Faster Possibility Detection by Combining Two Approaches*

Scott D. Stoller and Fred B. Schneider

Dept. of Computer Science, Cornell University, Ithaca, NY 14853, USA.
stoller@cs.cornell.edu, *fbs@cs.cornell.edu*

Abstract. A new algorithm is presented for detecting whether a particular computation of an asynchronous distributed system satisfies **Poss** Φ (read "possibly Φ"), meaning the system could have passed through a global state satisfying Φ. Like the algorithm of Cooper and Marzullo, Φ may be any global state predicate; and like the algorithm of Garg and Waldecker, **Poss** Φ is detected quite efficiently if Φ has a certain structure. The new algorithm exploits the structure of some predicates Φ not handled by Garg and Waldecker's algorithm to detect **Poss** Φ more efficiently than is possible with any algorithm that, like Cooper and Marzullo's, evaluates Φ on every global state through which the system could have passed. A second algorithm is also presented for off-line detection of **Poss** Φ. It uses Strassen's scheme for fast matrix multiplication. The intrinsic complexity of off-line and on-line detection of **Poss** Φ is discussed.

1 Introduction

A *history* of a distributed system can be modeled as a sequence of events in their order of occurrence. Since execution of a particular sequence of events leaves the system in a well-defined global state, a history uniquely determines a sequence of global states through which the system has passed. Unfortunately, in an asynchronous distributed system, no process can determine the order in which events on different processors actually occurred. Therefore, no process can determine the sequence of global states through which the system passed. This leads to an obvious difficulty for detecting whether a global state predicate (hereafter simply called a "predicate") held.

Cooper and Marzullo's solution to this difficulty involves two modalities, which we denote by **Poss** (read "possibly") and **Def** (read "definitely") [CM91]. These modalities are based on logical time [Lam78] as embodied in the *happened-before* relation $\rightarrow$, a partial order on events that reflects causal dependencies. A history of an asynchronous distributed system can be approximated by a *computation*, which is a set of the events that occurred together with their happened-before relation. Happened-before is useful for detection algorithms because, using

* This material is based on work supported in part by NSF/DARPA Grant No. CCR-9014363, NASA/DARPA grant NAG-2-893, and AFOSR grant F49620-94-1-0198. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not reflect the views of these agencies.

vector clocks [Fid88, Mat89], it—hence the computation—can be determined by processes in the system.

Happened-before is not a total order, so it does not uniquely determine the history. But it does restrict the possibilities. Histories *consistent* with a computation are exactly those sequences that correspond to total orders containing the happened-before relation (*i.e.*, sequences such that for all events e and e', if $e \rightarrow e'$, then e occurs before e' in the sequence). A computation satisfies **Poss** Φ iff, in *some* history consistent with that computation, the system passes through a global state satisfying Φ. A computation satisfies **Def** Φ iff, in *all* histories consistent with that computation, the system passes through a global state satisfying Φ.

Cooper and Marzullo give centralized algorithms for detecting **Poss** Φ and **Def** Φ for an arbitrary predicate Φ [CM91]. A stub at each process reports the local states of that process to a central monitor. The central monitor incrementally constructs a lattice to represent the set of histories consistent with the computation. A straightforward search of the lattice reveals whether the computation satisfies **Poss** Φ or **Def** Φ.

The generality of Cooper and Marzullo's approach is attractive. Unfortunately, their algorithms can be expensive. In a system of N processes, the worst-case size of the constructed lattice is $\Theta(S^N)$, where S is the maximum number of steps taken by a single process.[2] This worst case comes from the (exponential) number of histories consistent with a computation in which there is little communication. Any detection algorithm that constructs the entire lattice—whether it uses the method in [CM91, MN91] or the more efficient schemes in [DJR93, JMN95]—has worst-case time complexity that is at least linear in the size of the lattice. Thus, Cooper and Marzullo's algorithms for detecting **Poss** Φ and **Def** Φ have worst-case time complexity $\Omega(S^N)$.

Because the time needed to construct the lattice can be prohibitive, researchers have sought faster detection algorithms. One approach has been to change the problem—for example, detecting a different modality [FR94] or assuming that the system is partially synchronous [MN91]. Another approach has been to restrict the problem and develop efficient algorithms for detecting only certain classes of predicates [GW92, GW94, TG94].

Our work is inspired by an algorithm of Garg and Waldecker for a restricted problem [GW94]. Their algorithm detects **Poss** Φ only for Φ a Boolean combination of local predicates, where a *local predicate* is defined to be one that depends on the state of a single process. The worst-case time complexity of their algorithm is $\Theta(N^2S)$, which is significantly smaller than the worst-case size of the lattice. The efficiency of Garg and Waldecker's algorithm makes it ideal when the property to be detected can be expressed as a Boolean combination of local predicates. However, many properties can not be so expressed.

In this paper, we show how to combine Garg and Waldecker's approach with

[2] We use standard "order-of-magnitude" symbols O, Ω, and Θ. For definitions, see [BDG88, sec. 2.2], whose only idiosyncrasy is using Ω_∞ for the operator commonly denoted Ω.

any algorithm that constructs the lattice to detect **Poss** Φ. The result is a new algorithm that has the best features of both and improves on each. Our algorithm exploits the structure of some predicates Φ not handled by Garg and Waldecker's algorithm to detect **Poss** Φ more efficiently than is possible with an algorithm that constructs the entire lattice. In addition, our algorithm can detect **Poss** Φ for any predicate Φ. And, like Garg and Waldecker's algorithm, our algorithm detects Boolean combinations of local predicates in time linear in S.

As an illustration, consider the predicate

$$\mathcal{P} \triangleq \phi_{12}(x_1, x_2) \wedge \phi_3(x_3), \tag{1}$$

where variable x_i is a state component of process i. Garg and Waldecker's algorithm is inapplicable here, because ϕ_{12} is not local. And, since $N = 3$, the worst-case time complexity for constructing the entire lattice is $\Theta(S^3)$. Our algorithm detects **Poss** $\mathcal{P}$ with worst-case time complexity $\Theta(S^2)$ by decomposing the problem into multiple detection problems, each solvable using Garg and Waldecker's algorithm. To see how our algorithm works, note that if the state of process 1 is frozen, then each conjunct of $\mathcal{P}$ depends on the state of exactly one of the remaining processes, so each conjunct is effectively a local predicate. Thus, for each of the $O(S)$ states of process 1, **Poss** $\mathcal{P}$ can be detected with time complexity $O(S)$ using Garg and Waldecker's algorithm. It follows that for **Poss** $\mathcal{P}$, the worst-case time complexity of our algorithm is $\Theta(S^2)$.

The remainder of the paper is organized as follows. Section 2 introduces our model of distributed systems. In Section 3, we specify and give algorithms for off-line and on-line detection of **Poss** Φ. Example applications of the algorithms are given in Section 4. Section 5 discusses how to use Strassen's matrix-multiplication algorithm for off-line detection of **Poss** Φ when Φ has certain structure. This algorithm is faster than the more general off-line one described in Section 3, but we argue that fast matrix-multiplication routines, hence detection algorithms based on them, probably cannot be made on-line. In Section 6, we show that detecting **Poss** Φ is NP-complete, even for conjunctions where each conjunct depends on the states of at most two processes. We also comment on the difficulty of proving lower bounds for detecting **Poss** Φ.

2 System Model and Notation

A (local) state of a process is a mapping from identifiers to values. A history of a single process is represented as a sequence of that process's states. The α^{th} element of a sequence c is denoted $c[\alpha]$, and the set containing exactly the elements in a sequence c is denoted $\mathcal{U}(c)$. Let $[m..n]$ denote the set of integers from m to n, inclusive. We use integers $[1..N]$ as process names.

A computation c is represented as histories $c_1, \ldots, c_N$ of the constituent processes, together with a *happened-before* relation $\rightarrow$ that is a relation on local states instead of events [GW94]. In particular, define $\rightarrow$ to be the smallest transitive relation on $\bigcup_{i=1}^{N} \mathcal{U}(c_i)$ such that

1. $(\forall i \in [1..N] : (\forall \alpha \in [1..(|c_i| - 1)] : c_i[\alpha] \to c_i[\alpha + 1]))$.
2. For all s and s' in $\bigcup_{i=1}^{N} \mathcal{U}(c_i)$, if the event immediately following s is the sending of a message and the event immediately preceding s' is the reception of that message, then $s \to s'$.

We always use S to denote $\max(|c_1|, \ldots, |c_N|)$. We assume each process has a distinguished variable τ such that for each local state s, $s(\tau)$ is a *vector timestamp* [Mat89] and for all local states s and s', $s(\tau) < s'(\tau)$ iff $s \to s'$.

A *state* of a distributed system is a collection of local states; we represent such a collection as a function from process names to local states. Thus, for a state g, the (local) state of process i is $g(i)$, and the value of variable x at process i is $g(i)(x)$. The domain of a state g is denoted $\mathrm{dom}(g)$. A *global state* specifies the state of every process; thus, it is a state with domain $[1..N]$. The set of global states of a computation c is denoted $GS(c)$.

Two local states s and s' are *concurrent*, denoted $s \parallel s'$, iff neither happened before the other: $s \parallel s' \triangleq s \not\to s' \wedge s' \not\to s$. Two states g and g' are *concurrent*, denoted $g \parallel g'$, iff each local state in g is concurrent with all local states in g':

$$g \parallel g' \triangleq \bigwedge_{\substack{i \in \mathrm{dom}(g) \\ j \in \mathrm{dom}(g')}} g(i) \parallel g'(j). \qquad (2)$$

A state g is *consistent* iff its constituent local states are pairwise concurrent. The set of consistent global states of a computation c, denoted $CGS(c)$, is therefore characterized by[3]

$$g \in CGS(c) \quad \text{iff} \quad g \in GS(c) \wedge (\forall i, j \in \mathrm{dom}(g) : i \neq j \Rightarrow g(i) \parallel g(j)). \qquad (3)$$

Given a computation c and a set $F \subseteq [1..N]$ of processes, the *restriction of c to F*, denoted $c{\downarrow}F$, is the histories of only those processes in F together with the restriction of $\to$ to $\bigcup_{i \in F} \mathcal{U}(c_i)$. The restriction of a state g to F, denoted $g{\downarrow}F$, is the state obtained from g by restricting the domain to be $\mathrm{dom}(g) \cap F$. An overbar denotes complementation: $\overline{F} \triangleq [1..N] \setminus F$.

We regard predicates as Boolean-valued functions. Thus, $\Phi(g)$ is the truth value of predicate Φ in state g. When writing predicates in terms of state variables (as in (1)), we subscript each variable with the name of the process to which it belongs. For example, x_i is a component of the local state of process i. The set of processes on whose local states a predicate Φ depends is denoted $\Pi(\Phi)$. A predicate is defined to be *n-local* if $|\Pi(\Phi)| \leq n$, meaning Φ depends on the local states of at most n processes. Given a predicate Φ and a set F of processes, we say that Φ is *n-local for F* if $|\Pi(\Phi) \cap F| \leq n$, meaning Φ depends on the local states of at most n processes in F.

[3] $CGS(c)$ could also be defined directly in terms of histories: $g \in CGS(c)$ iff the system passes through g in some history consistent with c. The definition of $CGS(c)$ in terms of $\parallel$ is more convenient for reasoning about detection algorithms, so we take it as primary.

3 Detection Algorithm

3.1 Specification

The formal definition of $\mathbf{Poss}\,\Phi$ is [CM91]

$$c \models \mathbf{Poss}\,\Phi \quad \text{iff} \quad (\exists g : g \in CGS(c) \wedge \Phi(g)). \tag{4}$$

The off-line detection problem for **Poss** is: given a computation c and a predicate Φ, determine whether $c \models \mathbf{Poss}\,\Phi$ holds.

In the on-line problem, the detection algorithm is initially given the predicate but not the computation. Local states arrive at the monitor one at a time. For each process, the local states of that process arrive in the order they occurred. However, there is no constraint on the relative arrival order of local states of different processes. Detection must be announced as soon as local states comprising a CGS satisfying Φ have arrived.

3.2 Off-line Algorithm

The basis of our approach is to decompose the detection problem by partitioning the set of processes. The following lemma shows how $CGS(c)$ decomposes.

Lemma 1. For all computations c, all $F \subseteq [1..N]$, and all global states g of c, $g \in CGS(c) \quad \text{iff} \quad (g{\downarrow}F \in CGS(c{\downarrow}F)) \wedge (g{\downarrow}\overline{F} \in CGS(c{\downarrow}\overline{F})) \wedge ((g{\downarrow}F) \parallel (g{\downarrow}\overline{F}))$.

Proof.

$$
\begin{array}{ll}
 & g \in CGS(c) \\
= & \qquad \langle\!\langle \text{ Definition (3) of } CGS(c) \rangle\!\rangle \\
 & g \in GS(c) \wedge (\forall i \in [1..N] : (\forall j \in [1..N] : i \neq j \Rightarrow g(i) \parallel g(j))) \\
= & \qquad \langle\!\langle \text{ Definition of } {\downarrow}F,\ [1..N] = F \cup \overline{F}, \text{ and Range Partitioning Law for } \forall \rangle\!\rangle \\
 & (g{\downarrow}F \in GS(c{\downarrow}F)) \wedge (\forall i \in F : (\forall j \in F : i \neq j \Rightarrow g(i) \parallel g(j))) \\
 & \wedge (g{\downarrow}\overline{F} \in GS(c{\downarrow}\overline{F})) \wedge (\forall i \in \overline{F} : (\forall j \in \overline{F} : i \neq j \Rightarrow g(i) \parallel g(j))) \\
 & \wedge (\forall i \in F : (\forall j \in \overline{F} : i \neq j \Rightarrow g(i) \parallel g(j))) \\
 & \wedge (\forall i \in \overline{F} : (\forall j \in F : i \neq j \Rightarrow g(i) \parallel g(j))) \\
= & \qquad \langle\!\langle \text{ Definition (3) of } CGS, \text{ and } i \in F \wedge j \in \overline{F} \text{ implies } i \neq j \rangle\!\rangle \\
 & (g{\downarrow}F \in CGS(c{\downarrow}F)) \wedge (g{\downarrow}\overline{F} \in CGS(c{\downarrow}\overline{F})) \\
 & \wedge (\forall i \in F : (\forall j \in \overline{F} : g(i) \parallel g(j))) \wedge (\forall i \in \overline{F} : (\forall j \in F : g(i) \parallel g(j))) \\
= & \qquad \langle\!\langle \text{ By symmetry of } \parallel, \text{ the last two conjuncts are equivalent } \rangle\!\rangle \\
 & (g{\downarrow}F \in CGS(c{\downarrow}F)) \wedge (g{\downarrow}\overline{F} \in CGS(c{\downarrow}\overline{F})) \wedge (\forall i \in F : (\forall j \in \overline{F} : g(i) \parallel g(j))) \\
= & \qquad \langle\!\langle \text{ Definition (2) of } \parallel \rangle\!\rangle \\
 & (g{\downarrow}F \in CGS(c{\downarrow}F)) \wedge (g{\downarrow}\overline{F} \in CGS(c{\downarrow}\overline{F})) \wedge ((g{\downarrow}F) \parallel (g{\downarrow}\overline{F})) \qquad \Box
\end{array}
$$

To decompose $\mathbf{Poss}\,\Phi$, we define a predicate that is a variant of Φ specialized with respect to the states of some processes. Given a state g_1 with domain F, let Φ_{g_1} denote the following predicate on states g_2:

$$\Phi_{g_1}(g_2) \triangleq \Phi(g_1 \oplus g_2) \wedge (g_1 \parallel g_2) \tag{5}$$

```
for each g1 in CGS(c↓F) do
    if c↓F̄ ⊨ Poss Φ_g1 then                                  (*)
      return("detected")
    fi
rof
return("not detected")
```

Fig. 1. Possibility Detection Decomposition Algorithm (PDDA).

where $g_1 \oplus g_2$ is the global state whose values on processes in $\mathrm{dom}(g_1)$ are given by g_1, and whose values on other processes are given by g_2.

Lemma 2. For all computations c, all $F \subseteq [1..N]$, and all predicates Φ,

$$c \models \mathbf{Poss}\,\Phi \quad \text{iff} \quad (\exists g_1 : (g_1 \in CGS(c{\downarrow}F)) \wedge (c{\downarrow}\overline{F} \models \mathbf{Poss}\,\Phi_{g_1})).$$

Proof.

$$\begin{array}{ll}
& c \models \mathbf{Poss}\,\Phi \\
= & \qquad \langle\!\langle \text{ Definition (4) of } c \models \mathbf{Poss}\,\Phi \,\rangle\!\rangle \\
& (\exists g : g \in CGS(c) \wedge \Phi(g)) \\
= & \qquad \langle\!\langle \text{ Lemma 1 } \rangle\!\rangle \\
& (\exists g : (g{\downarrow}F \in CGS(c{\downarrow}F)) \wedge (g{\downarrow}\overline{F} \in CGS(c{\downarrow}\overline{F})) \wedge ((g{\downarrow}F) \parallel (g{\downarrow}\overline{F})) \wedge \Phi(g)) \\
= & \qquad \langle\!\langle \text{ Take } g_1 = g{\downarrow}F \text{ and } g_2 = g{\downarrow}\overline{F} \,\rangle\!\rangle \\
& (\exists g_1, g_2 : (g_1 \in CGS(c{\downarrow}F)) \wedge (g_2 \in CGS(c{\downarrow}\overline{F})) \wedge (g_1 \parallel g_2) \wedge \Phi(g_1 \oplus g_2)) \\
= & \qquad \langle\!\langle \text{ Definitions of } \Phi_{g_1} \text{ and } \mathbf{Poss} \,\rangle\!\rangle \\
& (\exists g_1 : (g_1 \in CGS(c{\downarrow}F)) \wedge (c{\downarrow}\overline{F} \models \mathbf{Poss}\,\Phi_{g_1}))
\end{array}$$

□

This lemma suggests the algorithm in Figure 1.[4] Any algorithms for computing $CGS(c{\downarrow}F)$ and $c{\downarrow}\overline{F} \models \mathbf{Poss}\,\Phi_{g_1}$ can be used as subroutines. PDDA is correct for all choices of F, but it is faster than evaluating Φ on every element of $CGS(c)$ only if F is chosen in a way that facilitates computation of $c{\downarrow}\overline{F} \models \mathbf{Poss}\,\Phi_{g_1}$. In particular, if Φ_{g_1} is a conjunction of predicates that are each 1-local for $\overline{F}$, in which case we say that F is a *fixed set* for Φ, then $\mathbf{Poss}\,\Phi_{g_1}$ can be detected efficiently using Garg and Waldecker's algorithm. Thus, if F is a fixed set for Φ, then fixing the states of processes in F yields a predicate in which each conjunct depends on the state of at most one of the remaining processes.

[4] It is natural to consider extending our ideas to detection of $\mathbf{Def}\,\Phi$ and look for a way to decompose detecting $\mathbf{Def}\,\Phi$ into easier subproblems. However, this does not seem promising. Detecting $\mathbf{Def}\,\Phi$ is equivalent to determining whether the set of consistent global states satisfying Φ is a $(\bot, \top)$-vertex separator for the lattice of consistent global states. Being a vertex separator is a rather global property of the lattice, so decomposing it seems difficult.

To express this condition more explicitly, assume Φ has the form $\Phi \triangleq \bigwedge_{\alpha=1}^{n} \phi_\alpha$ for $n \geq 1$. Since we allow $n = 1$, this entails no loss of generality. We consider the two pieces of Φ_{g_1} separately. The conjuncts in $\Phi(g_1 \oplus g_2)$ are, by definition, 1-local for $\overline{F}$ iff

$$(\forall \alpha \in [1..n] : |\Pi(\phi_\alpha) \cap \overline{F}| \leq 1). \tag{6}$$

The conjuncts in $g_1 \parallel g_2$ are 1-local for $\overline{F}$ independently of F, because, by inspection of the definition (2) of $\parallel$, each conjunct depends on exactly one local state of g_1 and exactly one local state of g_2, and by definition of g_1, $\mathrm{dom}(g_1) = F$, so processes in $\mathrm{dom}(g_1)$ are not in $\overline{F}$. Thus, F is a fixed set for Φ iff condition (6) holds.

For example, consider predicate $\mathcal{P}$ in (1). Take $F = \{1\}$. Expanding the definition gives

$$\begin{aligned}\mathcal{P}_{g_1}(g_2) = \;& \phi_{12}(g_1(1)(x_1), g_2(2)(x_2)) \wedge \phi_3(g_2(3)(x_3)) \\ & \wedge g_1(1) \not\rightarrow g_2(2) \wedge g_2(2) \not\rightarrow g_1(1) \\ & \wedge g_1(1) \not\rightarrow g_2(3) \wedge g_2(3) \not\rightarrow g_1(1).\end{aligned}$$

Each conjunct of $\mathcal{P}_{g_1}$ depends on at most one process in $\overline{F}$, so F is a fixed set for $\mathcal{P}$.

A fixed set exists for every Φ—just take $F = [1..(N-1)]$. However, this choice of F is not always the best. The following analysis shows that a minimum-sized fixed set should be used. The set $CGS(c\!\downarrow\! F)$ can be built with worst-case time complexity $O(|F| \cdot |CGS(c\!\downarrow\! F)| + |F|^3 S^2)$ using the algorithm in [DJR93], or slightly faster using the algorithm in [JMN95]. For each state g_1 in $CGS(c\!\downarrow\! F)$, Garg and Waldecker's algorithm detects $c\!\downarrow\! \overline{F} \models \mathbf{Poss}\, \Phi_{g_1}$ with worst-case time complexity $O(|\overline{F}|^2 S)$. Let $\mathrm{PDDA_{GW}}$ denote the specialized version of PDDA that always uses Garg and Waldecker's algorithm to detect $\mathbf{Poss}\, \Phi_{g_1}$; note that $\mathrm{PDDA_{GW}}$ requires F to be a fixed set for Φ. The cost of $\mathrm{PDDA_{GW}}$ is the cost of building $CGS(c\!\downarrow\! F)$ plus the cost of running Garg and Waldecker's algorithm $|CGS(c\!\downarrow\! F)|$ times. Thus, the worst-case time complexity of $\mathrm{PDDA_{GW}}$ is

$$O((|\overline{F}|^2 S + |F|)|CGS(c\!\downarrow\! F)| + |F|^3 S^2),$$

not including the cost of finding a fixed set (which is discussed in the next subsection). Note that the cost of finding a fixed set depends on the size of the formula and therefore is dominated by the cost analyzed above.

Since the worst-case size of $CGS(c\!\downarrow\! F)$ is $\Theta(S^{|F|})$, the worst-case time complexity of $\mathrm{PDDA_{GW}}$ is $\Theta(|\overline{F}|^2 S^{|F|+1} + |F| S^{|F|} + |F|^3 S^2)$. Thus, for fixed N and F, $\mathrm{PDDA_{GW}}$ runs in $O(S^{|F|+1})$ time. This is asymptotically less than the worst-case size $\Theta(S^N)$ of $CGS(c)$ whenever $|F| < N - 1$.

3.3 Finding a Fixed Set

We have not given an algorithm for finding minimum-sized fixed sets. The linear-time reductions in the following theorem show that finding a minimum-sized fixed set for a formula is equivalent to finding a minimum-sized vertex cover for a graph, a well-known NP-complete problem. If N is small, an exact solution can be found by exhaustive search; otherwise, an approximation algorithm can be used [GJ79, pp. 133-134].

Theorem 3. The problem of finding a minimum-sized fixed set is NP-complete.

Proof. We give linear-time reductions in both directions between finding a fixed set for a formula and finding a vertex cover for an undirected graph. Since finding a minimum-sized vertex cover is NP-complete [GJ79], it follows that finding a minimum-sized fixed set is also NP-complete.

Given an instance $\bigwedge_{\alpha=1}^{n} \phi_\alpha$ of finding a fixed set, define the edges of an undirected graph $G' \triangleq ([1..N], E')$ by $E' \triangleq \{\{i,j\} \mid i \neq j \wedge (\exists \alpha \in [1..n] : \{i,j\} \subseteq \Pi(\phi_\alpha))\}$. The following proof shows that (6) is equivalent to the definition of vertex cover. Thus, $F \subseteq [1..N]$ is a fixed set for $\bigwedge_{\alpha=1}^{n} \phi_\alpha$ iff F is a vertex cover for G'.

$$
\begin{array}{ll}
& (\forall \alpha \in [1..n] : |\Pi(\phi_\alpha) \cap \overline{F}| \leq 1) \\
= & \qquad \langle\!\langle \text{Definitions of intersection and cardinality} \rangle\!\rangle \\
& (\forall \alpha \in [1..n] : \neg(\exists i,j \in \Pi(\phi_\alpha) : i \neq j \ \wedge \ \{i,j\} \subseteq \overline{F})) \\
= & \qquad \langle\!\langle \text{De Morgan's Laws} \rangle\!\rangle \\
& (\forall \alpha \in [1..n] : (\forall i,j \in \Pi(\phi_\alpha) : i \neq j \Rightarrow \{i,j\} \not\subseteq \overline{F})) \\
= & \qquad \langle\!\langle \text{Definition of subset} \rangle\!\rangle \\
& (\forall \alpha \in [1..n] : (\forall i,j \in \Pi(\phi_\alpha) : i \neq j \Rightarrow i \in F \vee j \in F)) \\
= & \qquad \langle\!\langle \text{Definition of subset} \rangle\!\rangle \\
& (\forall \alpha \in [1..n] : (\forall i,j \in [1..N] : i \neq j \ \wedge \ \{i,j\} \subseteq \Pi(\phi_\alpha) \Rightarrow i \in F \vee j \in F)) \\
= & \qquad \langle\!\langle \text{If } x \text{ is not free in } q,\ (\forall x : p \Rightarrow q) \equiv ((\exists x : p) \Rightarrow q) \rangle\!\rangle \\
& (\forall i,j \in [1..N] : i \neq j \ \wedge \ (\exists \alpha \in [1..n] : \{i,j\} \subseteq \Pi(\phi_\alpha)) \Rightarrow i \in F \vee j \in F) \\
= & \qquad \langle\!\langle \text{Definition of } E' \rangle\!\rangle \\
& (\forall \{i,j\} \in E' : i \in F \vee j \in F)
\end{array}
$$

The last formula is the standard definition of a vertex cover.

Given an undirected graph $G = ([1..N], E)$, the corresponding instance of finding a fixed set is $\Phi(g) \triangleq \bigwedge_{\{i,j\} \in E'} \phi_{ij}(g(i), g(j))$, where $E' \triangleq \{\{i,j\} \in E \mid i \neq j\}$. Note that $F \cup \{i \in [1..N] \mid \{i,i\} \in E\}$ is a vertex cover for G iff F is a vertex cover for $G' \triangleq ([1..N], E')$. The above proof, read from bottom to top, and with the bottom hint changed to "Definition of Φ", shows that F is a vertex cover for G' iff F is a fixed set for Φ. □

3.4 The Benefit of Disjunctive Normal Form

The work needed to detect **Poss** Φ sometimes can be reduced by transforming Φ into a logically equivalent formula with a smaller minimum-sized fixed set. We show below that this is accomplished by putting Φ in disjunctive normal form (DNF), a canonical form where disjunctions are the outermost operators. Let $DNF(\Phi)$ denote the DNF for Φ. For example, for

$$\mathcal{D} \triangleq (\phi_{12}(x_1, x_2) \vee \phi_3(x_3)) \wedge \phi_4(x_4),$$

$DNF(\mathcal{D})$ is $(\phi_{12}(x_1, x_2) \wedge \phi_4(x_4)) \vee (\phi_3(x_3) \wedge \phi_4(x_4))$.

To see the benefit of putting a formula in DNF, first note that **Poss** distributes over disjunction:

Lemma 4. For all computations c and all predicates $\phi_1, \ldots, \phi_n$,

$$c \models \mathbf{Poss} \bigvee_{\alpha=1}^{n} \phi_\alpha \quad \text{iff} \quad \bigvee_{\alpha=1}^{n} (c \models \mathbf{Poss}\, \phi_\alpha).$$

Proof. See [GW94]. □

Thus, each disjunct of a formula can be detected separately. To describe how this fact is reflected in the complexity, we define a function f on formulas by: if Φ is a disjunction $\bigvee_{\alpha=1}^{n} \phi_\alpha$, then $f(\Phi)$ is $\max(f(\phi_1), \ldots, f(\phi_n))$; otherwise, $f(\Phi)$ is the size of a minimum-sized fixed set for Φ. By detecting each disjunct of Φ separately using PDDA$_{\mathrm{GW}}$, $\mathbf{Poss}\,\Phi$ can be detected in $O(S^{f(\Phi)+1})$ time. Now we demonstrate the benefit of DNF.

Lemma 5. For all formulas Φ, $f(DNF(\Phi)) \leq f(\Phi)$.

Proof. Structural induction on formulas. □

Theorem 6. Among all formulas equivalent to Φ using rules of propositional calculus, $DNF(\Phi)$ has the minimal value of f.

Proof. Let $DNF(\Phi) = \bigvee_{\alpha=1}^{n} \phi_\alpha$. Suppose the theorem is false. Then there is some formula Φ' equivalent to Φ using rules of propositional calculus, such that $f(\Phi') < f(DNF(\Phi))$. Since Φ and Φ' are equivalent, $DNF(\Phi)$ equals $DNF(\Phi')$, so $f(DNF(\Phi)) = f(DNF(\Phi'))$, and therefore the preceding inequality contradicts Lemma 5. □

Thus, the best complexity of PDDA$_{\mathrm{GW}}$ obtainable using propositional manipulation of Φ is achieved by forming $DNF(\Phi)$ and detecting its disjuncts separately. Consider, for example, formula $\mathcal{D}$ defined above. Any fixed set for $\mathcal{D}$ must contain at least two of the three processes mentioned in the first conjunct, so $f(\mathcal{D}) = 2$. By similar reasoning, minimum-sized fixed sets for the first and second disjuncts of $DNF(\mathcal{D})$ have size 1 and 0, respectively, so $f(DNF(\mathcal{D})) = 1$.

3.5 Enhancements

This subsection describes enhancements that speed up PDDA and PDDA$_{\mathrm{GW}}$ in some cases but do not change the worst-case complexity.

Fixed Conjuncts. A conjunct ϕ with $\Pi(\phi) \subseteq F$ is called a *fixed conjunct for F*. If Φ contains a fixed conjunct ϕ for F, then for each g_1 in $CGS(c{\downarrow}F)$, the enhanced algorithm first evaluates $\phi(g_1)$, then evaluates $c{\downarrow}\overline{F} \models \mathbf{Poss}\,\Phi_{g_1}$ only if $\phi(g_1)$ holds. More formally, the condition in line $(*)$ of PDDA in Figure 1 is replaced with $\phi(g_1)$ && $(c{\downarrow}\overline{F} \models \mathbf{Poss}\,\Phi_{g_1})$, where && is short-circuiting conjunction (as in C). For example, consider the predicate

$$\mathcal{Q} \triangleq (x_2 > 0) \wedge (x_4 > 0) \wedge (x_1 + x_2 < 4) \wedge (x_3 + x_4 < 5).$$

A minimum-sized fixed set for $\mathcal{Q}$ is $F = \{2, 4\}$, so $\mathcal{Q}$ can be detected in $O(S^3)$ time. The first two conjuncts of $\mathcal{Q}$ are fixed conjuncts for F, so if x_2 or x_4 is frequently non-positive, this technique will significantly speed up the detection.

It may be feasible to introduce new fixed conjuncts, regardless of whether Φ contains any. Simple predicate-logic reasoning shows that $\Phi(g) \equiv \Phi(g) \wedge (\exists g_2 \in GS_{\overline{F}} : \Phi((g{\downarrow}F) \oplus g_2))$, where $GS_{\overline{F}}$ is the set of all states with domain $\overline{F}$. The new conjunct is, by construction, a fixed conjunct for F. If this new conjunct can be simplified, then it can be used as described above. For example, another minimum-sized fixed set for $\mathcal{Q}$ is $F' = \{1, 3\}$. $\mathcal{Q}$ contains no fixed conjuncts for F'. Introducing a new conjunct as described above and simplifying yields the equivalent predicate

$$\hat{\mathcal{Q}} \triangleq (x_1 < 4) \wedge (x_3 < 5) \wedge (x_2 > 0) \wedge (x_4 > 0) \wedge (x_1 + x_2 < 4) \wedge (x_3 + x_5 < 5).$$

The first two conjuncts of $\hat{\mathcal{Q}}$ (*i.e.*, the new conjuncts) are fixed conjuncts for F'. Whether it is better to detect $\mathcal{Q}$ using fixed set F or to detect $\hat{\mathcal{Q}}$ using fixed set F' depends on the application. For example, if x_2 and x_4 are usually positive, and x_1 and x_3 are usually large, then the latter is preferable.

Constraining the Search of $c{\downarrow}\overline{F}$. Given a state g_1 of $c{\downarrow}F$, one can compute for each process in $\overline{F}$ the maximal range of local states of that process that are concurrent with g_1 [BM93, sec. 4.14.3]. This information can be exploited in PDDA by restricting the search of $c{\downarrow}\overline{F}$ so that only local states in these ranges are examined.

3.6 On-line Algorithm

The algorithms in [DJR93, JMN95] for computing $CGS(c)$ have on-line versions with the same time complexities as given above. The same is true of Garg and Waldecker's algorithm. It is straightforward to use the on-line versions of these algorithms to obtain on-line versions of PDDA and $\mathrm{PDDA_{GW}}$ having the same time complexities as above.

4 Examples of Applications

Load Balancing. Consider a system with three processors. Processors 1 and 2 are servers. Processor 3 is used as a server when the load is heavy and for other tasks when the load is light. $\mathrm{PDDA_{GW}}$ can be used to detect the conditions for switching processor 3 between server mode and "other tasks" mode. The conditions are

$$\begin{aligned} \mathit{Server} &\triangleq (\mathit{load}_1 + \mathit{load}_2) > a \wedge \mathit{avail}_3 \wedge \neg \mathit{srvr}_3 \\ \mathit{Other} &\triangleq (\mathit{load}_1 + \mathit{load}_2) < a \wedge \mathit{srvr}_3 \end{aligned}$$

where load_i is the load on processor i, a is a constant, srvr_3 indicates whether processor 3 is in server mode, and when processor 3 is not in server mode, avail_3 indicates whether it is available for immediate use as a server. When *Server* becomes *true*, processor 3 switches to server mode; when *Other* becomes *true*, processor 3 finishes servicing requests it has already received then switches to other tasks. Note that $\{1\}$ is a fixed set for each of these predicates, so $\mathrm{PDDA}_{\mathrm{GW}}$ detects them in $O(S^2)$ time, while the worst-case time complexity of a detection algorithm that constructs $CGS(c)$ is $\Omega(S^3)$, since $N = 3$.

Debugging Partitioned Databases. Consider a system with three processors that manage a database. Processor 1 stores an index of the entire database; the database contents are partitioned between processors 2 and 3. Each processor i stores a cutoff value in a local variable α_i. Processor 2 is responsible for records with keys less than or equal to α_2; processor 3 is responsible for records with keys greater than α_3. Processor 1 uses α_1 to decide where to forward updates.

Processor $i \in \{2, 3\}$ may change the cutoff by setting local variable $changing_i$ to *true*, sending appropriate messages to the other processors, then setting $changing_i$ to *false* when the operation is completed. The system is expected to satisfy the invariant

$$(\neg changing_2 \wedge \neg changing_3) \Rightarrow (\alpha_1 = \alpha_2 \wedge \alpha_2 = \alpha_3). \tag{7}$$

$\mathrm{PDDA}_{\mathrm{GW}}$ can be used to detect and report violations of this invariant. We want to detect the negation of (7). This can be done in $O(S^2)$ time by putting the negation of (7) in DNF and using $\mathrm{PDDA}_{\mathrm{GW}}$ to detect each disjunct separately. For comparison, the worst-case time complexity for a detection algorithm that constructs $CGS(c)$ is $\Omega(S^3)$, since $N = 3$.

Sorting Arrays. Consider a system of N processors that maintains an array of size $B \cdot N$ in sorted order. The array is distributed in contiguous blocks of size B, with the i^{th} block A_i allocated to processor i. The array is sorted if $(\bigwedge_{i=1}^{N} Sorted_i) \wedge (\bigwedge_{i=1}^{N-1} A_i[B] \leq A_{i+1}[1])$ holds, where $Sorted_i \triangleq (\forall j \in [1..(B-1)] : A_i[j] \leq A_i[j+1])$. Periodically, the values of some elements in the array change, and the system re-sorts the array. States in which the array is sorted can be detected using $\mathrm{PDDA}_{\mathrm{GW}}$ with fixed set $\{1, 3, 5, \ldots, N-1\}$ in $O(S^{N/2+1})$ time, where for convenience we assume N is even. For comparison, the worst-case time complexity of a detection algorithm that constructs $CGS(c)$ is $\Omega(S^N)$. Note that the exponent of S in the complexity of $\mathrm{PDDA}_{\mathrm{GW}}$ is smaller by an amount directly proportional to N—not just by a constant.

5 Faster Off-line Detection using Matrix Multiplication

In this section, we describe how fast matrix-multiplication algorithms allow faster off-line detection of $\mathbf{Poss}\,\Phi$ for certain predicates Φ. For convenience, we describe the technique as it applies to predicates of the form

$$\mathcal{M} \triangleq \phi_{12}(x_1, x_2) \wedge \phi_{23}(x_2, x_3) \wedge \phi_{13}(x_1, x_3), \tag{8}$$

and then discuss other classes of formulas to which it applies. The basic idea is to represent the values of each predicate ϕ_{ij} in computation c as an $S \times S$ Boolean matrix ϕ'_{ij}. We also encode the happened-before relation in these matrices:

$$\phi'_{ij}(\alpha, \beta) \triangleq \phi_{ij}(c_i[\alpha], c_j[\beta]) \wedge c_i[\alpha] \parallel c_j[\beta]. \tag{9}$$

From the definition of **Poss**, we see that $c \models \textbf{Poss}\,\mathcal{M}$ iff Ψ, where

$$\Psi \triangleq (\exists \alpha_1, \alpha_2, \alpha_3 \in [1..S] : \phi'_{12}(\alpha_1, \alpha_2) \wedge \phi'_{23}(\alpha_2, \alpha_3) \wedge \phi'_{13}(\alpha_1, \alpha_3)).$$

Let $\psi_{13}(\alpha_1, \alpha_3) \triangleq (\exists \alpha_2 \in [1..S] : \phi'_{12}(\alpha_1, \alpha_2) \wedge \phi'_{23}(\alpha_2, \alpha_3))$. Then

$$\Psi = (\exists \alpha_1, \alpha_3 \in [1..S] : \psi_{13}(\alpha_1, \alpha_3) \wedge \phi'_{13}(\alpha_1, \alpha_3)).$$

Using Strassen's matrix-multiplication algorithm, the matrix representing ψ_{13} can be computed with time complexity $\Theta(S^{\log_2 7})$ [Str69, AHU74].[5] By the naive algorithm, the truth of Ψ can then be determined in $\Theta(S^2)$ time. Thus, this algorithm detects $\textbf{Poss}\,\mathcal{M}$ with worst-case time complexity $\Theta(S^{\log_2 7} + S^2)$, or approximately $\Theta(S^{2.81})$. The worst-case time complexity of PDDA_{GW} on such predicates is $\Theta(S^3)$, so PDDA_{GW} is not optimal on this class of predicates.

This matrix-multiplication translation is not limited to predicates of the form (8). For example, it can be used with any predicate containing conjuncts $\phi_{ij}(x_i, x_j)$ and $\phi_{jk}(x_j, x_k)$, provided no single conjunct contains x_i, x_j and x_k.

Fast On-line Matrix Multiplication Considered Unlikely. This matrix-multiplication technique for off-line detection of $\textbf{Poss}\,\mathcal{M}$ does not extend to an on-line algorithm. To understand why, recall that Strassen's matrix-multiplication algorithm computes $C = A \cdot B$ by re-writing it in partitioned form as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

computing seven intermediate matrix products involving the submatrices of A and B, and expressing the submatrices of C as linear combinations of these intermediate results. Strassen's expressions for the submatrices of C involve cancellations, so these expressions cause spurious dependencies. For example, expanding the expression for C_{11} yields $C_{11} = \cdots + A_{22}B_{22} + \cdots - A_{22}B_{22} + \cdots$. Thus, A_{22} and B_{22} must be known in order to compute C_{11} using Strassen's method. By definition, $C_{11} = A_{11}B_{11} + A_{12}B_{21}$, so C_{11} does not actually depend on A_{22} or B_{22}. Such spurious dependencies can cause delays in detection, thereby violating the specification of on-line detection.

6 Complexity of Poss

Detection algorithms are proliferating, but little has been proved about their optimality. To obtain useful results, the complexity of the problem must be painted with a sufficiently fine brush. If one considers only the problem of detecting $\textbf{Poss}\,\Phi$ for arbitrary predicates Φ, then the worst-case time complexity is $\Omega(S^N)$, since a detection algorithm can do no better than to evaluate an arbitrary N-ary primitive relation on every possible consistent global state. This

[5] Any matrix-multiplication algorithm can be used. We phrase our remarks in terms of Strassen's algorithm, even though asymptotically faster algorithms exist [CW87], because Strassen's algorithm is relatively simple and well-known.

analysis does not distinguish algorithms that are asymptotically faster on certain predicates. For example, to characterize the advantage of $\mathrm{PDDA}_{\mathrm{GW}}$ over Cooper and Marzullo's algorithm, one must consider the worst-case complexity of both algorithms on various classes of predicates. At best, one might find a detection algorithm that is optimal for every class of predicates. As shown in Section 5, $\mathrm{PDDA}_{\mathrm{GW}}$ is not optimal for predicates like (8), so the off-line version of $\mathrm{PDDA}_{\mathrm{GW}}$ is not optimal in the strongest sense.

To check optimality of any algorithm for detecting $\mathbf{Poss}\,\Phi$, we must determine the intrinsic complexity of the problem. Chase and Garg took a step in this direction by proving that detecting $\mathbf{Poss}\,\Phi$ is NP-complete even when restricted to communication-free computations with $S \leq 2$ [CG94]. We advocate characterizing the complexity of detecting $\mathbf{Poss}\,\Phi$ for particular classes of formulas Φ. Since conjunctions of 1-local predicates can be detected in polynomial time [GW94], it is natural to ask about the complexity of detecting conjunctions of 2-local predicates. We show that this problem is NP-complete by giving a reduction from the *k-partite clique problem*, which is defined as follows.

Input: A k-partite undirected graph G, *i.e.*, disjoint sets $V_1, \ldots, V_k$ of nodes and an edge relation E such that $(\forall \{v,w\} \in E : (\forall i \in [1..k] : \{v,w\} \not\subseteq V_i))$.
Output: Does G have a k-clique?

Lemma 7. The k-partite clique problem is NP-complete, even with the restriction that $|V_i| \leq 3$ for $i \in [1..k]$.

Proof. This problem is a special case of the clique problem, which is in NP, so this problem is also in NP. The reduction from satisfiability to the clique problem given by Aho, Hopcroft, and Ullman [AHU74, pp. 384-386] has the property that it maps all instances of 3-satisfiability into instances of the clique problem in which the graph is k-partite with $|V_i| \leq 3$. Thus, their reduction shows that the k-partite clique problem with $|V_i| \leq 3$ is NP-hard. □

Theorem 8. Detecting $\mathbf{Poss}\,\Phi$ is NP-complete, even when restricted to communication-free computations with $S \leq 3$ and to predicates Φ that are conjunctions of 2-local predicates.

Proof. We give linear-time reductions in both directions between this restricted detection problem and the k-partite clique problem. Both transformations satisfy $|V_i| = |c_i|$ and $k = N$. The desired result follows immediately from Lemma 7.

Given an instance of $c \models \mathbf{Poss}\,\Phi$, we define an N-partite graph as follows. Since Φ is a conjunction of 2-local predicates, it can be written in the form

$$\bigwedge_{1 \leq i < j \leq N} \phi_{ij}(g(i), g(j)). \tag{10}$$

Let $V_i \triangleq \{i\} \times \mathcal{U}(c_i)$, and let $E \triangleq \{\{\langle i, \alpha\rangle, \langle j, \beta\rangle\} \mid \phi_{ij}(c_i[\alpha], c_j[\beta]) \wedge c_i[\alpha] \parallel c_j[\beta]\}$. It is easy to show that the N-partite graph $(\bigcup_{i=1}^{N} V_i, E)$ has an N-clique iff $c \models \mathbf{Poss}\,\Phi$.

Given an instance of the k-partite clique problem, let $N = k$, let each c_i be some total ordering of V_i, let $\rightarrow$ be the union of those total orderings, and for $1 \leq i < j \leq N$ let $\phi_{ij}(c_i[\alpha], c_j[\beta]) \triangleq (\{c_i[\alpha], c_j[\beta]\} \in E)$. It is easy to show that $c \models \mathbf{Poss}\,\Phi$ iff the given k-partite graph has a k-clique. $\square$

Theorem 8 characterizes the dependence of the complexity of detecting $\mathbf{Poss}\,\Phi$ on N but says nothing about the dependence on S. Since detection problems typically have $S \gg N$, the dependence on S is crucial. Unfortunately, proving lower bounds on the complexity in terms of S appears difficult. For example, one might conjecture that the worst-case complexity of detecting conjunctions of 2-local predicates (*i.e.*, formulas of form (10)) is $\Omega(S^N)$. This conjecture places an exponential lower bound on an NP-complete problem, so proving it is as hard as proving $P \neq NP$.

Complexity of On-line Detection. Optimality of the on-line version of $\mathrm{PDDA_{GW}}$ for almost all classes of predicates is an open question. The matrix-multiplication technique of Section 5 does not extend to the on-line case, so we do not know of any class of predicates on which the on-line version of $\mathrm{PDDA_{GW}}$ is not optimal. Showing optimality requires proving a lower bound, which appears to be difficult in the on-line case as well.[6]

Acknowledgments. We thank Dexter Kozen, Monika Rauch Henzinger, and Moshe Vardi for their comments on lower bounds.

References

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms.* Addison Wesley, 1974.

[BDG88] José Luis Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity I.* Springer-Verlag, 1988.

[BM93] Özalp Babaoğlu and Keith Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In Sape Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison Wesley, 2nd ed., 1993.

[CG94] Craig M. Chase and Vijay K. Garg. On techniques and their limitations for the global predicate detection problem in distributed systems. Technical Report ECE-PDS-1994-04, Parallel and Distributed Systems Laboratory, University of Texas at Austin, 1994.

[CM91] Robert Cooper and Keith Marzullo. Consistent detection of global predicates. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, 1991. Appeared as ACM SIGPLAN Notices 26(12):167-174, December 1991.

[6] There are non-linear lower bounds for dynamic (*i.e.*, on-line) graph connectivity problems [FH94]. However, the proofs of these lower bounds depend crucially on the fact that edges can be both added and deleted, while in the on-line version of our clique problem, edges are added but never deleted.

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Conference Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 1–6, 1987.

[DJR93] Claire Diehl, Claude Jard, and Jean-Xavier Rampon. Reachability analysis on distributed executions. In J.-P. Jouannaud and M.-C. Gaudel, editors, *TAPSOFT '93: Theory and Practice of Software Development*, volume 668 of *Lecture Notes in Computer Science*, pages 629–643. Springer-Verlag, 1993.

[FH94] Michael L. Fredman and Monika Rauch Henzinger. Lower bounds for dynamic connectivity problems in graphs. Technical Report TR 94-1420, Cornell University, April 1994. Also appeared in extended abstract: Monika Rauch. Improved Data Structures for Fully Dynamic Biconnectivity. In *Proc. 26th Annual Symposium on Theory of Computing (STOC '94)*, pages 686-695, 1994.

[Fid88] C. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *Proceedings of the 11th Australian Computer Science Conference*, pages 56–66, 1988.

[FR94] Eddy Fromentin and Michel Raynal. Inevitable global states: a concept to detect properties of distributed computations. Internal Publication PI-842, IRISA, June 1994.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[GW92] Vijay K. Garg and Brian Waldecker. Detection of unstable predicates in distributed programs. In *Proceedings of the 12th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*, pages 253–264. Springer-Verlag, 1992.

[GW94] Vijay K. Garg and Brian Waldecker. Detection of weak unstable predicates in distributed programs. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):299–307, 1994.

[JMN95] R. Jegou, R. Medina, and L. Nourine. Linear space algorithm for on-line detection of global predicates. To appear in *Proc. International Workshop on Structures in Concurrency Theory (STRICT '95)*, 1995.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–564, 1978.

[Mat89] Friedemann Mattern. Virtual time and global states of distributed systems. In M. Corsnard, editor, *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 120–131. North–Holland, 1989.

[MN91] Keith Marzullo and Gil Neiger. Detection of global state predicates. In *Proceedings of the 5th International Workshop on Distributed Algorithms*, volume 579 of *Lecture Notes in Computer Science*, pages 254–272. Springer-Verlag, 1991.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[TG94] Alexander I. Tomlinson and Vijay K. Garg. Monitoring functions on global states of distributed programs. Technical Report TR-PDS-1994-006, Parallel and Distributed Systems Laboratory, University of Texas at Austin, 1994.

This article was processed using the LaTeX macro package with LLNCS style