Addressing False Causality while Detecting Predicates in Distributed Programs *

Ashis Tarafdar
Dept. of Computer Sciences
University of Texas at Austin
Austin, TX 78712
(ashis@cs.utexas.edu)

Vijay K. Garg
Dept. of Electrical and Computer Engg.
University of Texas at Austin
Austin, TX 78712
(garg@ece.utexas.edu)

Abstract

The partial-order model of distributed computations based on the happened before relation has been criticized for allowing false causality between events. Our strong causality model addresses this problem by allowing multiple local threads of control.

This paper addresses the predicate detection problem for the class of weak conjunctive predicates in the strong causality model. We show that, in general, the problem is NP-complete. However, an efficient solution is demonstrated for a useful sub-case. Further, this solution can be used to achieve an exponential reduction in time for solving the general problem.

Our predicate detection algorithms can be applied to distributed debugging when processes have independent events, as in multi-threaded processes.

1 Introduction

A fundamental problem in distributed systems is that of *predicate detection* [1, 6] – detecting whether a global condition occurs while running a distributed program. Its main application is in the testing, debugging and monitoring of distributed programs. Predicate detection is usually specified in a model of distributed computation based on a *happened-before* relation [8], which models the independence of concurrent events on different processes. However, it has been criticized for allowing *false causality* between events [3, 11]. This paper addresses this issue by extending the model and shows how to solve predicate detection in the extended model.

Consider running a distributed mutual-exclusion program. The happened-before model of the resulting distributed computation is shown in Figure 1(i). If mutual-exclusion violation is the predicate that we are trying to detect, it would not be detected because the

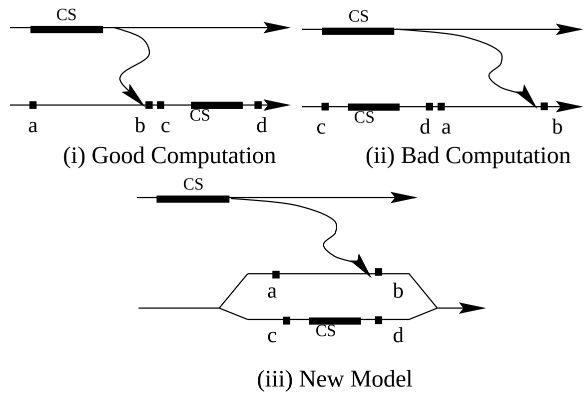

Figure 1: Example: Addressing False Causality

message ensures that the two critical sections cannot occur at the same time. However, in reality, the message may have been fortuitous and the sections of execution marked by intervals (a, b) and (c, d) may have been independent (for example, independent threads). The scenario in Figure 1(i) may be just one possible scheduling of events. Figure 1(ii) shows another scheduling in which mutual-exclusion would be violated.

A model that partially orders the events on a local process would allow events within a process to be independent. Figure 1(iii) shows such a model for the example. This representation models both of the previous schedulings. In general, there would be an exponential number of happened-before representations corresponding to a single representation in the new model. We call the new model a *strong causality diagram*.

Our contributions are two-fold. First, we define and motivate a new model of computation, strong causality diagrams, that extends the happened-before model to address false causality. Next, we present results in solving predicate detection in the strong

*supported in part by the NSF CCR-9414780, CCR-9520540, a General Motors Fellowship, Texas Education Board ARP-320 and an IBM grant

causality model. We focus on the important class of *weak conjunctive predicates* which express a large number of important global properties, and which can be solved efficiently in the happened-before model [7]. We demonstrate that for general strong causality diagrams, the problem is NP-complete. However, for certain restricted, but useful, classes of strong causality diagrams, the problem may be solved efficiently. These restricted classes correspond to having either the receive events or the send events totally ordered, while allowing all other events to be partially ordered. Further, we can decompose a general strong causality diagram into strong causality diagrams of these classes, to achieve an exponential reduction in time as compared to the naive solution. Lastly, we discuss how the strong causality diagram model may be applied in practical modeling of distributed programs with multi-threaded processes. We also discuss the applications of our results in predicate detection to testing and debugging such programs.

It is definitely harder to solve predicate detection in the strong causality model than in the happened-before model. However, even being able to efficiently solve predicate detection for a restricted class is a great improvement over the alternative of solving predicate detection on an exponential number of corresponding happened-before representations. In the worst case where the strong causality diagram does not fall into either totally ordered sends or totally ordered receives, we may decompose the problem into strong causality diagrams each of which belongs to one of these classes and so efficiently solve for each such diagram. Even though this solution is exponential, it achieves an exponential reduction as compared to the naive approach of considering all possible happened-before representations.

In Section 2, we discuss related work. In Section 3, we formally define the strong causality diagram model and define the predicate detection problem that we will be addressing. In Section 4, we present our results in solving predicate detection in strong causality diagrams. Section 5 discusses how to apply the strong causality model and the results in predicate detection in this model.

2 Related Work

Using partial orders to model concurrency avoids the combinatorial explosion involved in the interleaving model. This fact has led to a number of studies of such partial order models [8, 10]. Our work builds on the happened-before model [8] by extending the same idea to allow concurrency or independence between events within a local process. We are aware of no other study of such an extended happened-before model.

Predicate detection is a widely-studied problem [1, 6]. Approaches to solving predicate detection are divided into three categories: global snapshot based [2], lattice construction based [4], and predicate restriction based [7] approaches. The first approach can detect only *stable* predicates (which remain true once they become true), and the second approach uses the interleaving model of concurrency and, therefore, suffers from the above-mentioned combinatorial explosion. We follow the last approach that uses the partial order model and limits itself to classes of predicates which can be detected efficiently. We focus on the important class of weak conjunctive predicates for which an optimal solution was provided for the happened-before model of computation [7]. Our work extends this solution to the proposed strong causality model of computation.

3 Model and Problem Definition

The usual model of a distributed computation is based on the *happened-before* relation. The motivation for this model is that a partial ordering of events in a distributed computation provides a more feasible, concise and meaningful model than a total ordering of events. We call this model a *happened-before diagram* and represent it by $(S_1, S_2, \ldots S_n, <, \rightsquigarrow)$. Here $S_1, S_2, \ldots S_n$ are sets of states on each of n processes, $<$ is the *locally precedes* relation that totally orders the states on each process in their order of occurrence in time, and $\rightsquigarrow$ is the *remotely precedes* relation that relates the state that sends a message to the state that receives it. We will use letters $s, t, u, \ldots$ to denote states.

In this model, the notion of whether a global state can "occur" is defined as follows. Define the *causally precedes* (or happened before) relation, $\rightarrow$, as the transitive closure of $<$ and $\rightsquigarrow$ (i.e. $\rightarrow = (< \cup \rightsquigarrow)^+$). If $S = S_1 \cup S_2 \cup \ldots S_n$ then $(S, \rightarrow)$ forms an irreflexive poset. If two states s and t are incomparable in this poset (i.e., $s \not\rightarrow t$ and $t \not\rightarrow s$) then s and t are called *concurrent*, denoted by $s \| t$. A *global state* is defined as a set of n states, one from each S_i. We use letters G, H to denote global states. For a global state G, $G[i]$ denotes the state in G from S_i. However, all global states cannot occur in a computation. Global states which can occur are modeled as *consistent global states* – global states in which every pair of states is concurrent.

As has been discussed, the happened-before diagrams are limited because they totally order the events local to a process, thus, not modeling possibly independent local events. Our model, therefore, extends the locally precedes relation, $<$, from a total order to an irreflexive partial order, $<_s$, called the *strong locally precedes* relation. Our model of a distributed computation now becomes a *strong causality diagram*, $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$. We use the letters D, E to represent strong causality diagrams. Note that a happened-before diagram is a special case of a strong causality diagram in which $<_s$ is a total order $<$. All notions of global states and their consistency are defined in an analogous manner for strong causality diagrams. We use the notation $\rightarrow_s$ and $\|_s$ to stand for the *strong causally precedes* and *strong concurrent* relations respectively.

A *linearization* of a partial order is a total order that contains the partial order. We define a *local linearization* of a strong causality diagram to be the happened-before diagram that may be obtained by linearizing the $<_s$ relation within each of the sets S_i, to give a $<$ relation. We represent the set of all possible local linearizations of a strong causality diagram D by $Lin(D)$. Note that, in general, a single strong causality diagram would correspond to an exponential number of happened-before diagrams. In fact, this conciseness is part of the reason for working in the extended model.

We will be interested in whether a consistent global state satisfies certain global conditions. We model these as *global predicates*, boolean-valued functions defined on the set of all global states. Similarly, *local predicates* are boolean-valued functions defined on the set of states in a process. We will use c to represent a global predicate and x_i to represent local predicates defined on S_i. Our focus will be on an important class of global predicates called *conjunctive predicates*. These are predicates of the form $c = c_1 \wedge c_2 \wedge \ldots c_n$ with the usual semantics (i.e., $c(G) = c_1(G[1]) \wedge c_2(G[2]) \wedge \ldots c_n(G[n])$).

We now define our general problem in this model as:

Conjunctive Predicate Detection in General Strong Causality Diagrams (CPG):
Given a conjunctive predicate c and a strong causality diagram $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$, does there exist a consistent global state G such that $c(G)$ holds.

4 Solving Conjunctive Predicate Detection in Strong Causality Diagrams

The problem of weak conjunctive predicate detection was efficiently (in $O(mn^2)$ time, where m is a bound on the number of states in a process) and optimally solved for happened-before diagrams in [7].

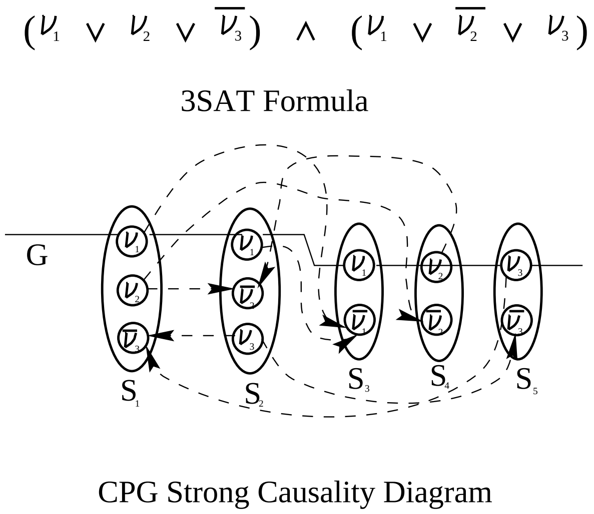

Figure 2: Example Transformation

However, in strong causality diagrams, the problem becomes expectedly harder. In fact, we have:

Theorem 1: *CPG is NP-complete.*

Proof: CPG is in NP because, given a global state G, $c(G)$ can be checked in polynomial time (assuming each c_i can be checked in polynomial time) and every pair of states in G can be checked for strong concurrency in polynomial time (how this is done will be clear in Section 5).

To show that CPG is NP-hard, we transform 3SAT to CPG. Let 3SAT be specified by l variables $\nu_1, \nu_2, \ldots \nu_l$, and k clauses $\rho_1, \rho_2, \ldots \rho_k$. Let $\rho_i[j]$, $(1 \le j \le 3)$ be the 3 literals in clause ρ_i. Let ν stand for any variable.

Our transformation (refer to the example in Figure 2 consists of defining $k + l$ processes represented by $k + l$ sets of states $S_1, S_2, \ldots S_{k+l}$. Each S_i for $1 \le i \le k$ corresponds to ρ_i, so that S_i contains three states $s_i[j]$, $(1 \le j \le 3)$, where $s_i[j]$ corresponds to $\rho_i[j]$. Each S_{k+i} for $1 \le i \le l$ contains two states $s_{k+i}[j]$, $(1 \le j \le 2)$, where $s_{k+i}[1]$ corresponds to the literal ν_i and $s_{k+1}[2]$ corresponds to the literal $\overline{\nu}_i$. Let $<_s = \emptyset$. Define $\rightsquigarrow$ as follows. $(s_i[y], s_j[z]) \in \rightsquigarrow, (i \neq j)$ iff $\rho_i[y] = \nu$ and $\rho_j[z] = \overline{\nu}$ for some variable ν. Further, we define $c = true$ since we are only interested in determining if there is a consistent global state.

First, we show that if CPG answers "yes" then 3SAT answers "yes". If CPG detects a consistent global state G, then for each variable ν_i, exactly one of the literals ν_i and $\overline{\nu}_i$ corresponds to a state in G. This is because $G[k + i]$ must either correspond to ν_i or $\overline{\nu}_i$, using the definition of S_{k+i}, and no two states in G may correspond to ν_j and $\overline{\nu}_j$ for any j (using the definition of $\rightsquigarrow$). So, to every variable ν_i, we assign *true* if $G[k+i]$ corresponds to literal ν_i and *false* if it corresponds to $\overline{\nu}_i$. This truth assignment satisfies the literal corresponding to $G[j]$, $(1 \leq j \leq k)$ and, therefore, satisfies each of the clauses.

Next, we show that if 3SAT answers "yes" then CPG answers "yes". Given a satisfying truth assignment to the variables , we construct a global state G as follows. Choose the state $G[i], (1 \leq i \leq k+l)$ so that the corresponding literal is satisfied by the truth assignment. This is possible for $G[i], (1 \leq i \leq k)$ because the truth assignment solves 3SAT and, therefore, satisfies all clauses. It is possible for $G[k+i], (1 \leq i \leq l)$ because either ν_i or $\overline{\nu}_i$ must be satisfied by the truth assignment. Every pair of states in G must be strong concurrent because of the definition of $\rightsquigarrow$ and $<_s$. G satisfies conjunctive predicate c because of the definition of c. $\Box$

The motivation for strong causality diagrams is that it concisely represents many happened-before diagrams. The following result confirms that solving predicate detection in a strong causality diagram is equivalent to solving predicate detection for all of the diagram's local linearizations. First, we define the new problem:

Conjunctive Predicate Detection in Local Linearizations of General Strong Causality Diagrams (CPL):
Given a conjunctive predicate c and a strong causality diagram $D = (S_1, S_2, \dots S_n, <_s, \rightsquigarrow)$, does there exist a consistent global state G in any $E \in Lin(D)$ such that $c(G)$ holds.

Theorem 2: *CPG is equivalent to CPL.*

Proof: Let $D = (S_1, S_2, \dots S_n, <_s, \rightsquigarrow)$ be a strong causality diagram and let c be a conjunctive predicate defined on it.

First, we prove that, if a global state G is a solution to CPL, then it is also a solution to CPG. So, let $E = (S_1, S_2, \dots S_n, <, \rightsquigarrow)$ be a local linearization of D, and let a global state G be consistent in E, and let $c(G)$ hold. We have to prove that G is also consistent in D. Since G is consistent in E, for any two states $G[i]$ and $G[j]$, $G[i] \not\rightarrow G[j]$. Since $\rightarrow_s \subseteq \rightarrow$, we have $G[i] \not\rightarrow_s G[j]$. So G is consistent in D.

Next, we prove that, if a global state G is a solution to CPG, then it is also a solution to CPL. So, let a global state G be consistent in D, and let $c(G)$. We construct a local linearization $E = (S_1, S_2, \dots S_n, <, \rightsquigarrow)$ of G as follows. If each S_i is totally ordered in D, then we are already done since we can choose $E = D$. So assume that two states x and y of the same process are incomparable in D. From D, we will construct another strong causality diagram $D' = (S_1, S_2, \dots, S_n, <_s', \rightsquigarrow)$ such that:
(1) $<_s \subseteq <_s'$,
(2) G is consistent in D', and
(3) x and y are comparable in D'.
By repeating this procedure, we will eventually reach a linearization of D. We make x and y comparable in $<_s'$ as follows.
Case 1: $x \rightarrow_s G[k]$ for some k.
We add (x, y) to $<_s$. We show that G is consistent in D'. If not, there exist i and j such that $G[i] \rightarrow_s x$ and $y \rightarrow_s G[j]$. However, this implies that $G[i] \rightarrow_s G[k]$ which is false.
Case 2: $x \not\rightarrow_s G[k]$ for all k.
We add (y, x) to $<_s$. We show that G is consistent in D'. If not, there exist i and j such that $G[i] \rightarrow_s y$ and $x \rightarrow_s G[j]$. However, this violates the condition that $x \not\rightarrow_s G[k]$ for all k. $\Box$

This result tells us that if we exhaustively detect a predicate in each of $Lin(D)$ then we have also done so for D. Since this would be very inefficient, we identify two classes of strong dependency diagrams for which we may apply a special predicate detection algorithm to a specially chosen representative from $Lin(D)$ in order to efficiently detect a predicate.

Consider a strong causality diagram $D = (S_1, S_2, \dots S_n, <_s, \rightsquigarrow)$. If $s \rightsquigarrow t$, then we call s a *send state* and we call t a *receive state*. Let *Snd* be the set of send states in S (S will stand for $S_1 \cup S_2 \cup \dots S_n$ throughout this paper.) and let *Rcv* be the set of receive states in S. We use Snd_i and Rcv_i to denote the set of send and receive states, respectively, in S_i. We say that D is *receive-ordered* if, for each i, the receive states in Rcv_i are totally ordered by $\rightarrow_s$ (i.e., $\forall i: \forall s, t \in Rcv_i: (s \rightarrow_s t) \vee (t \rightarrow_s s)$). We say that D is *send-ordered* if, for each i, the send states in Snd_i are totally ordered by $\rightarrow_s$.

Let CPR and CPS be the CPG problem specialized to receive-ordered and send-ordered strong causality diagrams, respectively.

Let $D = (S_1, S_2, \dots S_n, <_s, \rightsquigarrow)$ be a receive-ordered strong causality diagram. Let c be a conjunctive predicate defined on D. We now pick a special representative E from $Lin(D)$ so that it satisfies the following property:

$$\mathbf{P1}: \forall i: \forall s \in S_i: \forall t \in Rcv_i: (s \parallel_s t) \Rightarrow (s \leq t)$$

This ensures that we linearize the partial order $<_s$ on each process such that a receive state is ordered after all the states that are concurrent with it. The property is well-defined because no two receive states are concurrent.

Input:
- S_i set of states in process i
- $\prec_i$ transitive reduction of $<_s$ restricted to S_i
- Rcv_i set of receive states in process i ($Rcv \cap S_i$)

Output:
- $\mathcal{Q}_i$ queue of states in S_i, initially $\emptyset$, and finally contains all states in $<$ total order

Predicates:
- $select(Z)$ any element from non-empty set Z

Variables:
- $M = \emptyset$ set of states in S_i
- $R = \emptyset$ set of states in S_i
- s, t states in S_i
- $k[S_i]$ array of integers for each state in S_i

L1 **for** each state s in S_i **do**
L2 $\quad k[s]$:= no. of incoming edges in $\prec_i$ for s
L3 $\quad$ **if** $(k[s] = 0)$ **then**
L4 $\quad\quad$ **if** $(s \in Rcv_i)$ **then** $R := R \cup \{s\}$
L5 $\quad\quad$ **else** $M := M \cup \{s\}$
L6 **while** $(S_i \neq \emptyset)$ **do**
L7 $\quad$ **if** $(M \neq \emptyset)$ **then** $t := select(M)$
L8 $\quad$ **else** $t := select(R)$
L9 $\quad enqueue(\mathcal{Q}_i, t)$
L10 $\quad S_i := S_i - \{t\}$
L11 $\quad$ **for** each state s such that $t \prec_i s$ **do**
L12 $\quad\quad k[s] := k[s] - 1$
L13 $\quad\quad$ **if** $(k[s] = 0)$ **then**
L14 $\quad\quad\quad$ **if** $(s \in Rcv_i)$ **then** $R := R \cup \{s\}$
L15 $\quad\quad\quad$ **else** $M := M \cup \{s\}$

Figure 3: Algorithm RECEIVE-SORT

In order to ensure this property, we apply a special linearization algorithm, RECEIVE-SORT shown in Figure 3, for each process. The algorithm is a modification of a standard topological sort algorithm that gives a higher priority to non-receive states so that all states concurrent to a receive state precede it in the total ordering. The algorithm takes as input the partial order of $<_s$ restricted to a process and specified as its transitive reduction (or Hasse Diagram). It is easy to show that this correctly produces a linearization of the partial order for each process. It is also easy to show that the linearizations produced by the algorithm are those that ensure Property P1:

Theorem 3: *If $D = (S_1, S_2, \ldots S_n, <_s, \leadsto)$ is a receive-ordered strong causality diagram, and if $E = (S_1, S_2, \ldots S_n, <, \leadsto)$ is the local linearization of D produced by applying Algorithm* RECEIVE-SORT *for each process, then E satisfies Property* P1.

Proof: The following four invariants are maintained by the algorithm:
($INV1$) $\forall s \in S_i :\ k[s]$ = number of states immediately smaller than s in S_i ($|\{t \in S_i :\ t \prec_i s\}|$)
(Here S_i is the variable used in the algorithm and not the static S_i representing all states in the ith process.)
($INV2$) M is the set of minimal elements in S_i that are not in Rcv_i
($INV3$) R is the set of minimal elements in S_i that are in Rcv_i
($INV4$) S_i contains no element transitively smaller than (w.r.t ${\prec_i}^+$) a minimal element
It is easy to prove $INV1$ from the algorithm and then prove $INV2$ and $INV3$ using it. $INV4$ follows from $INV1$ by induction on the length of the transitive chain.

Let $s \in S_i$ and $t \in Rcv_i$ be two states such that $s \parallel_s t$. Since D is receive-ordered, either $s \notin Rcv_i$ or $s = t$. If $s = t$, we are done. So we consider the case of $s \notin Rcv_i$. Focus on the iteration in which t is enqueued in $\mathcal{Q}_i$. Since $t \in Rcv_i$, t must have been chosen in line L8 (using INV2, INV3). So $M = \emptyset$ in this iteration. Further, $R = \{t\}$ (using INV3, INV4 and that D is receive-ordered). We conclude that t is the *mimimum* element of S_i in this iteration. Therefore, $s \notin S_i$ in this iteration. Therefore, it must have been enqueued before this iteration. So $s<t$. $\Box$

From Property P1, we derive the following useful property:

P2 : $\forall i, j :\ (i \neq j) :\ \forall s \in S_i,\ t, u \in S_j :$
$$(s \rightarrow_s t) \wedge (t \leq u) \ \Rightarrow\ (s \rightarrow_s u)$$

Lemma 1: *If $D = (S_1, S_2, \ldots S_n, <_s, \leadsto)$ is a receive-ordered strong causality diagram and $E \in Lin(D)$ such that $E = (S_1, S_2, \ldots S_n, <, \leadsto)$, then: E satisfies Property* P1 $\Rightarrow$ *E satisfies Property* P2

Proof: Let $i \neq j$ and $s \in S_i$ and $t, u \in S_j$ and let $s \rightarrow_s t$ and $(t \leq u)$. Since $s \rightarrow_s t$ and s and t are on different processes, there must be some receive state v on S_j, such that $s \rightarrow_s v$ and $v \leq_s t$. Since $<$ is a linearization of $<_s$, we have $v \leq u$. If $v = u$, we are done. So assume $v < u$. Using the contra-positive of property P1, we have $u \not\parallel_s v$. We cannot have $u <_s v$ because $v \leq u$ and $<$ is a linearization of $<_s$. So we must have $v <_s u$. Since $s \rightarrow_s v$, we have $s \rightarrow_s u$ by transitivity. $\Box$

We now apply algorithm PRED-DETECT in Figure 4 to the special representative chosen from $Lin(D)$ using algorithm RECEIVE-SORT.

Input:
  $\mathcal{Q}_1, \mathcal{Q}_2, \ldots \mathcal{Q}_n$ process state queues in $<$ order
  $\rightarrow_s$ strong causally precedes relation
  $x_1, x_2, \ldots x_n$ local predicates for each process
Output:
  detected boolean
Constants:
  $all \;=\; \{1, 2, \ldots, n\}$
Variables:
  $low, newlow$ subsets of *all*
  k, l integers in *all*
L1 $low \;:=\; all$
L2 **while** $(low \;\neq\; \emptyset)$ **do**
L3 $newlow \;:=\; \emptyset$
L4   **for** k in low **do**
L5   **if** $(\;\neg x_k(head(\mathcal{Q}_k))\;)$ **then**
L6 $newlow \;:=\; newlow \;\cup\; \{k\}$
L7   **else**
L8   **for** l in all **do**
L9   **if** $(head(\mathcal{Q}_k) \rightarrow_s head(\mathcal{Q}_l))$ **then**
L10 $newlow \;:=\; newlow \;\cup\; \{k\}$
L11   **if** $(head(\mathcal{Q}_l) \rightarrow_s head(\mathcal{Q}_k))$ **then**
L12 $newlow \;:=\; newlow \;\cup\; \{l\}$
L13 $low \;:=\; newlow$
L14   **for** k in low **do**
L15 $deletehead(\mathcal{Q}_k)$
L16 $detected \;:=\; (\forall k : \; \neg empty(\mathcal{Q}_k))$

Figure 4: Algorithm PRED-DETECT

Our final result shows that applying PRED-DETECT to the representative in $Lin(D)$ is sufficient for detecting the predicate in D. The main idea of the algorithm is similar to that used in [7] to optimally solve the problem for happened-before diagrams. We start with the lowest global state and move upwards. If, in a global state G, we find a state $G[i]$ that strong causally precedes another state $G[j]$ then using Property P2, we are guaranteed that $G[i]$ also strong causally precedes every state higher than $G[j]$ in the total order $<$. So $G[i]$ can be safely discarded. If no such pair of states can be found, then the global state is consistent. The algorithm discards at least one state in each iteration and so must terminate.

Theorem 4: *If $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$ is a receive-ordered strong causality diagram, c is a conjunctive predicate, and $E = (S_1, S_2, \ldots S_n, <, \rightsquigarrow)$ is a local linearization of D satisfying property* P1*, then applying algorithm PRED-DETECT to E and c solves CPR for D and c.*

Proof: Lemma 1 implies that E satisfies P2. We first prove that if PRED-DETECT returns $detected = true$ then there is indeed a strong consistent global state in D such that c holds in it. It is easy to verify the following loop invariant:

$(INV1)\;\; \forall k, l \notin low : \; \neg empty(\mathcal{Q}_k) \wedge \neg empty(\mathcal{Q}_l) \Rightarrow x_k(head(\mathcal{Q}_k)) \wedge x_l(head(\mathcal{Q}_l)) \wedge head(\mathcal{Q}_k) \|_s head(\mathcal{Q}_l)$

So, if $detected = true$, then no queue is empty and the global state consisting of the heads of the queues must be strong consistent and c must hold.

We now prove that if there is a strong consistent global state G such that c holds, then the algorithm must return $detected = true$. The proof is by contradiction. Assume that the algorithm returns $detected = false$. Since some queue must be empty in the end, at least one state in G must be deleted in line L15. Let $G[j]$ be the state in G to be deleted first in line L15. Consider the iteration of loop L2 in which this happens. Since $c(G)$ holds, $G[j]$ could not have been added to *newlow* in line L6. So it must have been added in line L10 or L12. In either case, $G[j] \rightarrow_s head(\mathcal{Q}_i)$, for some queue $\mathcal{Q}_i$. Since $G[j]$ is the first state in G to be deleted, $head(\mathcal{Q}_i) \leq G[i]$. So by Property P2, $G[j] \rightarrow_s G[i]$ which contradicts the strong consistency of G. □

Having established that CPR can be solved efficiently, we now address CPS. For a send-ordered strong causality diagram, we can simply make use of symmetry to transform the strong causality diagram and convert the problem to CPR. Given a send-ordered strong causality diagram $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$, we define its *inversion*, $inv(D)$ as the strong causality diagram $(S_1, S_2, \ldots S_n, \rightsquigarrow', <_s')$ where $\rightsquigarrow' = \{(s,t) |\; (t,s) \in \rightsquigarrow\}$ and $<_s' = \{(s,t) |\; (t,s) \in <_s\}$. We now have:

Theorem 5: *CPS for a send-ordered strong causality diagram $D = (S_1, S_2, \ldots S_n, <_s, \rightsquigarrow)$ and conjunctive predicate c is equivalent to CPR for $inv(D) = (S_1, S_2, \ldots S_n, \rightsquigarrow', <_s')$ and conjunctive predicate c.*

Proof: First, the set of receive states $Rcv'(S)$ for $inv(D)$ is the same as the set of send states $Snd(S)$ for D (from the definition of $\rightsquigarrow'$). So $inv(D)$ is receive-ordered.

If G is consistent in D then for any i, j, $G[i] \not\rightarrow_s G[j]$ and $G[j] \not\rightarrow_s G[i]$. So $G[j] \not\rightarrow_s' G[i]$ and $G[i] \not\rightarrow_s' G[j]$ (from the definition of $\rightsquigarrow'$ and $<_s'$). So G is consistent in $inv(D)$.

Using a similar argument, if G is a consistent global state in $inv(D)$ then G is also consistent in D. □

If m is a bound on $|S_i|$ and e is the size of the transitive reduction of $<_s$, then we can deduce that the time complexity of applying RECEIVE-SORT to each process is $O(mn + e)$ and the time complexity of PRED-DETECT is $O(mn^2)$. So the time complexity to solve CPR or CPS is $O(mn^2 + e)$.

Having efficiently solved CPR and CPS, we now take another look at the general problem CPG. We know from Theorem 1 that CPG is NP-Complete. So, it a polynomial solution to CPG is unlikely. Two naive exponential solutions are possible. Let m be a bound on $|S_i|$.The first solution enlists every global state and checks if it is consistent, a process which takes $O(m^n n^2)$ time. The second applies a predicate detection algorithm (such as in [7]) to every local linearization of the strong causality diagram, which takes $O(m^{mn} mn^2)$ time.

However, these solutions do not perform any better for strong causality diagrams which are "close" to being send-ordered or receive-ordered. For example, in a strong causality diagram which has two possible linearizations of receive states in one process, we would expect not to have to pay the full price of the above naive solutions. We now provide a solution that degrades gracefully for diagrams that are close to being send-ordered or receive-ordered.

Let k_i be a bound on the number of linearizations of the $<_s$ relation restricted to the set of receive states in S_i. If we linearize for each process, we can construct a receive-ordered strong causality diagram by adding the ordering of receive states imposed by the linearizations. For all such possible combinations of linearizations, there would be $k = k_1 \times k_2 \times \ldots k_n$ possible receive-ordered strong causality diagrams. We know from Theorem 2 that applying predicate detection to each such receive-ordered strong causality diagrams would be equivalent to applying predicate detection to the original strong causality diagram. So we can solve CPG by applying our algorithm for CPR to k receive-ordered strong causality diagrams, taking $O(k(mn^2 + e))$. Notice that this degrades to the second naive approach in the worst case but achieves good results if k is small, or the original strong causality diagram is close to being receive-ordered. A similar approach could be used if the diagram were close to being send-ordered. Further, by decomposing it into receive-ordered diagrams instead of happened-before diagrams, we save an exponential number of applications of a predicate detection algorithm as compared to the second naive approach.

5 Applications

Applying Strong Causality Diagrams:

The main application of predicate detection has been in distributed debugging and testing. A trace of the distributed computation is taken at run-time and provides the information necessary for the model of a distributed computation. Our extension to the strong causality model allows us to debug distributed programs with multi-threaded processes.

The usual practical representation of the happened-before or causally precedes relation, $\rightarrow$, has been using vector clocks [9]. Two states can then be easily checked for their $\rightarrow$ relationship by comparing their vector clocks. Since we allow partial orders on each of the processes, we must extend the vector clocks to a vector of partially ordered logical clocks [5]. Each such clock value would be an unbounded set in the general case. However, since, in practice, we can place a predefined bound on the number of concurrent threads, we can represent each partially ordered logical clock as a fixed-size vector. The total clock size would then be $n \times l$ where n is the number of processes and l is the maximum number of concurrent threads on a process. Given such "expanded" clocks of two states, we can check their $\rightarrow_s$ relationship in constant time (if we know which processes and threads the two states belong to).

The expanded clocks keep track of the $\leadsto$ relation by logging send and receive events. The $<_s$ relation is a little more involved because we have to decide when two states are independent. In multi-threaded processes, we can keep track of all fork and join points and inter-thread communications through shared memory as described in [5]. Although multi-threaded processes are the most direct application of strong causality diagrams, there are other types of independent events that may be identified in processes. Within the strong causality diagram model, two events are independent so long as reversing their order of execution does not change the values of any of the predicates that may be applied to them. This allows a wide variety of independences to be defined. For example, operations on independent objects are independent, and so are receives or sends on independent ports.

Applying Conjunctive Predicates:

Local predicates are any boolean-valued functions defined on the states in a process. In practice, these states are the values of the variables defining the state of execution of the process. An example of a local predicate would be to check if a program reaches a certain function in the program text. This is equivalent to checking if the program counter variable for any thread reaches the function.

Global predicates can be any conjunctive predicates. Further, we can detect any predicates that may be reduced to conjunctive predicates. Any predicate that is a boolean expression (i.e. expression on local predicates using $\neg, \wedge, \vee$) may be converted into disjunctive normal form and we may then apply a detection algorithm for each of the conjunctions independently. Further, any global predicate that is only satisfied by a finite set of global states may be expressed (though inefficiently) as a boolean expression of local predicates [7]. An example of a global predicate would be detecting if functions on two processes are entered at the same time, violating a required mutual-exclusion property.

Applying Our Results:

We have provided efficient solutions to the conjunctive predicate detection problems for two classes of strong causality diagrams – receive-ordered and send-ordered. As we will now illustrate, these restrictions are met by many distributed computations in practice.

A scenario that arises very often, especially in client-server systems, is:

```
repeat
    receive a request ;
    create a thread to process the request
until done
```

It is clear that such a scenario is receive-ordered even though the sends and per-request processing may be independent.

Another scenario that is often used to model synchronous rounds is:

```
repeat
    receive and process messages
                  until time = end-of-round ;
    send messages
until done
```

If the sends in a round occur in a fixed order or use the same port, then they are totally ordered while receives and local processing may be independent. Thus, a distributed computation resulting from such a program would be send-ordered.

Thus, very often the natural design of distributed programs involves totally ordering the sends and/or the receives.

References

[1] O. Babaoglu and K. Marzullo. Consistent global states of distributed systems: fundamental concepts and mechanisms. In S. Mullender, editor, *Distributed Systems*, chapter 4. Addison-Wesley, 1993.

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

[3] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. In *Proc. of the 11th Symp. on Operating System Principles*, pages 44 – 57. ACM, 1993.

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

[5] C. Fidge. Logical time in distributed computing systems. *IEEE Computer*, 24(8):28 – 33, August 1991.

[6] V. K. Garg. Observation of global properties in distributed systems. In *Proceedings of the IEEE International Conference on Software and Knowledge Engineering*, pages 418 – 425, Lake Tahoe, Nevada, 1996.

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

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

[9] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms: Proc. of the International Workshop on Parallel and Distributed Algorithms*, pages 215 – 226. Elsevier Science Publishers B. V. (North Holland), 1989.

[10] V. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33 – 71, 1986.

[11] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson. Eraser: A dynamic data race detector for multi-threaded programs. In *Proc. of the 16th Symp. on Operating System Principles*. ACM, October 1997. (To be published).