENEOUS TREES
Bernhard CLAUS, Ghislaine CHARTIER
Mai 1992, 28 pages.

PI 659 ON THE COVARIANCE-SEQUENCE OF AR-PROCESSES. AN INTERPOLATION PROBLEM AND ITS EXTENSION TO MULTISCALE AR-PROCESSES
Bernhard CLAUS, Albert BENVENISTE
Mai 1992, 36 pages.

PI 660 EXCEPTION HANDLING IN COMMUNICATING SEQUENTIAL PROCESSES DESIGN, VERIFICATION AND IMPLEMENTATION
Jean-Pierre BANATRE, Valérie ISSARNY
Mai 1992, 38 pages.

PI 661 REACHABILITY ANALYSIS ON DISTRIBUTED EXECUTIONS
Claire DIEHL, Claude JARD, Jean-Xavier RAMPON
Juin 1992, 18 pages.

PI 662 RECONSTRUCTION 3D DE PRIMITIVES GEOMETRIQUES PAR VISION ACTIVE
Samia BOUKIR, François CHAUMETTE
Juin 1992, 40 pages.

PI 663 FILTRES SEMANTIQUES EN CALCUL PROPOSITIONNEL
Raymond ROLLAND
Juin 1992, 22 pages.

ISSN 0249-6399