Reachability Analysis on Distributed Executions

Claire DIEHL, Claude JARD, Jean-Xavier RAMPON

IRISA,
Campus de Beaulieu,
F-35042 Rennes, FRANCE.
tel: (33) 99 84 71 00
(jard,rampon)@irisa.fr

Abstract. The paper presents basic algorithms for trace checking of distributed programs. In distributed systems, detecting global properties requires a careful analysis of the causal structure of the execution. Based on the on-the-fly observation of the partial order of message causality, we show how to build the lattice of all the reachable states of the distributed system under test. The regular structure of this graph makes it possible to build it with a quasi-linear complexity, which improves substantially the state-of-the-art.

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.

This motivates the interest of the scientific community on parallelism for distributed program debugging. Actually, 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 of the execution. The causal structure, induced by the message exchanges between processes in the distributed system, forms a partial order, as Lamport remarked in 1978. As a consequence, numerous questions about distributed executions refer to the notions of linear extensions and order ideals (also called consistent cuts). These consistent cuts define the notion of global state (snapshot) for a distributed execution. This allows to view the trace checking as a standard model checking of the set of reachable global states. Actually, one can based the testing method on a kind of reachability analysis which exactly builds the

covering graph of the ideal lattice of the causality relation. In that context, testing must be performed on-the-fly, i.e. in parallel with the execution of the application under test.

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 [15] and its "on the fly" coding given by Mattern and Fidge's vector clocks [9, 19]. 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 quasi-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. Thanks to the theory of orders which provides a good basis to deal with these problems. Trace checking is a particular case of model-checking. The reachability analysis builds the transition system associated to the considered distributed execution. Due to the lattice structure of these transition systems, reachability analysis can be performed almost linearly in time. Moreover, no doubt that existing methods to reduce the state explosion problem in standard model-checking (see [11] for example) will also apply in the near future.

1.3 Related work

Message causality is fundamental to many problems on distributed traces. Actually, 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, 19];
- execution replay [16, 17];
- verifying logical properties in order to detect unexpected situations [7];
- getting performance measurements on global indicators [12, 5].

Most work in progress on the fundamentals of traces are based on the partial order defined by the causality relation [15]. To our knowledge, Cooper and Marzullo [7] 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. More recently, a few algorithms have been published [10, 20, 18, 6] to detect some restricted classes of global properties. We think they could be explained and proved using the lattice structure of the state space.

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 [15]) in E^2 is the smallest relation $\prec$ satisfying:

1. $\forall i \in \{1..n\}, \forall x, y \in E_i, \;\; x \prec_i y \Longrightarrow x \prec y$

2. $\forall x \in X,\ x \prec \bar{x}$ ($\bar{x}$ the corresponding receipt of the sending event x)
3. $\prec$ is transitive

2.2 Definitions and preliminaries on partially ordered sets

A set P associated with a partial order relation (i.e. an antisymmetric, transitive and reflexive or irreflexive binary relation on P) is called a partially ordered set or a *poset*. If the relation is reflexive such a poset is written $\widetilde{P} = (P, \leq_{\widetilde{P}})$ else $\widetilde{P} = (P, <_{\widetilde{P}})$.

Let x and y be two elements of P:

We say that x and y are *comparable* in $\widetilde{P}$, when either $x \leq_{\widetilde{P}} y$ or $y \leq_{\widetilde{P}} x$. Otherwise, x and y are said to be *incomparable* in $\widetilde{P}$. If $x \leq_{\widetilde{P}} y$ holds, then x is a *predecessor* of y in $\widetilde{P}$ and y is a *successor* of x in $\widetilde{P}$.

We say that x is *covered* by y in $\widetilde{P}$, and we write $x \text{-}<_{\widetilde{P}} y$, if $x <_{\widetilde{P}} y$ and $\forall z \in P$, $(x <_{\widetilde{P}} z \leq_{\widetilde{P}} y) \Rightarrow (z = y)$; x is an *immediate predecessor* of y in $\widetilde{P}$ and y is an *immediate successor* of x in $\widetilde{P}$. The directed graph associated of this *covering relation* (i.e. the transitive reduction of $<_{\widetilde{P}}$) is the *covering graph* of $\widetilde{P}$ and is denoted by $Cov(\widetilde{P}) = (P, E_P)$.

Let A be a subset of P:

- $E_P(A)$ is the set of the edges corresponding to the subgraph of $Cov(\widetilde{P})$ on A.
- The *subposet* of $\widetilde{P}$ on A, $\widetilde{P}/_A = \widetilde{A} = (A, \leq_P)$ is the poset induced by $\widetilde{P}$ on A.
- For $|A| \geq 2$, if all elements of A are pairwise comparable (resp. incomparable) in $\widetilde{P}$, A is a *chain* (resp. an *antichain*) in $\widetilde{P}$. The *height* of $\widetilde{P}$, $h(P)$, is the maximum cardinality minus one of a chain in $\widetilde{P}$. The *width* of $\widetilde{P}$, $w(P)$, is the maximum cardinality of an antichain in $\widetilde{P}$. A *chain decomposition* of $\widetilde{P}$ is a partition $\{P_i\}_{i \in I}$ of P where each P_i is a chain in $\widetilde{P}$
- $Max(A, \widetilde{P}) = \{a \in A, \forall x \in A,\ (a \leq_{\widetilde{P}} x) \Rightarrow (a = x)\}$ is the *maximal* elements set of A in $\widetilde{P}$. Analogously, $Min(A, \widetilde{P}) = \{a \in A, \forall x \in A, (x \leq_{\widetilde{P}} a) \Rightarrow (a = x)\}$ is the *minimal* elements set of A in $\widetilde{P}$.
- $\downarrow_{\widetilde{P}} A] = \{x \in P, \exists a \in A, x \leq_{\widetilde{P}} a\}$ (resp. $\uparrow_{\widetilde{P}} A] = \{x \in P, \exists a \in A, a \leq_{\widetilde{P}} x\}$) is the *predecessor set* (resp. *successor set*) of A in $\widetilde{P}$.
- $\downarrow_{\widetilde{P}} A[= \downarrow_{\widetilde{P}} A] \setminus A$ (resp. $\uparrow_{\widetilde{P}} A[= \uparrow_{\widetilde{P}} A] \setminus A$) is the *strictly predecessor set* (resp. *strictly successor set*) of A in $\widetilde{P}$.
- $\downarrow^{im}_{\widetilde{P}} A = Max(\downarrow_{\widetilde{P}} A[, \widetilde{P})$ (resp. $\uparrow^{im}_{\widetilde{P}} A = Min(\uparrow_{\widetilde{P}} A[, \widetilde{P})$) is the *immediate predecessor set* (resp. *immediate successor set*) of A in $\widetilde{P}$.
- If A is a singleton $\{x\}$, we shall simply write $\downarrow_{\widetilde{P}} x]$, $\uparrow_{\widetilde{P}} x]$, $\downarrow_{\widetilde{P}} x[$, $\uparrow_{\widetilde{P}} x[$, $\downarrow^{im}_{\widetilde{P}} x$ and $\uparrow^{im}_{\widetilde{P}} x$.

A *linear extension* of a poset $\widetilde{P}$ is a chain $\widetilde{C}$ on P that preserves $\widetilde{P}$, i.e.: $x \leq_{\widetilde{P}} y \Rightarrow x \leq_{\widetilde{C}} y$.

A is an *ideal* of $\widetilde{P}$ iff it contains all its predecessors $\downarrow_{\widetilde{P}} A] = A$. $I(P)$ is the set of all ideals of $\widetilde{P}$ and $\widetilde{I(P)}$ is this set ordered by inclusion [1].

A has an *infimum* (resp. a *supremum*) in $\widetilde{P}$ if $|Max(\{x \in P, x \leq_{\widetilde{P}} y \ \ \forall y \in A\}, \widetilde{P})| = 1$ (resp. $|Min(\{x \in P, y \leq_{\widetilde{P}} x \ \ \forall y \in A\}, \widetilde{P})| = 1$).

$\widetilde{P}$ is a *lattice* iff any of its two elements subset has a supremum and an infimum in $\widetilde{P}$.

2.3 On the fly computation of causality

In order to characterize on the fly the message causality, Fidge and Mattern [9, 19] 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 $(\mathbb{N}^n, <_{\mathbb{N}^n})$ [2]. Formally, the timestamping is defined by the map [8]:

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

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:

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

which obviously codes $\widetilde{O}$. Stamps are growing slower because they count less events and, as we will see in the next section, computing the covering graph of $\widetilde{O}$ 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[3]: $c_i := max(c_i, c_m)$.
- 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: $\delta(e) := c_i$.

[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 $\widetilde{I(P)}$ is a lattice.

[2] $<_{\mathbb{N}^n}$ 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 $\exists j \in \{1..n\}$, $x[j] < y[j]$

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

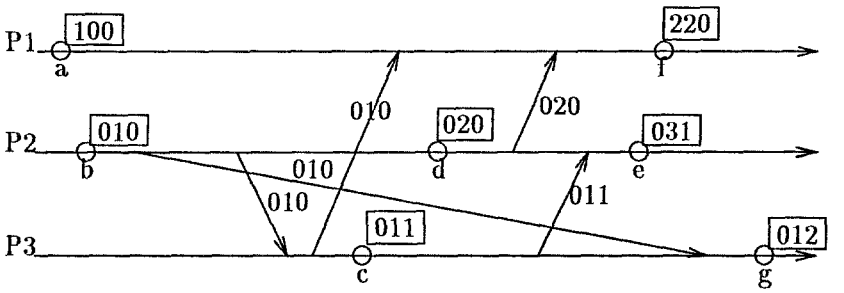

Fig. 1. Computation of message causality

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.

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

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:

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

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.

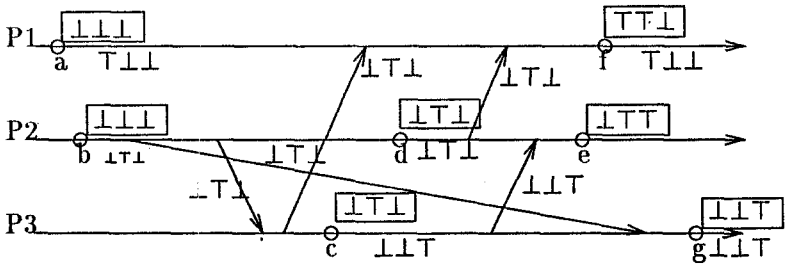

Fig. 2. On the fly computation of the covering

- When P_i receives a message stamped by $c_m \in \mathbb{N}^n$ 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.

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

- 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).

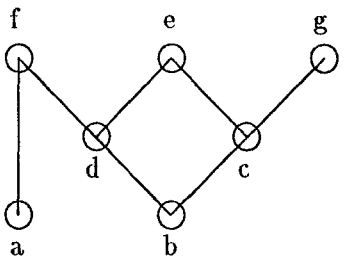

Fig. 3. The covering graph of our example

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

Studies of correlations between poset properties and lattice properties are always of interest since the well known result of Birkhoff [1] on finite distributive lattices and posets. In the infinite case, an extension of this result and interesting properties can be found in Bonnet and Pouzet [2]. In the finite case, one of the most recent studies with an algorithmic point of view has been done by Bordat [14]. In this paper, we are only concerned by finite posets.

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. Theorem 1 gives a complete characterization of correlations between these two lattices assuming that the initial poset has at least a maximal element.

For the proof of Theorem 1, we need the following lemma, saying that the ideals grow by adding one element at a time:

Lemma 1 *Let $\widetilde{Q}$ be a poset,*
$\forall I, J \in I(Q),\ I \prec\!\!\!\cdot_{\widetilde{I(Q)}} J \iff I \subset J$ *and* $|J \setminus I| = 1$

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 $I \prec\!\!\!\cdot_{\widetilde{I(Q)}} J$, then $I \subset J$. If $J \setminus I = \emptyset$ then $I = J$ which contradicts $I \prec\!\!\!\cdot_{\widetilde{I(Q)}} J$. If $|J \setminus I| \geq 2$, let $x, y \in J \setminus I$ with $x \neq y$. Without loss of generality, we can assume that $y \not<_{\widetilde{Q}} x$, then $I <_{\widetilde{I(Q)}} I \cup \downarrow_{\widetilde{Q}} x] <_{\widetilde{I(Q)}} J$ which contradicts again $I \prec\!\!\!\cdot_{\widetilde{I(Q)}} J$. □

$J \setminus I$ is called the *label* of the edge $I \prec\!\!\!\cdot_{\widetilde{I(Q)}} J$.

Let $\widetilde{P}$ be a poset and x one of its maximal element. As one can see in Figure 4, $\widetilde{I(P)}$ is structured in three different parts. The upper part $IP(x)$ is formed with the ideals containing x: $IP(x) = \{I \in I(P) : I \cap \{x\} \neq \emptyset\}$. The medium part $I(x)$ is formed with the ideals containing the immediat predecessors of x and not containing x: $I(x) = \uparrow_{\widetilde{I(P)}} (\downarrow_{\widetilde{P}} x[) \setminus IP(x)$. $I(x)$ and $IP(x)$ are two isomorphic sublattices. The lower part is formed with the remaining ideals. This is formalized by Theorem 1.

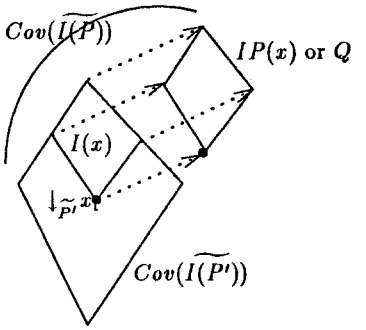

Fig. 4. Illustration of theorem 1

Theorem 1 *Let $\widetilde{P}$ be a poset, let $x \in Max(P, \widetilde{P})$, let $\widetilde{P'}$ be the subposet $P\widetilde{\setminus\{x\}}$ and*
$Cov(\widetilde{I(P')}) = (I(P'), E_{I(P')})$. Let $I(x) = \uparrow_{\widetilde{I(P')}}(\downarrow_{\widetilde{P}} x[)]$. Let $G(Q) = (Q, E_Q)$ be an acyclic directed graph isomorphic to $Cov(\widetilde{I(x)})$ by a map ϕ.
Then, the acyclic directed graph $G(Z) = (Z, E_Z)$ where $Z = Q \uplus I(P')$ and E_Z is defined by:

1. $\forall q_1, q_2 \in Q \times Q : q_1 q_2 \in E_Z \iff \phi(q_1)\phi(q_2) \in E_{I(P')}$
2. $\forall p, q \in I(P') \times Q : pq \in E_Z \iff q = \phi{-}1(p)$
3. $\forall p_1, p_2 \in I(P') \times I(P') : p_1 p_2 \in E_Z \iff p_1 p_2 \in E_{I(P')}$

is isomorphic to $Cov(\widetilde{I(P)})$ by the map φ:

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

Proof: Since $\widetilde{P'}$ is a subposet of $\widetilde{P}$ it is clear that $\varphi(Z) \subseteq I(P)$. For the same reason, for all $I \in I(P)$, if $x \notin I$ then $I \in I(P')$, otherwise $I \setminus \{x\} \in I(x)$ (moreover when $I_1 \neq I_2$ we have $I_1 \setminus \{x\} \neq I_2 \setminus \{x\}$). Thus $I(P) \subseteq \varphi(Z)$ and then φ is a bijection from Z to $I(P)$. It remains to show that φ is a morphism from $G(Z)$ to $Cov(\widetilde{I(P)})$.
Let us denote by $IP(x)$ the set $\{I \cup \{x\},\ I \in I(x)\}$ (remark that $I(P) = I(P') \uplus IP(x)$). $\forall K \in IP(x)$ we have $K \not\leq_{\widetilde{I(P)}} I$, $\forall I \in I(P')$ (since $x \notin I$). It is then clear that the subgraph of $G(Z)$ on $I(P')$ is isomorphic by the corresponding restriction of φ to the subgraph of $Cov(\widetilde{I(P)})$ on $I(P')$.
Let ψ be a map from $I(x)$ to $IP(x)$ such that $\psi(I) = I \cup \{x\}$. Since ψ is bijective and since $\forall I, J \in I(x),\ I \subset J \iff \psi(I) \subset \psi(J)$, the subgraphs of $Cov(\widetilde{I(P)})$ on $I(x)$ respectively on $IP(x)$ are isomorphic.
In order to conclude the proof it remains to study the edge connections between the subgraphs of $Cov(\widetilde{I(P)})$ on $I(P')$ and $IP(x)$. First we are going to show that $\forall I \in IP(x),\ \exists! I' \in I(P')$ such that $I' -<_{\widetilde{I(P)}} I$ and that $I' = I \setminus \{x\}$. By definition of $IP(x)$, $\forall I \in IP(x),\ I \setminus \{x\} \in I(P')$ thus by Lemma 1 $I' -<_{\widetilde{I(P)}} I$. For the unicity of I', assume that for $I \in IP(x)$ there exists $I'_1, I'_2 \in I(P')$ covered by I. Since $x \in I$ and x is neither in I'_1 nor in I'_2 then by Lemma 1 $I'_1 = I \setminus \{x\}$ and $I'_2 = I \setminus \{x\}$. Thus $I'_1 = I'_2$. It remains to show that we have no other edge connections: since $\widetilde{P'}$ is a subposet of $\widetilde{P}$ it is clear that $\forall I \in I(P') \setminus I(x)$, $\exists K \in I(x)$ such that $I <_{\widetilde{I(P')}} K$ and thus $I <_{\widetilde{I(P)}} K <_{\widetilde{I(P)}} K \cup \{x\}$.
□

To give an idea of the overall construction, let us take an example of poset $\widetilde{P}$, whose covering graph is given by Figure 3. We also suppose that elements are successively read in the order b, c, d, a, g, f, e which forms a linear extension of $\widetilde{P}$. For each element, we know its predecessor set.

The algorithm is illustrated in Figure 5. Black nodes denote nodes containing $\downarrow^{im}_{\widetilde{P}} x$ and which will be duplicated when incorporating a new event x. Doted edges denote 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 $\{\emptyset, \{b\}, \{b,c\}, \{b,d\}, \{b,c,d\}\}$ and the new incoming vertex for $\widetilde{P}$, 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: $label(y) = (label(x) \cup \{a\}) \setminus D(a)$.
2. add a new edge xy between unconnected vertices x and y checking that y was obtained from x.

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).

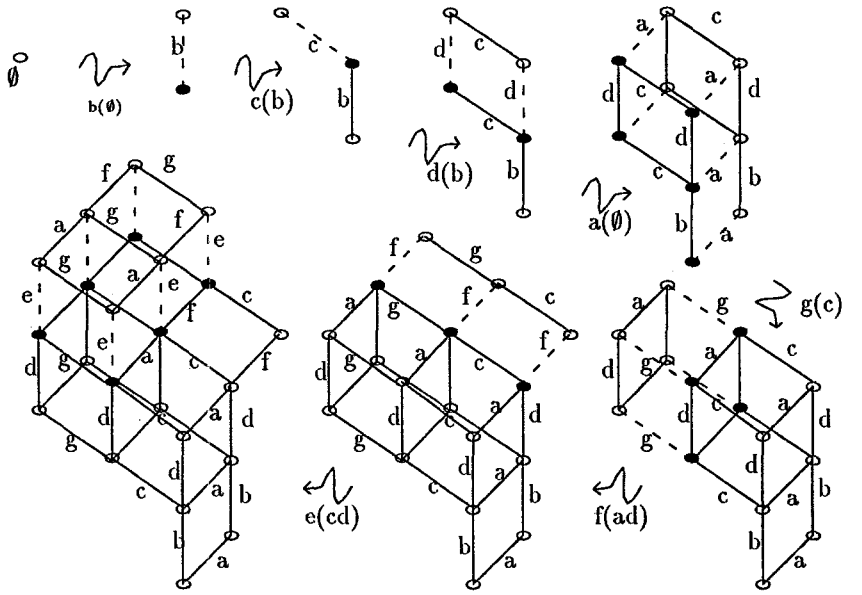

Fig. 5. On the fly computation of the state graph

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 (which extends the vector clock coding of causality) defined by:

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

This map embeds the lattice into the $(\mathbb{N}^k, \leq_{\mathbb{N}^k})$ lattice.

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

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

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

(iii) All maximal chains from I *to* J *in* $\widetilde{I(P)}$ *have length* $lg(I,J) = \sum_{i=1}^{k}(\Delta(J)[i] - \Delta(I)[i])$. $\forall i \in \{1,\ldots,k\}$, $\Delta(J)[i] - \Delta(I)[i] \geq 0$. *And there is at least one maximal chain.*

Proof: Obviously $(iii) \Longrightarrow (ii)$.

$(ii) \Longrightarrow (i)$: 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, $I = (I \cap (\bigcup_{1 \leq i \leq k} P_i)) \subseteq (J \cap (\bigcup_{1 \leq i \leq k} P_i)) = J$.

$(i) \Longrightarrow (iii)$: 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 $\mid J \mid = \mid I \mid + \alpha$. Since I and J are ideals, $\forall i \in \{1, \ldots, k\}$ we have $I \cap P_i \subseteq J \cap P_i$ and then, $\forall i \in \{1, \ldots, k\}$ we have $\Delta(I)[i] \leq_{I\!N} \Delta(J)[i]$ and thus $\Delta(J)[i] - \Delta(I)[i] \geq 0$. 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 thus graded. So all maximal chains have the same length. □

The "granulation algorithm"

Input:

(1) The transitive reduction of $\widetilde{P}$ with a chain decomposition [4] $\{P_i\}_{1 \leq i \leq k}$.

(2) For any $y \in P$, $\delta(y) = (\mid \downarrow_{\widetilde{P}} y] \cap P_i \mid)_{1 \leq i \leq k}$ (same definition as in paragraph 2.3) and $i(y)$ such that $y \in P_{i(y)}$.

(3) A vertex $x \in Max(P, \widetilde{P})$ and the set $D(x) = \downarrow_{\widetilde{P}}^{im} x$.

(4) When $P' = P \setminus \{x\}$ and $\widetilde{P}/_{P'} = \widetilde{P'}$, $Cov(\widetilde{I(P')})$, such that:

   (a) Each $y \in P'$ is directly related to its corresponding $\downarrow_{\widetilde{P'}} y]$ in $Cov(\widetilde{I(P')})$.

   (b) Each edge yz in $Cov(\widetilde{I(P')})$ 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 $Cov(\widetilde{I(P')})$ are stored ordered by increasing index of the chain their label belong to.

Body:

(1) Find $\downarrow_{\widetilde{P'}} D(x)]$ in $Cov(\widetilde{I(P')})$.

(2) Build a directed graph $G(Q)$ isomorphic to $Cov(\widetilde{I(\downarrow x[)})$, by a map ϕ (where $I(\downarrow x[) = \{I \in I(P'), \downarrow_{\widetilde{P'}} D(x)] \leq_{\widetilde{I(P')}} I\}$).

(3) For any $I \in I(\downarrow x[)$, create the directed edge $(I, \phi^{-1}(I))$ with label x and store this edge according to the storage order.

(4) Create a link between x and $\phi^{-1}(\downarrow_{\widetilde{P'}} D(x)])$.

Output:

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

   (a) Each $y \in P$ is directly related to its corresponding $\downarrow_{\widetilde{P}} y]$ in $Cov(\widetilde{I(P)})$.

   (b) Each edge yz in $Cov(\widetilde{I(P)})$ 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 $\widetilde{I(P)}$.

[4] The chain decomposition on $\widetilde{P}$ is an extension of the chain decomposition on $\widetilde{P'}$, that is: $\forall i$, $i \leq i \leq k$ such that $i \neq i(x)$ we have $\{P_i\} = \{P'_i\}$ and $\{P_{i(x)}\}$ is $\{P'_{i(x)}\}$ with x added as maximal element.

(c) Outgoing edges of any vertices in $Cov(\widetilde{I(P)})$ are stored ordered by increasing index of the chain their label belong to.

Theorem 2 *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 1. 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 $\Delta(\downarrow_{\widetilde{P'}} y) = \delta(y)$ and $\Delta(\downarrow_{\widetilde{P'}} D(x)) = \delta(x) - (0..1_{i(x)}..0)$. Then from proposition 1, we know there exists a chain in $Cov(\widetilde{I(P')})$ from $\downarrow_{\widetilde{P'}} y$ to $\downarrow_{\widetilde{P'}} D(x)$ with exactly $\Delta(\downarrow_{\widetilde{P'}} D(x))[i] - \Delta(\downarrow_{\widetilde{P'}} y)[i]$ edges belonging to P_i. Thus starting from $\downarrow_{\widetilde{P'}} y$, we choose an outgoing edge with label z and chain index j belonging to $Ind(y, D(x))$ where $Ind(y, D(x)) = \{i, \gamma(i) > 0 \ with \ \gamma(i) = \Delta(\downarrow_{\widetilde{P'}} D(x))[i] - \Delta(\downarrow_{\widetilde{P'}} y)[i]\}$. Let $\gamma(j) = \gamma(j) - 1$, by induction on z we arrive in $\downarrow_{\widetilde{P'}} D(x)$ 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 edges 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 [5].

Corollary 1 *Let $\widetilde{P}$ be a poset, $\widetilde{I(P)}$ can be computed in time complexity:*

$$O((\mid I(P) \mid + \mid P \mid^2)w(P)).$$

4 Conclusion

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. Furthermore, there is a need for the development of

[5] 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).

on–the–fly techniques which allows the trace analysis in parallel with its execution.

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.

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 quasi-linear complexity. Its on the fly characteristic and also its time complexity substantially improve the Cooper and Marzullo's contribution for detecting global predicates. The lattice seems also a good formal basis to prove specific algorithms for detecting restricted subclasses of global properties.

We have implemented our ideas in our favorite distributed environment Echidna [13]. Firstly, we have focussed our attention on visualization the covering relation and the state graph. Presently we are coupling 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, the french-israeli research cooperation and the research center of the french army (Celar).

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, 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. B. Charron-Bost, C. Delporte, and H. Fauconnier. *Local and Temporal Predicates in Distributed Systems.* Research report 92-36, LITP - Paris 7, 1992.
7. 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.
8. C. Diehl and C. Jard. Interval approximations of message causality in distributed executions. In Finkel and Jantzen, editors, *STACS*, pages 363–374, Springer–Verlag, LNCS 577, Cachan, february 1992.
9. 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.
10. Vijay K. Garg and Brian Waldecker. *Detection of Unstable Predicates in Distributed Programs.* Technical Report TR-92-07-82, University of Texas at Austin, march 1992.
11. P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety. In *Computer Aided Verification, LNCS 575*, pages 332–342, Aalborg, Denmark., 1991.
12. 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.
13. C. Jard and J.-M. Jézéquel. ECHIDNA, an Estelle-compiler to prototype protocols on distributed computers. *Concurrency Practice and Experience*, 4(5):377–397, August 1992.
14. Bordat J.P. Sur l'algorithmique combinatoire d'ordres finis. Thèse de doctorat d'état, USTL Montpellier, 1992.
15. L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
16. T. Leblanc and J. Mellor-Crummey. Debugging parallel programs with instant replay. *IEEE Transactions on Computers*, C-36(4):471–482, April 1987.
17. 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, apr 1991.
18. Hurfin M., N. Plouzeau, and M. Raynal. *Détection de séquences atomiques de prédicats locaux dans les exécutions réparties.* Research report , IRISA, 1993.
19. 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.
20. R. Schwarz and F. Mattern. *Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail.* Technical Report 215/91, University of Kaiserslautern, 1991.