The Complexity of Dynamic Data Race Prediction

Umang Mathur
University of Illinois, Urbana Champaign
USA
umathur3@illinois.edu

Andreas Pavlogiannis
Aarhus University
Denmark
pavlogiannis@cs.au.dk

Mahesh Viswanathan
University of Illinois, Urbana Champaign
USA
vmahesh@illinois.edu

Abstract

Writing concurrent programs is notoriously hard due to scheduling non-determinism. The most common concurrency bugs are data races, which are accesses to a shared resource that can be executed concurrently. Dynamic data-race prediction is the most standard technique for detecting data races: given an observed, data-race-free trace t, the task is to determine whether t can be reordered to a trace t^* that exposes a data-race. Although the problem has received significant practical attention for over three decades, its complexity has remained elusive. In this work, we address this lacuna, identifying sources of intractability and conditions under which the problem is efficiently solvable. Given a trace t of size n over k threads, our main results are as follows.

First, we establish a general $O(k \cdot n^{2\cdot(k-1)})$ upper-bound, as well as an $O(n^k)$ upper-bound when certain parameters of t are constant. In addition, we show that the problem is NP-hard and even W[1]-hard parameterized by k, and thus unlikely to be fixed-parameter tractable. Second, we study the problem over acyclic communication topologies, such as server-clients hierarchies. We establish an $O(k^2 \cdot d \cdot n^2 \cdot \log n)$ upper-bound, where d is the number of shared variables accessed in t. In addition, we show that even for traces with $k = 2$ threads, the problem has no $O(n^{2-\epsilon})$ algorithm under the Orthogonal Vectors conjecture. Since any trace with 2 threads defines an acyclic topology, our upper-bound for this case is optimal up to polynomial improvements for up to moderate values of k and d. Finally, motivated by existing heuristics, we study a distance-bounded version of the problem, where the task is to expose a data race by a witness trace that is similar to t. We develop an algorithm that works in $O(n)$ time when certain parameters of t are constant.

LICS '20, July 8–11, 2020, Saarbrücken, Germany

ACM ISBN 978-1-4503-7104-9/20/07…$15.00
https://doi.org/10.1145/3373718.3394783

CCS Concepts: • **Theory of computation → Parameterized complexity and exact algorithms**; • **Software and its engineering → Software testing and debugging**.

Keywords: Data Race Prediction, Complexity

ACM Reference Format:

Umang Mathur, Andreas Pavlogiannis, and Mahesh Viswanathan. 2020. The Complexity of Dynamic Data Race Prediction. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3373718.3394783

1 Introduction

A concurrent program is said to have a data race if it can exhibit an execution in which two conflicting accesses[1] to the same memory location are "concurrent". Data races in concurrent programs are often symptomatic of bugs in software like data corruption [6, 19, 27], pose challenges in defining the semantics of programming languages, and have led to serious problems in the past [44]; it is no surprise that data races have been deemed *pure evil* [7]. Automatically finding data races in programs remains a widely studied problem because of its critical importance in building correct concurrent software. Data-race detection techniques can broadly be classified into static and dynamic. Given that the race-detection problem in programs is undecidable, static race detection approaches [26, 34] are typically conservative, produce false alarms, and do not scale to large software. On the other hand, since dynamic approaches [15, 24, 33, 37] have the more modest goal of discovering data races by analyzing a *single* trace, they are lightweight, and can often scale to production-level software. Moreover, many dynamic approaches are *sound*, i.e., do not raise false race reports. The effectiveness and scalability of dynamic approaches has lead to many practical advances on the topic. Despite a wide-spread interest on the problem, characterizing its complexity has remained elusive.

Informally, the *dynamic race prediction* problem is the following: given an observed trace t of a multi-threaded program, determine if t demonstrates the presence of a data race in the program that generates t. This means that either t has two

[1] Two accesses are conflicting if they access the same memory location, with one of them being a write access.

conflicting data accesses that are concurrent, or a different trace resulting from scheduling the threads of t in a different order, witnesses such a race. Additional traces that result from alternate thread schedules are captured by the notion of a *correct reordering* of t, that characterizes a set of traces that can be exhibited by *any* program that can generate t; a precise definition of correct reordering is given in Section 2.1. So formally, the data race prediction problem is, given a trace t, determine if there is a correct reordering of t in which a pair of conflicting data accesses are concurrent.

While the data race prediction problem is clearly in NP — guess a correct reordering and check if it demonstrates a data race — its precise complexity has not been identified. Evidence based on prior work, suggests a belief that the problem might be NP-complete. First, related problems, like data-race detection for programs with strong synchronization primitives [28–30], or verifying sequential consistency [16], are known to be NP-hard. Second, all known "complete" algorithms run in worst-case exponential time. These approaches either rely on an explicit enumeration of all correct reorderings [10, 38], or they are symbolic approaches that reduce the race prediction problem to a constraint satisfaction problem [18, 36, 42]. On the other hand, a slew of "partial order"-based methods have been proposed, whose goal is to predict data races in polynomial time, but at the cost of being incomplete and failing to detect data races in some traces. These include algorithms based on the classical *happens-before* partial order [15, 21, 22, 24, 39], and those based on newer partial orders that improve the prediction of data races over happens-before [20, 32, 35, 41].

In this paper we study the problem of data-race prediction from a complexity-theoretic perspective. Our goal is to understand whether the problem is intractable, the causes for intractability, and conditions under it can can be solved efficiently. We provide partial answers to all these questions, and in some cases characterize the tractability/intractability landscape precisely in the form of optimality results.

Contributions. Consider an input trace t of size n over k threads. Our main contributions are as follows. We refer to Section 2.3 for a formal summary.

Our first result shows that the data-race prediction problem is solvable in $O(k \cdot n^{2\cdot(k-1)})$ time, and can be improved to $O(n^k)$ when certain additional parameters of t are constant. We note that most benchmarks used in practice have a constant number of threads [15, 20, 22, 32, 35, 41] , and in such cases our upper-bound is polynomial.

The observation that data race predication is in polynomial time for constantly many threads naturally leads to two follow-up questions. Does the problem remain tractable for any k? And if not, is it fixed parameter tractable (FPT) with respect to k, i.e., is there an algorithm with running time of the form $O(f(k) \cdot n^{O(1)})$? Our second result answers both these questions in the negative, by showing that the problem is W[1]-hard. This formally establishes the NP-hardness of the problem, and excludes efficient algorithms when k is even moderately large (e.g., $k = \Omega(\log n)$).

We then investigate whether there are practically relevant contexts where data-race prediction is more efficiently solvable, i.e., the degree of the polynomial is fixed and independent of k. We consider the case of traces over acyclic communication topologies, such as pipelines, server-clients hierarchies and divide-and-conquer parallelism. Our third result shows that, perhaps surprisingly, over such topologies data-race prediction can be solved in $O(k^2 \cdot d \cdot n^2 \cdot \log n)$ time, where d is the total number of synchronization variables (locks) and global memory locations.

In practice, the size n of the trace is by far the dominating parameter, while k and d are many orders of magnitude smaller. Hence, given the above upper-bound, the relevant question is whether the complexity on n can be improved further. Our fourth result shows that this is unlikely: we show that, under the Orthogonal Vectors conjecture, there is no $O(n^{2-\epsilon})$ algorithm even for traces restricted to only 2 threads. As any trace with 2 threads induces an acyclic topology, our upper-bound is (conditionally) optimal up to polynomial improvements.

Finally, the majority of practical data-race prediction heuristics search for a data race witness among correct reorderings that are very similar to the observed trace t, i.e., by only attempting a few event reorderings on t. Motivated by these approaches, we investigate the complexity of a distance-bounded version of data-race predication, where the goal is to expose a data race by only looking at correct reorderings of t that are a small distance away. Here, distance between traces is measured by the number of critical sections and write events whose order is reversed from t. Our fifth result is a linear-time (and thus, optimal) algorithm for this problem, when certain parameters of the trace t are constant. This result gives a solid basis for the principled development of fast heuristics for dynamic data-race prediction.

Technical contributions. Towards our main results, we make several technical contributions that might be of independent interest. We summarize some of them below.

1. We improve the lower-bound of the well-known problem on verifying sequential consistency with read-mapping (VSC-rm) [16] from the long-lasting NP-hardness to W[1]-hard.
2. We show that VSC-rm can be solved efficiently on tree communication topologies of any number of threads, which improves a recent result of [32] for only 2 threads, as well as a result of [9] for more than 2 threads.
3. The first challenge in data-race prediction given a trace t is to choose the set X of events of t over which to attempt to construct a correct reordering. Identifying

such choices for X is a significant challenge [18, 32, 35]. We establish non-trivial upper-bounds on the number of choices for X, and show that they are constantly many when certain parameters of t are constant.
4. Particularly for tree communication topologies, we show that a single choice for such X suffices.

Finally, we note that our notion of a predictable data race in a trace t requires as a witness a reordering t^* of t in which every read event reads from the same write event as in t. This guarantees that t^* is valid in any program that produced t. More permissive reorderings, e.g., requiring that every read event reads the same value, are also possible, and can capture potentially more races. Our notion of witness reflects the most common practice in race-detection literature, where trace logging typically does not track the values.

Related Work. Antoni Mazurkiewicz [1, 25] used the notion of *traces* to mathematically model executions of concurrent programs. Bertoni et. al. [4] studied various language-theoretic questions about Mazurkiewicz traces. The folklore results about the NP-hardness of race detection are often attributed to Netzer and Miller [28–30]. However, the problem considered in their work differs in significant ways from the problem of data-race prediction. First, the notion of feasible executions in [28] (the counterpart of the notion of correct reorderings) requires that any two conflicting events be ordered in the same way as the observed execution, and hence, are less permissive. Next, the NP-hardness arises from the use of complex synchronization primitives like `wait` and `signal`, which are more powerful than the primitives we study here (release/acquire of locks and read/write of registers). The results due to Netzer and Miller, thus, do not apply to the problem of data-race prediction. Gibbons and Korach [16] establish NP-hardness for a closely related problem in distributed computing — verifying sequential consistency with a read mapping (VSC-rm). Yet again, the problem is different than the problem of race prediction. Complexity theoretic investigations have also been undertaken for other problems in distributed computing like linearizability [13, 16, 17], serializability [31] and transactional consistency [5]. Hence, although there have been many theoretical results on related problems in concurrency, none of them addresses dynamic data-race prediction. Our work fills this gap. Some proofs are relegated to a technical report [23].

2 Preliminaries

2.1 Model

General notation. Given a natural number k, let $[k] = \{1, \ldots, k\}$. Given a function $f : X \to Y$, we let $\mathrm{dom}(f) = X$ and $\mathrm{img}(f) = Y$. Given two functions f, g, we write $f \subseteq g$ to denote that $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and for every $x \in \mathrm{dom}(f)$ we have $f(x) = g(x)$. Given a set $X' \subseteq \mathrm{dom}(f)$, we denote by $f|X'$ the function with $\mathrm{dom}(f|X') = X'$ and $f|X' \subseteq f$.

Concurrent program. We consider a shared-memory concurrent program $\mathcal{P}$ that consists of k threads $\{p_i\}_{i\in[k]}$, under sequential consistency semantics [40]. For simplicity of presentation we assume no thread is created dynamically and the set $\{p_i\}_{i\in[k]}$ is known a-priori. Communication between threads occurs over a set of global variables $\mathcal{G}$, and synchronization over a set of locks $\mathcal{L}$ such that $\mathcal{G} \cap \mathcal{L} = \emptyset$. We let $\mathcal{V} = \mathcal{G} \cup \mathcal{L}$ be the set of all variables of $\mathcal{P}$. Each thread is deterministic, and performs a sequence of operations. We are only interested in the operations that access a global variable or a lock, which are called *events*. In particular, the allowed events are the following.

1. Given a global variable $x \in \mathcal{G}$, a thread can either *write* to x via an event $\mathsf{w}(x)$ or *read* from x via an event $\mathsf{r}(x)$.
2. Given a lock $\ell \in \mathcal{L}$, a thread can either *acquire* ℓ via an event $\mathrm{acq}(\ell)$ or *release* ℓ via an event $\mathrm{rel}(\ell)$.

Each event is atomic, represented by a tuple (a, b, c, d), where

1. $a \in \{\mathsf{w}, \mathsf{r}, \mathrm{acq}, \mathrm{rel}\}$ represents the type of the event (i.e., write, read, lock-acquire or lock-release event),
2. b represents the variable or lock that the event accesses,
3. c is the thread of the event, and
4. d is a unique identifier of the event.

Given an event e, we let $\mathrm{loc}(e)$ denote the global variable or lock that e accesses. We occasionally write $e(x)$ to denote an event e with $\mathrm{loc}(e) = x$, while the thread and event id is often implied by the context. We denote by $\mathcal{W}_p$ (resp. $\mathcal{R}_p$, $\mathcal{L}^A_p$, $\mathcal{L}^R_p$) the set of all write (resp. read, lock-acquire, lock-release) events that can be performed by thread p. We let $\mathcal{E}_p = \mathcal{W}_p \cup \mathcal{R}_p \cup \mathcal{L}^A_p \cup \mathcal{L}^R_p$. We denote by $\mathcal{E} = \bigcup_p \mathcal{E}_p$, $\mathcal{W} = \bigcup_p \mathcal{W}_p$, $\mathcal{R} = \bigcup_p \mathcal{R}_p$, $\mathcal{L}^A = \bigcup_p \mathcal{L}^A_p$, $\mathcal{L}^R = \bigcup_p \mathcal{L}^R_p$ the events, write, read, lock-acquire and lock-release events of the program $\mathcal{P}$, respectively. Given an event $e \in \mathcal{E}$, we denote by $\mathrm{p}(e)$ the thread of e. Finally, given a set of events $X \subseteq \mathcal{E}$, we denote by $\mathcal{R}(X)$ (resp., $\mathcal{W}(X)$, $\mathcal{L}^A(X)$, $\mathcal{L}^R(X)$) the set of read (resp., write, lock-acquire, lock-release) events of X. For succinctness, we let $\mathcal{WR}(X) = \mathcal{W}(X) \cup \mathcal{R}(X)$, $\mathcal{RL}(X) = \mathcal{R}(X) \cup \mathcal{L}^R(X)$ and $\mathcal{WL}(X) = \mathcal{W}(X) \cup \mathcal{L}^A(X)$. The semantics of $\mathcal{P}$ are the standard for sequential consistency [40].

Conflicting events. Given two distinct events $e_1, e_2 \in \mathcal{E}$, we say that e_1 and e_2 are *conflicting*, denoted by $e_1 \bowtie e_2$, if (i) $\mathrm{loc}(e_1) = \mathrm{loc}(e_2)$ (i.e., both events access the same global variable or the same lock) and (ii) $\{e_1, e_2\} \cap \mathcal{W} \neq \emptyset$ or $\{e_1, e_2\} \cap \mathcal{L}^A \neq \emptyset$ i.e., at least one of them is either a write event or a lock-acquire event. We extend the notion of conflict to sets of events in the natural way: two sets of events $X_1, X_2 \subseteq \mathcal{E}$ are called *conflicting*, denoted by $X_1 \bowtie X_2$ if $\exists (e_1, e_2) \in (X_1 \times X_2)$ such that $e_1 \bowtie e_2$.

Event sequences. Let t be a sequence of events. We denote by $\mathcal{E}(t)$ the set of events, by $\mathcal{L}(t)$ the set of locks, and by $\mathcal{G}(t)$ the set of global variables in t. We let $\mathcal{W}(t)$ (resp.,

$\mathcal{R}(t)$, $\mathcal{L}^A(t)$, $\mathcal{L}^R(t)$) denote the set $\mathcal{W}(\mathcal{E}(t))$ (resp., $\mathcal{R}(\mathcal{E}(t))$, $\mathcal{L}^A(\mathcal{E}(t))$, $\mathcal{L}^R(\mathcal{E}(t))$), i.e., it is the set of write (resp., read, lock-acquire, lock-release) events of t. Given two distinct events $e_1, e_2 \in \mathcal{E}(t)$, we say that e_1 *is earlier than* e_2 in t, denoted by $e_1 <_t e_2$ iff e_1 appears before e_2 in t. We say that e_1 is *thread-ordered earlier than* e_2, denoted $e_1 <_{\mathsf{TO}(t)} e_2$, when $e_1 <_t e_2$ and $\mathsf{p}(e_1) = \mathsf{p}(e_2)$. For events $e_1, e_2 \in \mathcal{E}(t)$, we say $e_1 \leq_t e_2$ (resp. $e_1 \leq_{\mathsf{TO}(t)} e_2$) if either $e_1 = e_2$ or $e_1 <_t e_2$ (resp. $e_1 <_{\mathsf{TO}(t)} e_2$). We will often use $<_{\mathsf{TO}}$ (resp. $\leq_{\mathsf{TO}}$) in place of $<_{\mathsf{TO}(t)}$ (resp. $\leq_{\mathsf{TO}(t)}$) when the trace t is clear from context. Given a set of events $X \subseteq \mathcal{E}$, we denote by $t|X$ the *projection* of t onto X. Given a thread p_i, we let $t|p_i = t|\mathcal{E}_{p_i}$. Given two sequences t_1, t_2, we denote by $t_1 \circ t_2$ their concatenation.

Lock events. Given a sequence of events t and a lock-acquire event $\mathsf{acq} \in \mathcal{L}^A(t)$, we denote by $\mathsf{match}_t(\mathsf{acq})$ the earliest lock-release event $\mathsf{rel} \in \mathcal{L}^R(t)$ such that $\mathsf{rel} \bowtie \mathsf{acq}$ and $\mathsf{acq} <_{\mathsf{TO}} \mathsf{rel}$, and let $\mathsf{match}_t(\mathsf{acq}) = \bot$ if no such lock-release event exists. If $\mathsf{match}_t(\mathsf{acq}) \neq \bot$, we require that $\mathsf{p}(\mathsf{acq}) = \mathsf{p}(\mathsf{match}_t(\mathsf{acq}))$, i.e., the two lock events belong to the same thread. Similarly, given a lock-release event $\mathsf{rel} \in \mathcal{L}^R(t)$, we denote by $\mathsf{match}_t(\mathsf{rel})$ the latest acquire event $\mathsf{acq} \in \mathcal{L}^A(t)$ such that $\mathsf{match}_t(\mathsf{acq}) = \mathsf{rel}$ and require that such a lock-acquire event always exists. Given a lock-acquire event acq, the *critical section* $\mathrm{CS}_t(\mathsf{acq})$ is the set of events e such that (i) $\mathsf{acq} <_{\mathsf{TO}} e$ and (ii) if $\mathsf{match}_t(\mathsf{acq}) \neq \bot$, then $e <_{\mathsf{TO}} \mathsf{match}_t(\mathsf{acq})$. For simplicity of presentation, we assume that locks are not re-entrant. That is, for any two lock-acquire events $\mathsf{acq}_1, \mathsf{acq}_2$ with $\mathsf{acq}_1 \bowtie \mathsf{acq}_2$ and $\mathsf{acq}_1 <_{\mathsf{TO}} \mathsf{acq}_2$, we must have $\mathsf{match}_t(\mathsf{acq}_1) <_{\mathsf{TO}} \mathsf{acq}_2$. The *lock-nesting depth* of t is the maximum number ℓ such that there exist distinct lock-acquire events $\{\mathsf{acq}_i\}_{i=1}^{\ell}$ with (i) $\mathsf{acq}_1 <_{\mathsf{TO}} \mathsf{acq}_2 <_{\mathsf{TO}} \cdots <_{\mathsf{TO}} \mathsf{acq}_\ell$, and (ii) for all $i \in [\ell]$, if $\mathsf{match}_t(\mathsf{acq}_i) \in \mathcal{E}(t)$ then $\mathsf{acq}_\ell <_{\mathsf{TO}} \mathsf{match}_t(\mathsf{acq}_i)$.

Traces and reads-from functions. An event sequence t is called a *trace* if for any two lock-acquire events $\mathsf{acq}_1, \mathsf{acq}_2 \in \mathcal{L}^A(t)$, if $\mathsf{loc}(\mathsf{acq}_1) = \mathsf{loc}(\mathsf{acq}_2)$ and $\mathsf{acq}_1 <_t \mathsf{acq}_2$, then $\mathsf{rel}_1 = \mathsf{match}_t(\mathsf{acq}_1) \in \mathcal{L}^R(t)$ and $\mathsf{rel}_1 <_t \mathsf{acq}_2$. A trace therefore ensures that locks obey mutual exclusion, i.e., critical sections over the same lock cannot overlap.

Given a trace t, we define its *reads-from function* $\mathrm{RF}_t : \mathcal{R}(t) \to \mathcal{W}(t)$ as follows: $\mathrm{RF}_t(\mathsf{r}) = \mathsf{w}$ iff $\mathsf{w} <_t \mathsf{r}$ and $\forall \mathsf{w}' \in \mathcal{W}(t)$ with $\mathsf{w} \bowtie \mathsf{w}'$, we have $\mathsf{w}' <_t \mathsf{r} \Rightarrow \mathsf{w}' <_t \mathsf{w}$. That is, RF_t maps every read event r to the write event w that r observes in t. For simplicity, we assume that t starts with a write event to every location, hence RF_t is well-defined. For notational convenience, we extend the reads-from function RF_t to lock-release events, such that, for any lock-release event $\mathsf{rel} \in \mathcal{L}^R(t)$, we have $\mathrm{RF}_t(\mathsf{rel}) = \mathsf{match}_t(\mathsf{rel})$, i.e., rel observes its matching lock acquire event.

Correct reordering, enabled events and predictable data races. A trace t^* is a correct reordering of trace t if (i) $\mathcal{E}(t^*) \subseteq \mathcal{E}(t)$, (ii) for every thread p_i, we have that $t^*|p_i$ is a prefix of $t|p_i$, and (iii) $\mathrm{RF}_{t^*} \subseteq \mathrm{RF}_t$, i.e., the reads-from functions of t^* and t agree on their common read and lock-release events. Given a trace t, an event $e \in \mathcal{E}(t)$ and a correct reordering t^* of t, we say that e is *enabled* in t^* if $e \notin \mathcal{E}(t^*)$ and for every $e' \in \mathcal{E}(t)$ such that $e' <_{\mathsf{TO}} e$, we have that $e' \in \mathcal{E}(t^*)$. Given two conflicting events $e_1, e_2 \in \mathcal{E}(t)$ with $\mathsf{loc}(e_1) = \mathsf{loc}(e_2) \in \mathcal{G}$, we say the pair (e_1, e_2) is a predictable data race of trace t if there is a correct reordering t^* of t such that both e_1 and e_2 are enabled in t^*. Finally, we say t has a predictable data race if there is a pair (e_1, e_2) which is a predictable data race of t.

Note that predictability of a race is defined with respect to a correct reordering in which every read event observes the same write event. This requirement guarantees that the correct reordering is a valid trace of any concurrent program that produced the initial trace. Hence, every such program is racy. More permissive notions of predictability can also be defined, e.g., by requiring that, in a correct reordering, every read event reads the same value (possibly from a different write event). This alternative definition would capture potentially more predictable races. Our definition of correct reorderings reflects the most common practice in race-detection literature, where trace logging typically does not track the values [20, 22, 32, 35, 41].

The communication topology. The trace t naturally induces a *communication topology* graph $G = (V, E)$ where (i) $V = \{p_i\}_i$ and (ii) $E = \{(p_i, p_j) \mid i \neq j \text{ and } \mathcal{E}(p_i) \bowtie \mathcal{E}(p_j)\}$. In words, we have one node in G per thread, and there is an edge between two distinct nodes if the corresponding threads execute conflicting events (note that G is undirected). For simplicity, we assume that G is connected. In later sections, we will make a distinction between tree topologies (i.e., that do not contain cycles) and general topologies (that might contain cycles). Common examples of tree topologies include stars (e.g., server-clients), pipelines, divide-and-conquer parallelism, and the special case of two threads.

2.2 Problem Statement

In the dynamic data-race prediction problem, we are given an observed trace t, and the task is to identify whether t has a predictable data race. In this work we focus on the following decision problem — given a trace t and two (read or write) conflicting events $e_1, e_2 \in \mathcal{E}(t)$, the task is to decide whether (e_1, e_2) is a predictable data race of t. Clearly, having established the complexity of the decision problem, the general problem can be solved by answering the decision problem for all $O(n^2)$ pairs of conflicting variable access events of t. In the other direction, as the following lemma observes, detecting whether t has some predictable data race is no easier than detecting whether a given event pair of t constitutes a predictable data race.

Lemma 2.1. *Given a trace t of length n and two events $e_1, e_2 \in \mathcal{E}(t)$, we can construct a trace t' in $O(n)$ time so that t' has a predictable data race iff (e_1, e_2) is a predictable data race of t .*

To make the presentation simpler, we assume w.l.o.g that there are no open critical sections in t, i.e., every lock-acquire event acq is followed by a matching lock-release event $\mathsf{match}_t(\mathsf{acq})$. Motivated by practical applications, we also study the complexity of dynamic data-race prediction parameterized by a notion of distance between the input trace t and the witness t^* that reveals the data race.

Trace distances. Consider a trace t and a correct reordering t' of t. The set of *reversals* between t and t' is defined as

$$\mathrm{Rv}(t, t') = \{(\mathsf{w}_1, \mathsf{w}_2) \in \mathcal{WL}(t') \times \mathcal{WL}(t') \mid \mathsf{w}_1 \bowtie \mathsf{w}_2 \text{ and } \mathsf{w}_1 <_t \mathsf{w}_2 \text{ and } \mathsf{w}_2 <_{t'} \mathsf{w}_1\} .$$

In words, $\mathrm{Rv}(t, t')$ contains the pairs of conflicting write events or lock-acquire events, the order of which has been *reversed* in t' when compared to t. The *distance* of t' from t is defined as $\delta(t, t') = |\mathrm{Rv}(t, t')|$. Our notion of distance, thus, only counts the number of reversals of conflicting write or lock-acquire events instead of counting reversals over all events (or even conflicting write-read events).

Distance-bounded dynamic data race prediction. Consider a trace t and two events e_1, e_2 of t. Given an integer $\ell \geq 0$, the ℓ-distance-bounded dynamic data-race prediction problem is the promise problem[2] that allows for any answer (True/False) if (e_1, e_2) is a predictable data race of t and every witness correct reordering t^* is such that $\delta(t, t^*) > \ell$.

2.3 Summary of Main Results

Here we state the main results of this work, and present the technical details in the later parts of the paper.

2.3.1 The General Case. First, we study the complexity of the problem with respect to various parameters of the input trace. These parameters are the number of threads, the number of variables, the lock-nesting depth, as well as the lock-dependence factor, which, intuitively, measures the amount of data flow between critical sections. In the following, $<_{\mathsf{TRF}}$ (formal definition in Section 3), is the smallest partial order that contains $<_{\mathsf{TO}}$, and also orders read events after their corresponding observed write event $\big($i.e., $\mathrm{RF}(\mathsf{r}) <_{\mathsf{TRF}} \mathsf{r}$ for every $\mathsf{r} \in \mathcal{RL}(\mathcal{E}(t))\big)$.

The lock-dependence factor. The *lock-dependence graph* of a trace t is the graph $G_t = (V_t, E_t)$ defined as follows.

1. The set of vertices is $V_t = \mathcal{L}^A(t)$, i.e., it is the set of lock-acquire events of t.
2. The set of edges is such that $(\mathsf{acq}_1, \mathsf{acq}_2) \in E_t$ if (i) $\mathsf{acq}_1 \not<_{\mathsf{TRF}} \mathsf{acq}_2$, (ii) $\mathsf{acq}_1 <_{\mathsf{TRF}} \mathsf{match}_t(\mathsf{acq}_2)$, and (iii) $\mathsf{match}_t(\mathsf{acq}_1) \not<_{\mathsf{TRF}} \mathsf{match}_t(\mathsf{acq}_2)$.

[2]The promise problem ([14]) given languages L_{True} and L_{False} is to design an algorithm A such that $A(x) = \mathsf{True}$ for every $x \in L_{\mathsf{True}}$, $A(x) = \mathsf{False}$ for every $x \in L_{\mathsf{False}}$, and $A(x)$ is any of True or False if $x \notin L_{\mathsf{True}} \cup L_{\mathsf{False}}$.

Given a lock-acquire event $\mathsf{acq} \in V_t$, let A_{acq} be the set of lock-acquire events that can reach acq in G_t. We define the *lock dependence factor* of t as $\max_{\mathsf{acq} \in V_t} |A_{\mathsf{acq}}|$. We show the following theorem.

Theorem 2.2. *Consider a trace t of length n, k threads, lock-nesting depth γ, and lock-dependence factor ζ. The dynamic data-race prediction problem on t can be solved in $O(\alpha \cdot \beta)$ time, where $\alpha = \min(n, k \cdot \gamma \cdot \zeta)^{k-2}$ and $\beta = k \cdot n^k$.*

In particular, the problem is polynomial-time solvable for a fixed number of threads k. In practice, the parameters k, γ and ζ behave as constants, and in such cases our upper-bound becomes $O(n^k)$. Theorem 2.2 naturally leads to two questions, namely (i) whether there is a polynomial-time algorithm for any k, and (ii) if not, whether the problem is FPT with respect to the parameter k, i.e., can be solved in $O(f(k) \cdot n^{O(1)})$ time, for some function f. Question (ii) is very relevant, as typically k is several orders of magnitude smaller than n. We complement Theorem 2.3 with the following lower-bound, which answers both questions in negative.

Theorem 2.3. *The dynamic data-race prediction problem is W[1]-hard parameterized by the number of threads.*

2.3.2 Tree Communication Topologies. Next, we study the problem for tree communication topologies, such as pipelines and server-clients architectures. We show the following theorem.

Theorem 2.4. *Let t be a trace over a tree communication topology with n events, k threads and d variables. The dynamic data-race prediction problem for t can be solved in $O(k^2 \cdot d \cdot n^2 \cdot \log n)$ time.*

Perhaps surprisingly, in sharp contrast to Theorem 2.3, for tree topologies there exists an efficient algorithm where the degree of the polynomial is fixed and does not depend on any input parameter (e.g., number of threads). Note that the dominating factor in this complexity is n^2, while k and d are typically much smaller. Hence, the relevant theoretical question is whether the dependency on n can be improved further. We show that this is unlikely, by complementing Theorem 2.4 with the following conditional lower-bound, based on the Orthogonal Vectors conjecture [8].

Theorem 2.5. *Let t be a trace with n events, $k \geq 2$ threads and $d \geq 9$ shared global variables with at least one lock. There is no algorithm that solves the decision problem of dynamic data-race prediction for t in time $O(n^{2-\epsilon})$, for any $\epsilon > 0$, unless the Orthogonal Vectors conjecture fails.*

Since $k = 2$ implies a tree communication topology, the result of Theorem 2.4 is conditionally optimal, up-to poly-logarithmic factors, for a reasonable number of threads and variables (e.g., when $k, d = \log^{O(1)}(n)$).

2.3.3 Witnesses in Small Distance. Finally, we study the problem in more practical settings, namely, when (i) the number of threads, lock-nesting depth, lock-dependence factor of t are bounded, and (ii) we are searching for a witness at a small distance from t.

Theorem 2.6. *Fix a reversal bound $\ell \geq 0$. Consider a trace t of length n and constant number of threads, lock-nesting depth and lock-dependence factor. The ℓ-distance-bounded dynamic data-race prediction problem for t can be solved in $O(n)$ time.*

3 Trace Ideals

3.1 Partial Orders

Partially ordered sets. A *partially ordered set* (or *poset*) is a pair (X, P) where X is a set of (write, read, lock-acquire, lock-release) events and P is a reflexive, antisymmetric and transitive relation over X. We will often write $e_1 \leq_P e_2$ to denote $(e_1, e_2) \in P$. Given two events $e_1, e_2 \in X$ we write $e_1 <_P e_2$ to denote that $e_1 \leq_P e_2$ and $e_1 \neq e_2$, and write $e_1 \ll_P e_2$ to denote that $e_1 <_P e_2$ and there exists no event e such that $e_1 <_P e <_P e_2$. Given two distinct events $e_1, e_2 \in X$, we say that e_1 and e_2 are *unordered* by P, denoted by $e_1 \parallel_P e_2$, if neither $e_1 <_P e_2$ nor $e_2 <_P e_1$. We call an event $e \in X$ *maximal* if there exists no $e' \in X$ such that $e <_P e'$. Given a set $Y \subseteq X$, we denote by $P|Y$ the *projection* of P on Y, i.e., we have $P|Y \subseteq Y \times Y$, and for all $e_1, e_2 \in Y$, we have $e_1 \leq_{P|Y} e_2$ iff $e_1 \leq_P e_2$. Given two posets (X, P) and (X, Q), we say that the partial order Q *refines* P, denoted by $Q \sqsubseteq P$, if for every two events $e_1, e_2 \in X$, if $e_1 \leq_P e_2$ then $e_1 \leq_Q e_2$. If Q refines P, we say that P is *weaker* than Q. We denote by $Q \sqsubset P$ the fact that $Q \sqsubseteq P$ and $P \not\sqsubseteq Q$. A *linearization* of (X, P) is a total order over X that refines P. An *order ideal* (or simply *ideal*) of a poset (X, P) is subset $Y \subseteq X$ such that for every two events $e_1 \in Y$ and $e_2 \in X$ with $e_2 \leq_P e_1$, we have $e_2 \in Y$. An event e is *executable* in ideal Y if $Y \cup \{e\}$ is also an ideal of (X, P). The number of threads and variables of a poset (X, P) is the number of threads and variables of the events of X.

Partially ordered sets with reads-from functions. A *poset with a reads-from function* (or *rf-poset*) is a tuple (X, P, RF) where (i) $\mathrm{RF}\colon \mathcal{RL}(X) \to \mathcal{WL}(X)$ is a reads-from function such that for all $\mathsf{r} \in \mathcal{RL}(X)$, we have $\mathrm{RF}(\mathsf{r}) \in \mathcal{W}(X)$ iff $\mathsf{r} \in \mathcal{R}(X)$, and (ii) (X, P) is a poset where for all $\mathsf{r} \in \mathcal{RL}(X)$ we have $\mathrm{RF}(\mathsf{r}) <_P \mathsf{r}$. Notation from posets is naturally lifted to rf-posets, e.g., an ideal of $\mathcal{P}$ is an ideal of (X, P).

Thread-reads-from order and trace ideals. Given a trace t, the thread-reads-from order $\mathsf{TRF}(t)$ (or simply TRF when t is clear from context) is the weakest partial order over the set $\mathcal{E}(t)$ such that (i) $\mathsf{TRF} \sqsubseteq \mathsf{TO}$, and (ii) $(\mathcal{E}(t), \mathsf{TRF}, \mathrm{RF}_t)$ is an rf-poset. In particular, TRF is the transitive closure of $\big(\mathsf{TO} \cup \{\mathrm{RF}_t(\mathsf{r}) < \mathsf{r} \mid \mathsf{r} \in \mathcal{R}(t)\}\big)$. A *trace ideal* of t is an ideal X of the poset $(\mathcal{E}(t), \mathsf{TRF})$. We say an event $e \in \mathcal{E}(t) \setminus X$ is *enabled* in X if for every $e' <_{\mathsf{TO}} e$, we have $e' \in X$. We call X *lock-feasible* if for every two lock-acquire events $\mathrm{acq}_1, \mathrm{acq}_2 \in \mathcal{L}^A(X)$ with $\mathrm{acq}_1 \bowtie \mathrm{acq}_2$, we have $\mathrm{match}_t(\mathrm{acq}_i) \in X$ for some $i \in [2]$. We call X *feasible* if it is lock-feasible, and there exists a partial order P over X such that (i) $P \sqsubseteq \mathsf{TRF}|X$ and (ii) for every pair of lock-acquire events $\mathrm{acq}_1, \mathrm{acq}_2 \in \mathcal{L}^A(X)$ with $\mathrm{acq}_1 \bowtie \mathrm{acq}_2$, and $\mathrm{match}_t(\mathrm{acq}_1) \notin X$, we have $\mathrm{rel}_2 <_P \mathrm{acq}_1$, where $\mathrm{rel}_2 = \mathrm{match}_t(\mathrm{acq}_2)$. If X is feasible, we define the *canonical rf-poset* of X as $(X, Q, \mathrm{RF}_t\,|X)$, where Q is the weakest among all such partial orders P. It is easy to see that Q is well-defined, i.e., there exists at most one weakest partial order among all such partial orders P.

The realizability problem of feasible trace ideals. The realizability problem for an rf-poset $\mathcal{P} = (X, P, \mathrm{RF})$ asks whether there exists a linearization t^* of P such that $\mathrm{RF}_{t^*} = \mathrm{RF}$. Given a trace t and a feasible trace ideal X of t, the realizability problem for X is the realizability problem of the canonical rf-poset (X, P, RF) of X. The following remark relates the decision problem of dynamic race prediction in t with the realizability of trace ideals of t.

Remark 1. *If t^* is a witness of the realizability of X, then t^* is a correct reordering of t. Two conflicting events $e_1, e_2 \in \mathcal{E}(t)$ are a predictable data race of t iff there exists a realizable trace ideal X of t such that e_1, e_2 are enabled in X.*

Read pairs and triplets. For notational convenience, we introduce the notion of read pairs and read triplets. Given an rf-poset $\mathcal{P} = (X, P, \mathrm{RF})$, a *read pair* (or *pair* for short) of $\mathcal{P}$ is a pair (w, r) such that $\mathsf{r} \in \mathcal{RL}(X)$ and $\mathsf{w} = \mathrm{RF}(\mathsf{r})$ (note that $\mathsf{w} \in X$). A *read triplet* (or *triplet* for short) is a triplet $(\mathsf{w}, \mathsf{r}, \mathsf{w}')$ such that (i) (w, r) is a pair of $\mathcal{P}$, (ii) $\mathsf{w}' \in X$, and (iii) $\mathsf{w}' \neq \mathsf{w}$ and $\mathsf{r} \bowtie \mathsf{w}'$. We denote by $\mathsf{Pairs}(\mathcal{P})$ and $\mathsf{Triplets}(\mathcal{P})$ the set of pairs and triplets of $\mathcal{P}$, respectively.

Closed rf-posets. We call an rf-poset $\mathcal{P} = (X, P, \mathrm{RF})$ *closed* if for every triplet $(\mathsf{w}, \mathsf{r}, \mathsf{w}') \in \mathsf{Triplets}(\mathcal{P})$, we have (i) if $\mathsf{w}' <_P \mathsf{r}$ then $\mathsf{w}' <_P \mathsf{w}$, and (ii) if $\mathsf{w} <_P \mathsf{w}'$ then $\mathsf{r} <_P \mathsf{w}'$. Given, an rf-poset $\mathcal{P} = (X, P, \mathrm{RF})$, the *closure* of $\mathcal{P}$ is an rf-poset $\mathcal{Q} = (X, Q, \mathrm{RF})$ where Q is the weakest partial order over X such that $Q \sqsubseteq P$ and $\mathcal{Q}$ is closed. If no such Q exists, we let the closure of $\mathcal{P}$ be $\bot$. The closure is well-defined [32]. The associated Closure problem is, given an rf-poset $\mathcal{P}$, decide whether the closure of $\mathcal{P}$ is not $\bot$.

Remark 2. *An rf-poset is realizable only if its closure exists and is realizable.*

3.2 Bounds on the Number of Feasible Trace Ideals

Remark 1 suggests that the dynamic data-race prediction problem for a trace t is reducible to deciding whether t has some realizable trace ideal. In general, if t has length n and k threads, there exist n^k possible trace ideals to test for realizability. Here we derive another upper-bound on the number of such ideals that are sufficient to test, based on the number of threads of t, its lock-nesting depth and its lock-dependence factor. These parameters typically behave as constants in

practice, and thus understanding the complexity of dynamic data race prediction in terms of these parameters is crucial.

Causal cones. Given an event $e \in \mathcal{E}(t)$, the *causal cone* $\mathsf{Cone}_t(e)$ of $e \in \mathcal{E}(t)$ is the smallest trace ideal X of t so that e is enabled in X. In words, we construct $\mathsf{Cone}_t(e)$ by taking the TRF-downwards closure of the thread-local predecessor e' of e (i.e., $e' \ll_{\mathsf{TO}} e$). Given a non-empty set of events $S \subseteq \mathcal{E}(t)$, we define the causal cone of S as $\mathsf{Cone}_t(S) = \bigcup_{e \in S} \mathsf{Cone}_t(e)$; notice that $\mathsf{Cone}_t(S)$ is a trace ideal.

Candidate ideal set. Given a set of events X, we denote by $\mathrm{OpenAcqs}(X)$ the set of lock-acquire events acq such that $\mathsf{match}_t(\mathrm{acq}) \notin X$. Given two events $e_1, e_2 \in \mathcal{E}(t)$, the *candidate ideal set* $\mathsf{CIS}_t(e_1, e_2)$ of e_1, e_2 is the smallest set of trace ideals of t such that the following hold.

1. $\mathsf{Cone}_t(\{e_1, e_2\}) \in \mathsf{CIS}_t(e_1, e_2)$.
2. Let $Y \in \mathsf{CIS}_t(e_1, e_2)$, $\mathrm{acq} \in \mathrm{OpenAcqs}(Y)$, $\mathrm{rel} = \mathsf{match}_t(\mathrm{acq})$, and $Y' = \mathsf{Cone}_t(Y \cup \{\mathrm{rel}\}) \cup \{\mathrm{rel}\}$. If $e_1, e_2 \notin Y'$, then $Y' \in \mathsf{CIS}_t(e_1, e_2)$.

In light of Remark 1, we will decide whether (e_1, e_2) is a predictable data race by deciding the realizability of ideals in the candidate ideal set. Item 2 states that, as long as there is some ideal Y in the candidate ideal set such that Y leaves some critical section open, we construct another ideal $Y' \supset Y$ by choosing one such open critical section and closing it, and add Y' in the candidate set as well. Intuitively, the open critical section of Y might deem Y not realizable, while closing that critical section might make Y' realizable. Clearly, if $e_1 \in Y'$ or $e_2 \in Y'$, then the realizability of Y' does not imply a data race on e_1, e_2 as one of the two events is not enabled in Y' (Remark 1). As the following lemma shows, in order to decide whether (e_1, e_2) is a predictable data race of t, it suffices to test for realizability all the ideals in $\mathsf{CIS}_t(e_1, e_2)$.

Lemma 3.1. *(e_1, e_2) is a predictable data race of t iff there exists a realizable ideal $X \in \mathsf{CIS}_t(e_1, e_2)$ such that $e_1, e_2 \notin X$.*

The following lemma gives an upper-bound on $|\mathsf{CIS}_t(e_1, e_2)|$, i.e., on the number of ideals we need to test for realizability.

Lemma 3.2. *We have $|\mathsf{CIS}_t(e_1, e_2)| \leq \min(n, \alpha)^{k-2}$, where $\alpha = k \cdot \gamma \cdot \zeta$, and k is the number of threads, γ is the lock-nesting depth, and ζ is the lock-dependence factor of t.*

4 The General Case

In this section we address the general case of dynamic data-race prediction. The section is organized in two parts, which present the formal details of Theorem 2.2 and Theorem 2.3.

4.1 Upper Bound

In this section we establish Theorem 2.2. Recall that, by Lemma 3.1, the problem is reducible to detecting a realizable rf-poset in the candidate ideal set of the two events that are tested for a data-race. Rf-poset realizability is known to be NP-complete [16], and solvable in polynomial time when the number of threads is bounded [2]. Here we establish more precise upper-bounds, based on the number of threads. In particular, we show the following.

Lemma 4.1. *Rf-poset realizability can be solved in $O(k \cdot n^k)$ time for an rf-poset of size n and k threads.*

Frontiers and extensions. Let $\mathcal{P} = (X, P, \mathrm{RF})$ be an rf-poset, and consider an ideal Y of $\mathcal{P}$. The *frontier* of Y, denoted $\mathsf{Frontier}_{\mathcal{P}}(Y)$, is the set of pairs $(\mathsf{w}, \mathsf{r}) \in \mathsf{Pairs}(\mathcal{P})$ such that $\mathsf{w} \in Y$ and $\mathsf{r} \notin Y$. An event e executable in Y is said to *extend* Y if for every triplet $(\mathsf{w}, \mathsf{r}, e) \in \mathsf{Triplets}(\mathcal{P})$, we have $(\mathsf{w}, \mathsf{r}) \notin \mathsf{Frontier}_{\mathcal{P}}(Y)$. In this case, we say that $Y \cup \{e\}$ is an *extension* of Y via e.

Ideal graphs and canonical traces. Let $\mathcal{P} = (X, P, \mathrm{RF})$ be an rf-poset. The *ideal graph* of $\mathcal{P}$, denoted $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ is a directed graph defined as follows.

1. $V_{\mathcal{P}}$ is the set of ideals of $\mathcal{P}$.
2. We have $(Y_1, Y_2) \in E_{\mathcal{P}}$ iff Y_2 is an extension of Y_1.

The *ideal tree* of $\mathcal{P}$, denoted $T_{\mathcal{P}} = (\mathcal{I}_{\mathcal{P}}, \mathcal{R}_{\mathcal{P}})$ is a (arbitrary) spanning tree of $G_{\mathcal{P}}$ when restricted to nodes reachable from $\emptyset$. We let $\emptyset$ be the root of $T_{\mathcal{P}}$. Given an ideal $Y \in \mathcal{I}_{\mathcal{P}}$, we define the *canonical trace* t_Y of Y inductively, as follows. If $Y = \emptyset$ then $t_Y = \epsilon$. Otherwise, Y has a parent Y' in $T_{\mathcal{P}}$ such that $Y = Y' \cup \{e\}$ for some event $e \in X$. We define $t_Y = t_{Y'} \circ e$. Lemma 4.1 relies on the following lemmas.

Lemma 4.2. *We have $X \in \mathcal{I}_{\mathcal{P}}$ iff $\mathcal{P}$ is realizable.*

Lemma 4.3. *The ideal graph $G_{\mathcal{P}}$ has $O(n^k)$ nodes.*

Proof of Theorem 2.2. Consider a trace t and two conflicting events $e_1, e_2 \in \mathcal{WR}(t)$. By Lemma 3.1, to decide whether (e_1, e_2) is a predictable data race of t, it suffices to iterate over all feasible trace ideals X in the candidate ideal set $\mathsf{CIS}_t(e_1, e_2)$, and test whether X is realizable. By Lemma 3.2, we have $|\mathsf{CIS}_t(e_1, e_2)| = O(\alpha)$, where $\alpha = \min(n, k \cdot \gamma \cdot \zeta)^{k-2}$. Finally, due to Lemma 4.1, the realizability of every such ideal can be performed in $O(k \cdot n^k) = O(\beta)$ time. □

4.2 Hardness of Data Race Prediction

Here we establish that the problem of dynamic data-race prediction is W[1]-hard when parameterized by the number of threads k. Our proof is established in two steps. In the first step, we show the following lemma.

Lemma 4.4. *Rf-poset realizability parameterized by the number of threads k is W[1]-hard.*

Rf-poset realizability is known to be NP-hard [16, Theorem 4.1], and Lemma 4.4 strengthens that result by showing that the problem is even unlikely to be FPT. In the second step, we show how the class of W[1]-hard instances of in Lemma 4.4 can be reduced to dynamic data-race prediction.

Hardness of rf-poset realizability. Our reduction is from the INDEPENDENT-SET(c) problem, which takes as input an undirected graph $G = (V, E)$ and asks whether G has an independent set of size c. INDEPENDENT-SET(c) parameterized by c is one of the canonical W[1]-hard problems [12].

Given an input $G = (V, E)$ of INDEPENDENT-SET(c) with $n = |V|$, we construct an rf-poset $\mathcal{P}_G = (X, P, \text{RF})$ of size $O(c \cdot n)$ and $O(c)$ threads such that $\mathcal{P}_G$ is realizable iff G has an independent set of size c. We assume wlog that every node in G has at least one neighbor, otherwise, we can remove all s such nodes and solve the problem for parameter $c' = c - s$. The rf-poset $\mathcal{P}_G$ consists of $k = 2 \cdot c + 2$ total orders (X_i, τ_i). Figure 1 provides an illustration. In high level, for each $i \in [c]$, τ_i and τ_{c+i} are used to encode the i-th copy of G, whereas the last two total orders are auxiliary. Superscripts on the events and/or their variables refer to the node of G that is encoded by those events. Below we describe the events and certain orderings between them. The partial order P is the transitive closure of these orderings.

1. For $i = 2 \cdot c + 1$, τ_i consists of a single event $\tau_i = \mathsf{w}(x)$.
2. For $i = 2 \cdot c + 2$, we have $\tau_i = \sigma \circ \mathfrak{o}$, where
$$\sigma = \mathsf{r}(s_1), \ldots, \mathsf{r}(s_c), \mathsf{acq}(\ell_1), \ldots, \mathsf{acq}(\ell_c) \qquad \text{and}$$
$$\mathfrak{o} = \mathsf{r}(x), \mathsf{rel}(\ell_c), \ldots, \mathsf{rel}(\ell_1) \,.$$
3. For each $i \in [c]$, we have $\tau_i = \tau_i^1 \circ \tau_i^2 \circ \ldots \circ \tau_i^n$, where each τ_i^j encodes node j of G and is defined as follows. Let $\overline{\tau}_i^j = \sigma_i^j \circ \mathfrak{o}_i^j$, where
$$\sigma_i^j = \mathsf{acq}_i(\ell_{\{j,l_1\}}), \ldots, \mathsf{acq}_i(\ell_{\{j,l_m\}}) \qquad \text{and}$$
$$\mathfrak{o}_i^j = \mathsf{rel}_i(\ell_{\{j,l_m\}}), \ldots, \mathsf{rel}_i(\ell_{\{j,l_1\}})$$
where $l_1, \ldots, l_m$ are the neighbors of j in G. For each $j \in [n] \setminus \{1, n\}$, the sequence τ_i^j is identical to $\overline{\tau}_i^j$, with the addition that the innermost critical section (i.e., between $\mathsf{acq}_i(\ell_{\{j,l_m\}})$ and $\mathsf{rel}_i(\ell_{\{j,l_m\}})$) contains the sequence $\mathsf{w}(y_i^j), \mathsf{r}(z_i^j)$. The sequence τ_i^1 is defined similarly, except that the innermost critical section contains the sequence $\mathsf{w}(s_i), \mathsf{r}(z_i^1)$. Finally, the sequence τ_i^n is defined similarly, except that the innermost critical section contains the sequence $\mathsf{w}(y_i^n), \mathsf{r}_i(x)$.
4. For each $i \in [c]$, we have $\tau_{c+i} = \tau_{c+i}^1 \circ \tau_{c+i}^2 \circ \ldots \circ \tau_{c+i}^{n-1}$, where $\tau_{c+i}^j = \mathsf{acq}^j(\ell_i), \mathsf{w}(z_i^j), \mathsf{r}(y_i^{j+1}), \mathsf{rel}^j(\ell_i)$.

Note that every memory location is written exactly once, hence the reads-from function RF is defined implicitly. In addition, for every read event r, we have $\text{RF}(\mathsf{r}) <_P \mathsf{r}$, as well as $\mathsf{r}(x) <_P \mathsf{r}_i(x)$ for each $i \in [c]$.

Correctness. We now sketch the correctness of the construction.Assume that $\mathcal{P}$ is realizable by a witness t. We say that $\mathsf{r}(x)$ *separates* a critical section in t if the lock-acquire (resp., lock-release) event of that critical section appears before (resp., after) $\mathsf{r}(x)$ in t. The construction guarantees that, for each $i \in [c]$, $\mathsf{r}(x)$ separates the critical sections of τ_i that encode some node l_i of G. By construction, these critical sections are on locks $\ell_{\{l_i,v\}}$, where v ranges over the neighbors of l_i in G. Hence, for any $i' \neq i$, the node $l_{i'}$ cannot be a neighbor of l_i, as this would imply that both critical sections on lock $\ell_{\{l_i,l_{i'}\}}$ are opened before $\mathsf{r}(x)$ and closed after $\mathsf{r}(x)$ in t, which clearly violates lock semantics. Thus, an independent set $A = \{l_1, \ldots, l_c\}$ of G is formed by taking each l_i to be the node of G, the critical sections of which belong to thread τ_i and are separated by $\mathsf{r}(x)$ in t. On the other hand, if G has an independent set $A = \{l_1, \ldots, l_c\}$, a witness t that realizes $\mathcal{P}$ can be constructed by separating the critical sections of the node l_i in τ_i, for each $i \in [c]$.

Hardness of dynamic data-race prediction. Finally, we turn our attention to the hardness of dynamic data-race prediction. Consider the INDEPENDENT-SET(c) problem on a graph G and the associated rf-poset $\mathcal{P}_G = (X, P, \text{RF})$ defined above. We construct a trace t with $\mathcal{E}(t) = X$ and $\text{RF}_t = \text{RF}$ such that $(\mathsf{w}(x), \mathsf{r}(x))$ is a predictable data-race of t iff $\mathcal{P}_G$ is realizable. In particular, t consists of $2 \cdot c + 2$ threads p_i, one for each total order τ_i of $\mathcal{P}_G$. We obtain t as
$$t = \tau_{2 \cdot c+1} \circ t_1 \circ \ldots \circ t_c \circ \tau_{2 \cdot c+2} \,,$$
where each t_i is an appropriate interleaving of the total orders τ_i and τ_{c+i} that respects the reads-from function RF. We conclude the proof of Theorem 2.3 by observing that G has an independent set of size c iff $(\mathsf{w}(x), \mathsf{r}(x))$ is a predictable data-race of t.

Remark 3. *It is known that* INDEPENDENT-SET(c) *cannot be solved in $f(c) \cdot n^{o(c)}$ time under ETH [11]. As our reduction to rf-poset realizability and dynamic data-race prediction uses $k = O(c)$ threads, each of these problems does not have a $f(k) \cdot n^{o(k)}$-time algorithm under ETH.*

5 Tree Communication Topologies

In this section we focus on the case where the input trace t constitutes a tree communication topology. The section is organized in two parts, which present the formal details of Theorem 2.4 and Theorem 2.5.

5.1 An Efficient Algorithm for Tree Topologies

In this section we present the formal details of Theorem 2.4. Recall that the Theorem 2.2 states an $O(\alpha \cdot \beta)$ upper-bound for dynamic data-race prediction, where β is the complexity of deciding rf-poset realizability, and α is an upper-bound on the number of candidate ideals whose realizability we need to check. For tree communication topologies, we obtain Theorem 2.4: (i) we show an improved upper-bound β on the complexity of the realizability of trace ideals over tree topologies, and (ii) we show that it suffices to check the realizability of a single trace ideal (i.e., $\alpha = 1$). We start with point (i), and then proceed with (ii).

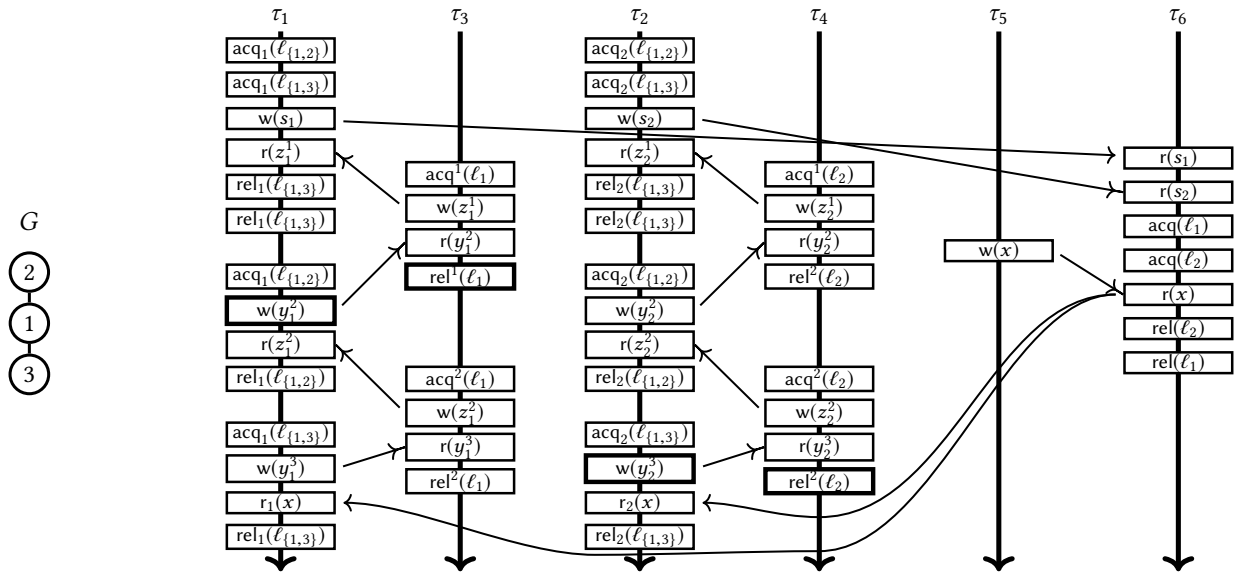

Figure 1. Illustration of the o-poset $\mathcal{P}_G$ given a graph G and independent-set size $c = 2$. Edges represent orderings in P.

Tree-inducible rf-posets. Let (X, P) be a poset where $X \subseteq \mathcal{E}(t)$. We call (X, P) *tree-inducible* if X can be partitioned into k sets $\{X_i\}_{1 \le i \le k}$ such that the following conditions hold.

1. The graph $T = ([k], \{(i, j) \mid X_i \bowtie X_j\})$ is a tree.
2. $P|X_i$ is a total order for each $i \in [k]$.
3. For every node $\ell \in [k]$ such that ℓ is an internal node in T and for every two connected components C_1, C_2 of T that are created after removing ℓ from T, we have the following. Consider two nodes $i \in C_1$ and $j \in C_2$ and two events $e_1 \in X_i$ and $e_2 \in X_j$ such that $e_1 <_P e_2$, there exists some event $e \in X_\ell$ such that $e_1 <_P e <_P e_2$.

We call an rf-poset (X, P, RF) tree-inducible if (X, P) is tree-inducible. The insight is that traces from tree communication topologies yield tree-inducible trace ideals. Our motivation behind tree inducibility comes from the following lemma.

Lemma 5.1. *Rf-poset realizability of tree-inducible rf-posets can be solved in $O(k^2 \cdot d \cdot n^2 \cdot \log n)$ time, for an rf-poset of size n, k threads and d variables.*

The proof of Lemma 5.1 is in two steps. Recall the definition of closed rf-posets from Section 3.1. First, we show that a tree-inducible, closed rf-poset is realizable (Lemma 5.2). Second, we show that the closure of a tree-inducible rf-poset is also tree-inducible (Lemma 5.3).

Lemma 5.2. *Every closed, tree-inducible rf-poset is realizable.*

Indeed, consider a tree-inducible, closed rf-poset $\mathcal{P} = (X, P, \text{RF})$. The witness t realizing $\mathcal{P}$ is obtained in two steps.

1. We construct a poset (X, Q) with $Q \sqsubseteq P$ as follows. Initially, we let Q be identical to P. Let $\mathcal{P}$ be tree-inducible to a tree $T = ([k], \{(i, j) \mid X_i \bowtie X_j\})$. We traverse T top-down, and for every node i and child j of i, for every two events $e_1 \in X_i$ and $e_2 \in X_j$ with $e_1 \bowtie e_2$ and $e_2 \not<_P e_1$, we order $e_1 <_Q e_2$. Finally, we transitively close Q.
2. We construct t by linearizing (X, Q) arbitrarily.

The next lemma shows that tree-inducibility is preserved under taking closures (if the closure exists).

Lemma 5.3. *Consider an rf-poset $\mathcal{P} = (X, P, \text{RF})$ and let $\mathcal{Q} = (X, Q, \text{RF})$ be the closure of $\mathcal{P}$. If $\mathcal{P}$ is tree-inducible then $\mathcal{Q}$ is also tree-inducible.*

Since, by Remark 2, an rf-poset is realizable only if its closure exists and is realizable, Lemma 5.2 and Lemma 5.3 allow to decide the realizability of an rf-poset by computing its closure. The complexity of the algorithm comes from the complexity of deciding whether the closure exists.

Recall that Lemma 3.2 provides an upper-bound on the number of trace ideals of t that we need to examine for realizability in order to decide data-race prediction. We now proceed with point (ii) towards Theorem 2.4, i.e., we show that for tree communication topologies, a single ideal suffices. Our proof is based on the notion of lock causal cones below.

Lock causal cones. Consider a trace t that defines a tree communication topology $G_t = (V_t, E_t)$. Given an event $e \in \mathcal{E}(t)$ the *lock causal cone* $\mathsf{LCone}_t(e)$ of e is the set X defined by the following process. Consider that G_t is rooted in $\mathrm{p}(e)$.

1. Initially X contains all predecessors of e in $(\mathcal{E}(t), \text{TO})$. We perform a top-down traversal of G_t, and consider a current thread p_1 visited by the traversal.
2. Let p_2 be the parent of p_1 in the traversal tree, and e_2 be the unique maximal event in $(X|p_2, \text{TO})$, i.e., e_2 is the last event of thread p_2 that appears in X. We insert in X all events $e_1 \in \mathcal{E}(t)|p_1$ such that $e_1 <_{\text{TRF}} e_2$.

3. While there exists some lock-acquire event $\mathrm{acq}_1 \in X|_{p_1}$ and there exists another lock-acquire event $\mathrm{acq}_2 \in \mathrm{OpenAcqs}(X)$ with $\mathrm{acq}_1 \bowtie \mathrm{acq}_2$ and $\mathrm{p}(\mathrm{acq}_2) = p_2$, we insert in X all predecessors of rel_1 in $(\mathcal{E}(t), \mathrm{TO})$ (including rel_1), where $\mathrm{rel}_1 = \mathrm{match}_t(\mathrm{acq}_1)$.

Observe that, by construction, $\mathrm{LCone}_t(e)$ is a lock-feasible trace ideal of t. In addition, for any two events $e_1, e_2 \in \mathcal{E}(t)$, the set $\mathrm{LCone}_t(e_1) \cup \mathrm{LCone}_t(e_2)$ is an ideal of t, though not necessarily lock-feasible. Our motivation behind lock causal cones comes from the following lemma. Intuitively, we can decide a predictable data-race by deciding the realizability of the ideal that is the union of the two lock-causal cones.

Lemma 5.4. *Let $X = \mathrm{LCone}_t(e_1) \cup \mathrm{LCone}_t(e_2)$. We have that (e_1, e_2) is a predictable data race of t iff (i) $\{e_1, e_2\} \cap X = \emptyset$, and (ii) X is a realizable trace ideal of t.*

The ($\Leftarrow$) direction of the lemma is straightforward. We refer to [23] for the ($\Rightarrow$) direction. Finally, Theorem 2.4 follows immediately from Lemma 5.1 and Lemma 5.4.

Proof of Theorem 2.4. By Lemma 5.4, we have that (e_1, e_2) is a predictable data race of t iff $\{e_1, e_2\} \cap X = \emptyset$ and X is realizable. By Lemma 5.1, deciding the realizability of X is done in $O(k^2 \cdot d \cdot n^2 \cdot \log n)$ time. The desired result follows. □

5.2 A Lower Bound for Two Threads

In this section we prove a conditional quadratic lower bound for dynamic data-race prediction for two threads. Our proof is via a reduction from the Orthogonal Vectors problem. To make it conceptually simpler, we present our reduction in two steps. First, we show a fine-grained reduction from Orthogonal Vectors to the realizability of an rf-poset with 2 threads and 7 variables. Afterwards, we show how the realizability problem for the rf-posets of the first step can be reduced to the decision problem of dynamic data race prediction with 2 threads, 9 variables and 1 lock.

The Orthogonal Vectors problem (OV). An instance of Orthogonal Vectors consists of two sets A, B, where each set contains $n/2$ binary vectors in D dimensions. The task is to determine whether there exists a pair of vectors $(a, b) \in A \times B$ that is orthogonal, i.e., for all $i \in [D]$ we have $a[i] \cdot b[i] = 0$. There exist algorithms that solve the problem in $O(n^2 \cdot D)$ and $O(2^D \cdot n)$ time, simply by computing the inner product of each pair $(a, b) \in A \times B$ and following a classic Four-Russians technique, respectively. It is conjectured that there is no truly sub-quadratic algorithm for OV [8].

Conjecture 5.5 (Orthogonal Vectors)**.** *There is no algorithm for OV that operates in $O(n^{2-\epsilon} \cdot D^{O(1)})$ time, for any $\epsilon > 0$.*

It is also known that SETH implies the OV conjecture [43]. We first relate OV with rf-poset realizability.

Lemma 5.6. *Rf-poset realizability for an rf-poset with 2 threads and 7 variables has no $O(n^{2-\epsilon})$-time algorithm for any $\epsilon > 0$, under the Orthogonal Vectors conjecture.*

Reduction from OV to rf-poset realizability. For a fine-grained reduction from OV to rf-poset realizability, consider an OV instance (A, B), where $A = (a_j)_{1 \le j \le n/2}$, $B = (b_l)_{1 \le l \le n/2}$, and each $a_j, b_l \in \{0, 1\}^D$. We will construct an rf-poset $\mathcal{P} = (X, P, \mathrm{RF})$ with 2 threads and 7 variables such that the closure of $\mathcal{P}$ exists iff there exists a pair of orthogonal vectors $(a, b) \in A \times B$. Since 2 threads define a tree-inducible rf-poset, Remark 2 and Lemma 5.2 imply that $\mathcal{P}$ is realizable iff OV has a positive answer. The set X consists of two disjoint sets X_A, X_B, so that each is totally ordered in P. For ease of presentation, we denote by τ_A, τ_B the linear orders $(X_A, P|_{X_A})$ and $(X_B, P|_{X_B})$, respectively. To develop some insight, we start with a high-level view of the construction, and then proceed with the details.

Overview of the construction. The linear orders τ_A and τ_B encode the vectors of A and B, respectively. Each of τ_A and τ_B consists of $n/2$ segments, so that the i-th (resp. $(n/2-i+1)$-th) segment of τ_A (resp. τ_B) encodes the contents of the i-th vector of A (resp., B). The two total orders are constructed with a closure computation in mind, which inserts event orderings in P one-by-one. In high-level, an ordering $e_1 < e_2$ encodes the test of whether the bits in a specific coordinate i of two vectors $a_j \in A$ and $b_l \in B$ have product 0. If yes, and moreover, $i < D$, then the closure conditions enforce a new ordering $e_1' < e_2'$, which encodes the test of the bits in coordinate $i + 1$. Otherwise $i = D$, and the closure has been computed and an orthogonal pair has been found. On the other hand, if the bits in coordinate i have product 1, the two current vectors are not orthogonal, and the closure conditions enforce a new ordering $e_1'' < e_2''$, which encodes the test of the first coordinate of the next pair of vectors. The above is achieved using 7 variables $\{x_i\}_{i \in [7]}$.

Formal construction. We now present the formal details of the construction (illustrated in Figure 2). The construction creates various events which have the form e^{a_j} and e^{b_l} when they are used at the vector level, and have the form $e_i^{a_j}$ and $e_i^{b_l}$, where $i \in [D]$, when they are used at the coordinate level. As a general rule, for each $j, l \in [n/2 - 1]$, we have $e^{a_j} <_{\tau_A} e^{a_{j+1}}$ and $e^{b_{l+1}} <_{\tau_B} e^{b_l}$, both for events at the vector and at the coordinate level. At the coordinate level, we also have $e_{i+1}^{a_j} <_{\tau_A} e_i^{a_j}$ and $e_{i+1}^{b_l} <_{\tau_B} e_i^{b_l}$. For succinctness, we often write $e_1, e_2 < e_3$ to denote $e_1 < e_3$ and $e_2 < e_3$. We next describe the events and orderings between them. The partial order P is the transitive closure of these orderings.

Events on x_1 and x_2. For every vector $a_j \in A$ and coordinate $i \in [D]$, we create three events $\mathrm{w}_i^{a_j}(x_1)$, $\mathrm{w}_i^{a_j}(x_2)$ and $\mathrm{r}_i^{a_j}(x_2)$. We make $\mathrm{RF}(\mathrm{r}_i^{a_j}(x_2)) = \mathrm{w}_i^{a_j}(x_2)$, and order

$$\mathrm{w}_i^{a_j}(x_1), \mathrm{w}_i^{a_j}(x_2) <_{\tau_A} \mathrm{r}_i^{a_j}(x_2)\ . \tag{1}$$

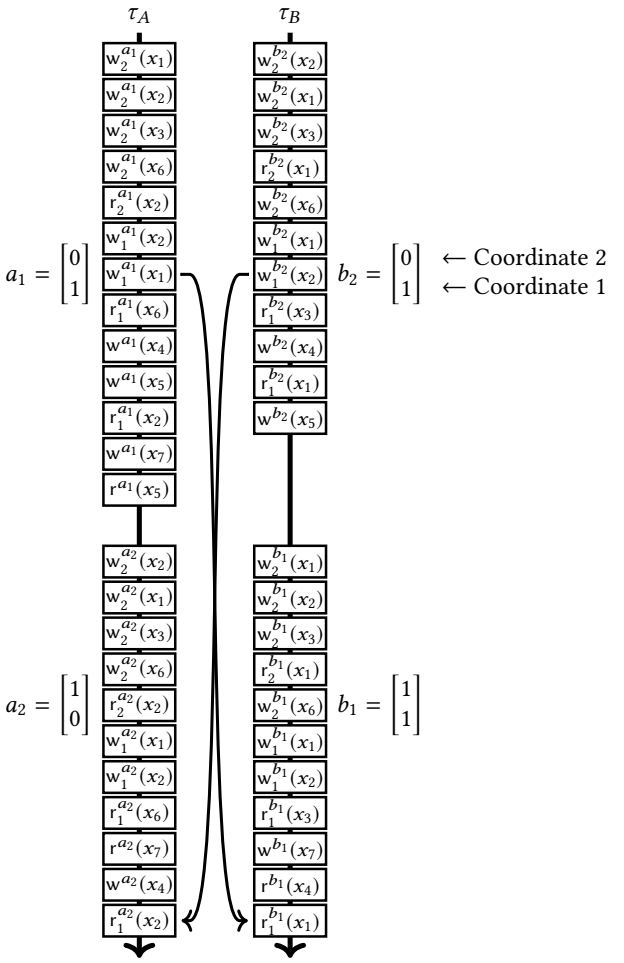

Figure 2. Illustration of the reduction of an OV instance $(A = \{a_1, a_2\}, B = \{b_1, b_2\})$ to the Closure problem of an $\mathcal{P}$.

For every vector $b_l \in B$ and coordinate $i \in [D]$, we create three events $\mathrm{w}_i^{b_l}(x_1)$, $\mathrm{r}_i^{b_l}(x_1)$ and $\mathrm{w}_i^{b_l}(x_2)$. We make $\mathrm{RF}(\mathrm{r}_i^{b_l}(x_1)) = \mathrm{w}_i^{b_l}(x_1)$, and order

$$\mathrm{w}_i^{b_l}(x_1), \mathrm{w}_i^{b_l}(x_2) <_{\tau_B} \mathrm{r}_i^{b_l}(x_1) \,. \tag{2}$$

In addition, we order

$$\begin{aligned} \mathrm{w}_i^{a_j}(x_2) <_{\tau_A} \mathrm{w}_i^{a_j}(x_1) \quad \text{iff} \quad a_j[i] = 1 \qquad \text{and} \\ \mathrm{w}_i^{b_l}(x_1) <_{\tau_B} \mathrm{w}_i^{b_l}(x_2) \quad \text{iff} \quad b_l[i] = 1 \,. \end{aligned} \tag{3}$$

Observe that if $a_j[i] \cdot b_l[i] = 1$, and if we order $\mathrm{w}_i^{a_j}(x_1) < \mathrm{w}_i^{b_l}(x_1)$ then transitively $\mathrm{w}_i^{a_j}(x_2) < \mathrm{w}_i^{b_l}(x_2)$ and hence by closure $\mathrm{r}_i^{a_j}(x_2) < \mathrm{w}_i^{b_l}(x_2)$.

Events on x_3. Let $i \in [D-1]$ be a coordinate. For every vector $a_j \in A$, we create an event $\mathrm{w}_{i+1}^{a_j}(x_3)$, and order

$$\mathrm{w}_{i+1}^{a_j}(x_1), \mathrm{w}_{i+1}^{a_j}(x_2) <_{\tau_A} \mathrm{w}_{i+1}^{a_j}(x_3) <_{\tau_A} \mathrm{w}_i^{a_j}(x_1), \mathrm{w}_i^{a_j}(x_2) \,. \tag{4}$$

For every vector $b_l \in B$, we create two events $\mathrm{w}_{i+1}^{b_l}(x_3)$ and $\mathrm{r}_i^{b_l}(x_3)$, and make $\mathrm{RF}(\mathrm{r}_i^{b_l}(x_3)) = \mathrm{w}_{i+1}^{b_l}(x_3)$. We order

$$\begin{aligned} \mathrm{w}_{i+1}^{b_l}(x_1), \mathrm{w}_{i+1}^{b_l}(x_2) <_{\tau_B} \mathrm{w}_{i+1}^{b_l}(x_3) <_{\tau_B} \mathrm{r}_{i+1}^{b_l}(x_1) \quad \text{and} \\ \mathrm{w}_i^{b_l}(x_1), \mathrm{w}_i^{b_l}(x_2) <_{\tau_B} \mathrm{r}_i^{b_l}(x_3) <_{\tau_B} \mathrm{r}_i^{b_l}(x_1) \,. \end{aligned} \tag{5}$$

Observe that if we order $\mathrm{w}_i^{a_j}(x_1) < \mathrm{w}_i^{b_l}(x_1)$ then we also have $\mathrm{w}_{i+1}^{a_j}(x_3) < \mathrm{r}_i^{b_l}(x_3)$, hence by closure $\mathrm{w}_{i+1}^{a_j}(x_3) < \mathrm{w}_{i+1}^{b_l}(x_3)$ and thus $\mathrm{w}_{i+1}^{a_j}(x_1) < \mathrm{r}_i^{b_l}(x_1)$.

Events on x_6. For every coordinate $i \in [D-1]$, we do as follows. For every vector $a_j \in A$, we create two events $\mathrm{w}_{i+1}^{a_j}(x_6)$ and $\mathrm{r}_i^{a_j}(x_6)$, and make $\mathrm{RF}(\mathrm{r}_i^{a_j}(x_6)) = \mathrm{w}_{i+1}^{a_j}(x_6)$. We order

$$\begin{aligned} \mathrm{w}_{i+1}^{a_j}(x_3) <_{\tau_A} \mathrm{w}_{i+1}^{a_j}(x_6) <_{\tau_A} \mathrm{r}_{i+1}^{a_j}(x_2) \quad \text{and} \\ \mathrm{w}_i^{a_j}(x_1), \mathrm{w}_i^{a_j}(x_2) <_{\tau_A} \mathrm{r}_i^{a_j}(x_6) <_{\tau_A} \mathrm{r}_i^{a_j}(x_2) \,. \end{aligned} \tag{6}$$

For every vector $b_l \in B$, we create one event $\mathrm{w}_{i+1}^{b_l}(x_6)$, and order

$$\mathrm{r}_{i+1}^{b_l}(x_1) <_{\tau_B} \mathrm{w}_{i+1}^{b_l}(x_6) <_{\tau_B} \mathrm{w}_i^{b_l}(x_1), \mathrm{w}_i^{b_l}(x_2) \,. \tag{7}$$

Observe that if we order $\mathrm{r}_{i+1}^{a_j}(x_2) < \mathrm{w}_{i+1}^{b_l}(x_2)$, since $\mathrm{w}_{i+1}^{b_l}(x_2) <_{\tau_B} \mathrm{r}_{i+1}^{b_l}(x_1)$, we also have $\mathrm{w}_{i+1}^{a_j}(x_6) < \mathrm{w}_{i+1}^{b_l}(x_6)$, hence by closure $\mathrm{r}_i^{a_j}(x_6) < \mathrm{w}_{i+1}^{b_l}(x_6)$ and thus $\mathrm{w}_i^{a_j}(x_2) < \mathrm{w}_i^{b_l}(x_2)$.

Events on x_4. For every vector $a_j \in A$, we create one event $\mathrm{w}^{a_j}(x_4)$, and order

$$\mathrm{r}_1^{a_j}(x_6) <_{\tau_A} \mathrm{w}^{a_j}(x_4) <_{\tau_A} \mathrm{r}_1^{a_j}(x_2) \,. \tag{8}$$

For every vector $b_l \in B$, with $l \in [n/2-1]$, we create two events $\mathrm{w}^{b_{l+1}}(x_4)$ and $\mathrm{r}^{b_l}(x_4)$, and make $\mathrm{RF}(\mathrm{r}^{b_l}(x_4)) = \mathrm{w}^{b_{l+1}}(x_4)$. We order

$$\begin{aligned} \mathrm{r}_1^{b_l}(x_3) <_{\tau_B} \mathrm{r}^{b_l}(x_4) <_{\tau_B} \mathrm{r}_1^{b_l}(x_1) \quad \text{and} \\ \mathrm{r}_1^{b_{l+1}}(x_3) <_{\tau_B} \mathrm{w}^{b_{l+1}}(x_4) <_{\tau_B} \mathrm{r}_1^{b_{l+1}}(x_1) \,. \end{aligned} \tag{9}$$

Observe that if we order $\mathrm{r}_1^{a_j}(x_2) < \mathrm{w}_1^{b_l}(x_2)$ then we also have $\mathrm{w}^{a_j}(x_4) < \mathrm{r}^{b_l}(x_4)$ (since $\mathrm{w}_1^{b_l}(x_2) <_{\tau_B} \mathrm{r}_1^{b_l}(x_3)$ by Eq. (5)) and thus by closure $\mathrm{w}^{a_j}(x_4) < \mathrm{w}^{b_{l+1}}(x_4)$.

Events on x_5 and x_7. For every vector $a_j \in A$, with $j \in [n/2-1]$, we create two events $\mathrm{w}^{a_j}(x_5)$ and $\mathrm{r}^{a_j}(x_5)$, and make $\mathrm{RF}(\mathrm{r}^{a_j}(x_5)) = \mathrm{w}^{a_j}(x_5)$. We also create two events $\mathrm{w}^{a_j}(x_7)$ and $\mathrm{r}^{a_{j+1}}(x_7)$, and make $\mathrm{RF}(\mathrm{r}^{a_{j+1}}(x_7)) = \mathrm{w}^{a_j}(x_7)$. We order

$$\begin{aligned} &\mathrm{w}^{a_j}(x_4) <_{\tau_A} \mathrm{w}^{a_j}(x_5) <_{\tau_A} \mathrm{r}_1^{a_j}(x_2) <_{\tau_A} \mathrm{w}^{a_j}(x_7) <_{\tau_A} \mathrm{r}^{a_j}(x_5) \text{ and} \\ &\mathrm{r}_1^{a_{j+1}}(x_6) <_{\tau_A} \mathrm{r}^{a_{j+1}}(x_7) <_{\tau_A} \mathrm{w}^{a_{j+1}}(x_4) \,. \end{aligned} \tag{10}$$

We also create two events $\mathrm{w}^{b_{n/2}}(x_5)$ and $\mathrm{w}^{b_1}(x_7)$, and order

$$\begin{aligned} \mathrm{r}_1^{b_{n/2}}(x_1) <_{\tau_B} \mathrm{w}^{b_{n/2}}(x_5) <_{\tau_B} \mathrm{w}_D^{b_{n/2}-1}(x_1), \mathrm{w}_D^{b_{n/2}-1}(x_2) \quad \text{and} \\ \mathrm{r}_1^{b_1}(x_3) <_{\tau_B} \mathrm{w}^{b_1}(x_7) <_{\tau_B} \mathrm{r}^{b_1}(x_4) \,. \end{aligned} \tag{11}$$

Observe that if we order $\mathrm{r}_1^{a_j}(x_2) < \mathrm{w}_1^{b_{n/2}}(x_2)$ then we also have $\mathrm{w}^{a_j}(x_5) < \mathrm{w}^{b_{n/2}}(x_5)$ (since $\mathrm{w}_1^{b_{n/2}}(x_2) <_{\tau_B} \mathrm{r}_1^{b_{n/2}}(x_1)$ by a

previous item) and thus by closure $\mathrm{r}^{a_j}(x_5) < \mathrm{w}^{b_{n/2}}(x_5)$. But then also $\mathrm{w}^{a_j}(x_7) < \mathrm{w}^{b_1}(x_7)$ (since $\mathrm{w}^{b_{n/2}}(x_5) <_{\tau_B} \mathrm{w}^{b_1}(x_7)$) and thus by closure $\mathrm{r}^{a_{j+1}}(x_7) < \mathrm{w}^{b_1}(x_7)$.

Final orderings. Finally, we order

$$\mathrm{w}^{b_{n/2}}(x_5) <_{\tau_B} \mathrm{w}_D^{b_{n/2-1}}(x_1), \mathrm{w}_D^{b_{n/2-1}}(x_2) \,. \tag{12}$$

For each $j \in [n/2-1]$, we order

$$\mathrm{r}^{a_j}(x_5) <_{\tau_A} \mathrm{w}_D^{a_{j+1}}(x_1), \mathrm{w}_D^{a_{j+1}}(x_2) \,.$$

For each $j \in [n/2-2]$, we order

$$\mathrm{r}^{b_{j+1}}(x_1) <_{\tau_B} \mathrm{w}_D^{b_j}(x_1), \mathrm{w}_D^{b_j}(x_2) \,. \tag{13}$$

We make two orderings across τ_A and τ_B, namely

$$\mathrm{w}_1^{a_1}(x_1) <_P \mathrm{r}_1^{b_1}(x_1) \quad \text{and} \quad \mathrm{w}_1^{b_{n/2}}(x_2) <_P \mathrm{r}_1^{a_{n/2}}(x_2) \,. \tag{14}$$

Correctness. Observe that we have used 7 variables, while $|X_A| + |X_B| = O(n \cdot D)$, and the reduction can be easily computed in linear time. We refer to [23] for the proof of correctness. This concludes Lemma 5.6, as any algorithm for rf-poset realizability on the above instances that runs in $O((n \cdot D)^{2-\epsilon})$ time also solves OV in $O(n^{2-\epsilon} \cdot D^{O(1)})$ time. Although the full proof is rather technical, the correctness is conceptually straightforward. We illustrate the key idea on the example of Figure 2, where we perform closure operations by inserting new orderings in a partial order $<$.

By construction, we have $\mathrm{w}_1^{a_1}(x_1) < \mathrm{r}_1^{b_1}(x_1)$, which signifies testing the first coordinate of vectors a_1 and b_1. Note that $a_1[1] \cdot b_1[1] = 1$, for which our encoding guarantees that eventually $\mathrm{r}_1^{a_1}(x_2) < \mathrm{w}_1^{b_1}(x_2)$. Indeed, since $\mathrm{w}_1^{a_1}(x_1) < \mathrm{r}_1^{b_1}(x_1)$, by closure we also have $\mathrm{w}_1^{a_1}(x_1) < \mathrm{w}_1^{b_1}(x_1)$. In turn, this leads to $\mathrm{w}_1^{a_1}(x_2) < \mathrm{w}_1^{b_1}(x_2)$, and by closure, we also have $\mathrm{r}_1^{a_1}(x_2) < \mathrm{w}_1^{b_1}(x_2)$. This leads to $\mathrm{w}^{a_1}(x_4) < \mathrm{r}^{b_1}(x_4)$, and by closure, we have $\mathrm{w}^{a_1}(x_4) < \mathrm{w}^{b_2}(x_4)$. This last ordering leads to $\mathrm{w}_1^{a_1}(x_1) < \mathrm{r}_1^{b_2}(x_1)$, which signifies testing the first coordinate of vectors a_1 and b_2, i.e., moving with the next vector of B.

The process for a_1 and b_2 is similar to a_1 and b_1, as the two vectors are found not orthogonal already in the first coordinate. As previously, we eventually arrive at $\mathrm{r}_1^{a_1}(x_2) < \mathrm{w}_1^{b_2}(x_2)$. Note that this leads to $\mathrm{w}^{a_1}(x_5) < \mathrm{w}^{b_2}(x_5)$, and by closure, we have $\mathrm{r}^{a_1}(x_5) < \mathrm{w}^{b_2}(x_5)$. In turn, this leads to $\mathrm{w}^{a_1}(x_7) < \mathrm{w}^{b_1}(x_7)$, and by closure, we have $\mathrm{r}^{a_2}(x_7) < \mathrm{w}^{b_1}(x_7)$. This last ordering leads to $\mathrm{w}_1^{a_2}(x_1) < \mathrm{r}_1^{b_1}(x_1)$, which signifies testing the first coordinate of vectors a_2 and b_1, i.e., moving with the next vector of A and the first vector of B.

The process for a_2 and b_1 is initially different than before, as $a_2[1] \cdot b_1[1] = 0$, i.e., the test on the first coordinate does not deem a_2 and b_1 not orthogonal. By closure, the ordering $\mathrm{w}_1^{a_2}(x_1) < \mathrm{r}_1^{b_1}(x_1)$ leads to $\mathrm{w}_1^{a_2}(x_1) < \mathrm{w}_1^{b_1}(x_1)$. This leads to $\mathrm{w}_2^{a_2}(x_3) < \mathrm{r}_1^{b_1}(x_3)$, and by closure, we have $\mathrm{w}_2^{a_2}(x_3) < \mathrm{w}_2^{b_1}(x_3)$. This leads to $\mathrm{w}_2^{a_2}(x_1) < \mathrm{r}_2^{b_1}(x_1)$, which signifies testing the second coordinate of vectors a_2 and b_1. As $a_2[2] \cdot b_2[2] = 1$, the two vectors are discovered as non-orthogonal, which is captured by an eventual ordering $\mathrm{r}_2^{a_2}(x_2) < \mathrm{w}_2^{b_1}(x_2)$. This ordering which witnesses non-orthogonality is propagated downwards to the first coordinate, i.e., $\mathrm{r}_1^{a_2}(x_2) < \mathrm{w}_1^{b_1}(x_2)$. This propagation is made by events on variable x_6. Indeed, first note that, as $\mathrm{r}_2^{a_2}(x_2) < \mathrm{w}_2^{b_1}(x_2)$, we also have $\mathrm{w}_2^{a_2}(x_6) < \mathrm{w}_2^{b_1}(x_6)$, and by closure, we have $\mathrm{r}_1^{a_2}(x_6) < \mathrm{w}_2^{b_1}(x_6)$. This leads to $\mathrm{w}_1^{a_2}(x_2) < \mathrm{w}_1^{b_1}(x_2)$, and by closure, $\mathrm{r}_1^{a_2}(x_2) < \mathrm{w}_1^{b_1}(x_2)$, which marks the two vectors as non-orthogonal. This leads to $\mathrm{w}^{a_2}(x_4) < \mathrm{r}^{b_1}(x_4)$, and by closure, we have $\mathrm{w}^{a_2}(x_4) < \mathrm{w}^{b_2}(x_4)$. This last ordering leads to $\mathrm{w}_1^{a_2}(x_1) < \mathrm{r}_1^{b_2}(x_1)$, which signifies testing the first coordinate of vectors a_2 and b_2, i.e., moving with the next vector of B.

The process for a_2 and b_2 is initially similar to the previous case, as $a_2[1] \cdot b_2[1] = 0$. However, because we also have $a_2[2] \cdot b_2[2] = 0$, we will *not* order $\mathrm{r}_2^{a_2}(x_2) < \mathrm{w}_2^{b_2}(x_2)$, and the closure will terminate after ordering $\mathrm{w}_2^{a_2}(x_1) < \mathrm{w}_2^{b_2}(x_1)$. Since no cyclical orderings were introduce, the closure of $\mathcal{P}$ exists, and by Lemma 5.2, $\mathcal{P}$ is realizable. Finally, observe that *if* we eventually had $\mathrm{r}_1^{a_2}(x_2) < \mathrm{w}_1^{b_2}(x_2)$ (signifying that a_2 and b_2 are not orthogonal, hence there is no orthogonal pair in $A \times B$), this would create a cycle with the ordering $\mathrm{w}_1^{b_2}(x_2) <_P \mathrm{r}_1^{a_2}(x_2)$, and by Remark 2, $\mathcal{P}$ would *not* be realizable.

Reduction to dynamic data-race prediction. Consider an instance of the rf-poset $\mathcal{P} = (X, P, \mathrm{RF})$ constructed in the above reduction, and we construct a trace t and two events $e_1, e_2 \in \mathcal{E}(t)$ such that $\mathcal{P}$ is realizable iff (e_1, e_2) is a predictable data race of t. The trace t consists of two threads p_A, p_B and two local traces τ'_A and τ'_B such that τ'_A and τ'_B contain the events of p_A and p_B, respectively. Each of τ'_A and τ'_B is identical to τ_A and τ_B of $\mathcal{P}$, respectively, with some additional events inserted in it. In particular, besides the variables x_i, $i \in [7]$ that appear in the events of X, we introduce one variable y and one lock ℓ. For the event set, we have

$$\begin{aligned}\mathcal{E}(t) = &X \cup \{\mathrm{w}(y), \mathrm{r}(y)\} \cup \{\mathrm{acq}_A(\ell), \mathrm{rel}_A(\ell)\} \cup \\ &\{\mathrm{acq}_B(\ell), \mathrm{rel}_B(\ell)\} \cup \{\mathrm{w}(z), \mathrm{r}(z)\} \,.\end{aligned}$$

The local traces τ'_A and τ'_B are constructed as follows.

1. For τ'_A, we insert an empty critical section $\mathrm{acq}_A(\ell), \mathrm{rel}_A(\ell)$ right after $\mathrm{w}_1^{a_1}(x_1)$. Additionally, we insert the read event $\mathrm{r}(y)$ right before $\mathrm{r}_1^{a_{n/2}}(x_2)$, and the event $\mathrm{r}(z)$ as the last event of τ'_A.
2. For τ'_B, we insert the write event $\mathrm{w}(y)$ right after $\mathrm{w}_1^{b_{n/2}}(x_2)$. Additionally, we insert $\mathrm{w}(z)$ right after $\mathrm{r}_1^{b_1}(x_1)$, and surround these two events with $\mathrm{acq}_B(\ell), \mathrm{rel}_B(\ell)$.

Finally, we obtain t as $t = \tau'_B \circ \tau'_A$, i.e., the two local traces are executed sequentially and there is no context switching. The task is to decide whether $(\mathrm{w}(z), \mathrm{r}(z))$ is a predictable data race of t. We refer to [23] for the correctness of the construction, which concludes Theorem 2.5.

6 Witnesses in Small Distance

The results in the previous sections neglect information provided by the input trace t about constructing a correct reordering that witnesses the data race. Indeed, our hardness results show that, in the worst case, the orderings in t provide no help. However, in practice when a data race exists, a witness trace t^* can be constructed that is similar to t. In fact, virtually all practical techniques predict data races by constructing t^* to be very similar to t (e.g., [15, 20, 32, 35, 41]).

The distance-bounded realizability problem of feasible trace ideals. Given a natural number ℓ, a trace t and a feasible trace ideal X of t, the solution to the *ℓ-distance-bounded realizability* problem is False if X is not realizable, True if there is a witness t^* that realizes X such that $\delta(t, t^*) \leq \ell$, and can be any answer (True/False) if X is realizable but any witness t^* that realizes X is such that $\delta(t, t^*) > \ell$. We remark that this formulation is that of a *promise problem* [14]. We are interested in the case where $\ell = O(1)$. There exists a straightforward algorithm that operates in $O(|X|^{2\cdot\ell})$ time. The algorithm iterates over all possible subsets of pairs of conflicting write and lock-acquire events that have size at most ℓ, and tries all possible combinations of conflicting-write reversals in that set. Theorem 2.6 is based on the following lemma, which states that the problem can be solved much faster when k is also constant.

Lemma 6.1. *Consider a natural number ℓ, a trace t over n events and k threads, and a feasible trace ideal X of t. The ℓ-distance-bounded realizability problem for X can be solved in $O(k^{\ell+O(1)} \cdot n)$ time.*

Proof of Theorem 2.6. By Lemma 6.1, given a trace ideal X of t, we can solve the ℓ-distance-bounded realizability problem for X in $O(n)$ time. The proof then follows by Lemma 3.1 and Lemma 3.2, as to decide whether (e_1, e_2) is a predictable data race of t, it suffices to examine $O(1)$ trace ideals of t. □

In the remaining of this section we prove Lemma 6.1. We first define the notion of read extensions of graphs. Afterwards, we present the algorithm for the lemma, and show its correctness and complexity.

Read extensions. Consider a digraph $G = (X, E)$ where X is a set of events. Given two events $e_1, e_2 \in G$, we write $e_1 \rightsquigarrow e_2$ to denote that e_2 is reachable from e_1. We call G *write-ordered* if for every two distinct conflicting write or lock-acquire events $\mathsf{w}_1, \mathsf{w}_2 \in \mathcal{WL}(X)$, we have $\mathsf{w}_1 \rightsquigarrow \mathsf{w}_2$ or $\mathsf{w}_2 \rightsquigarrow \mathsf{w}_1$ in G. Given an acyclic write-ordered graph $G_1 = (X, E_1)$, the *read extension* of G_1 is the digraph $G_2 = (X, E_2)$ where $E_2 = E_1 \cup A \cup B$, where the sets A and B are defined as follows.

$$A = \{(\mathsf{r}, \mathsf{w}) \in \mathcal{RL}(X) \times \mathcal{WL}(X) \mid \mathsf{r} \bowtie \mathsf{w} \text{ and } (\mathrm{RF}_t(\mathsf{r}), \mathsf{w}) \in E_1\} ,$$
$$B = \{(\mathsf{w}, \mathsf{r}) \in \mathcal{WL}(X) \times \mathcal{RL}(X) \mid \mathsf{r} \bowtie \mathsf{w} \text{ and } (\mathsf{w}, \mathrm{RF}_t(\mathsf{r})) \in E_1\} .$$

A fast algorithm for distance-bounded rf-poset realizability. Let $\mathcal{P} = (X, P, \mathrm{RF})$ be the canonical rf-poset of X, and the task is to decide the realizability of $\mathcal{P}$ with ℓ reversals. We describe a recursive algorithm for solving the problem for some rf-poset $\mathcal{Q} = (X, Q, \mathrm{RF})$ with ℓ' reversals, for some $\ell' \leq \ell$, where initially $Q = P$ and $\ell' = \ell$.

Algorithm and correctness. Consider the set

$$C = \{(\mathsf{w}_1, \mathsf{w}_2) \in \mathcal{WL}(X) \times \mathcal{WL}(X) \mid \mathsf{w}_1 \bowtie \mathsf{w}_2 \quad \text{and} \quad \mathsf{w}_1 \parallel_Q \mathsf{w}_2 \text{ and } \mathsf{w}_1 <_t \mathsf{w}_2\} .$$

We construct a graph $G_1 = (X, E_1)$, where $E_1 = (\mathsf{TRF}|X) \cup C$. Note that G_1 is write-ordered. If it is acyclic, we construct the read extension G_2 of G_1. Observe that if G_2 is acyclic then any linearization t^* of G realizes $\mathcal{Q}$, hence we are done. Now consider that either G_1 or G_2 is not acyclic, and let $G = G_1$ if G_1 is not acyclic, otherwise $G = G_2$. Given a cycle $\mathcal{C}$ of G, represented as a collection of edges, define the set of *cross-edges* of $\mathcal{C}$ as $\mathcal{C} \setminus Q$. Note that, since there are k threads, G has a cycle with $\leq k$ cross edges. In addition, any trace t^* that realizes $\mathcal{Q}$ must linearize an rf-poset (X, Q_a, RF) where $a = (e_1, e_2)$ ranges over the cross-edges of $\mathcal{C}$. In particular, we take $Q_a = Q \cup \{b\}$, where

$$b = \begin{cases} (e_2, e_1), & \text{if } a \in \mathcal{WL}(X) \times \mathcal{WL}(X) \\ (\mathrm{RF}(e_2), e_1), & \text{if } a \in \mathcal{WL}(X) \times \mathcal{RL}(X) \\ (e_2, \mathrm{RF}(e_1)), & \text{if } a \in \mathcal{RL}(X) \times \mathcal{WL}(X) . \end{cases}$$

Observe that any such choice of b reverses the order of two conflicting write events or lock-acquire events in t. Since there are $\leq k$ cross edges in $\mathcal{C}$, there are $\leq k$ such choices for Q_a. Repeating the same process recursively for the rf-poset (X, Q_a, RF) for $\ell' - 1$ levels solves the ℓ'-distance-bounded realizability problem for $\mathcal{Q}$. Since initially $\ell' = \ell$ and $Q = P$, this process solves the same problem for $\mathcal{P}$ and thus for X.

Complexity. The recursion tree above has branching $\leq k$ and depth $\leq \ell$, hence there will be at most k^ℓ recursive instances. In [23], we provide some lower-level algorithmic details which show that each instance can be solved in $O(k^{O(1)} \cdot n)$ time. The main idea is that each of the graphs G_1 and G_2 have a sparse transitive reduction [3] of size $O(k \cdot n)$, and thus each graph can be analyzed in $O(k \cdot n)$ time.

Acknowledgments

We thank anonymous reviewers for their constructive feedback on an earlier draft of this manuscript. Remark 3 is due to anonymous reviewer. Umang Mathur is partially supported by a Google PhD Fellowship. Mahesh Viswanathan was partially supported by grant NSF SHF 1901069.

References

[1] I. J. Aalbersberg. 1988. Theory of Traces. *Theor. Comput. Sci.* 60, 1 (Sept. 1988), 1–82. https://doi.org/10.1016/0304-3975(88)90051-5

[2] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, Magnus Lång, Tuan Phong Ngo, and Konstantinos Sagonas. 2019. Optimal Stateless Model Checking for Reads-from Equivalence under Sequential Consistency. *Proc. ACM Program. Lang.* 3, OOPSLA, Article Article 150 (Oct. 2019), 29 pages. https://doi.org/10.1145/3360576

[3] A. V. Aho, M. R. Garey, and J. D. Ullman. 1972. The Transitive Reduction of a Directed Graph. *SIAM J. Comput.* 1, 2 (1972), 131–137. https://doi.org/10.1137/0201008

[4] A. Bertoni, G. Mauri, and N. Sabadini. 1989. Membership Problems for Regular and Context-Free Trace Languages. *Inf. Comput.* 82, 2 (Aug. 1989), 135–150. https://doi.org/10.1016/0890-5401(89)90051-5

[5] Ranadeep Biswas and Constantin Enea. 2019. On the Complexity of Checking Transactional Consistency. *Proc. ACM Program. Lang.* 3, OOPSLA, Article Article 165 (Oct. 2019), 28 pages. https://doi.org/10.1145/3360591

[6] Hans-J. Boehm. 2011. How to Miscompile Programs with "Benign" Data Races. In *Proceedings of the 3rd USENIX Conference on Hot Topic in Parallelism (HotPar'11)*. USENIX Association, USA, 3.

[7] Hans-J. Boehm. 2012. Position Paper: Nondeterminism is Unavoidable, but Data Races Are Pure Evil. In *Proceedings of the 2012 ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability (RACES '12)*. Association for Computing Machinery, New York, NY, USA, 9–14. https://doi.org/10.1145/2414729.2414732

[8] Karl Bringmann. 2019. Fine-Grained Complexity Theory (Tutorial). In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019) (Leibniz International Proceedings in Informatics (LIPIcs))*, Rolf Niedermeier and Christophe Paul (Eds.), Vol. 126. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 4:1–4:7. https://doi.org/10.4230/LIPIcs.STACS.2019.4

[9] Marek Chalupa, Krishnendu Chatterjee, Andreas Pavlogiannis, Kapil Vaidya, and Nishant Sinha. 2018. Data-centric Dynamic Partial Order Reduction. In *Proceedings of the 45rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '18)*.

[10] Feng Chen and Grigore Roşu. 2007. Parametric and Sliced Causality. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07)*. Springer-Verlag, Berlin, Heidelberg, 240–253. http://dl.acm.org/citation.cfm?id=1770351.1770387

[11] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. 2006. Strong computational lower bounds via parameterized complexity. *J. Comput. System Sci.* 72, 8 (2006), 1346 – 1367. https://doi.org/10.1016/j.jcss.2006.04.007

[12] Rodney G. Downey and Michael R. Fellows. 1999. *Parameterized Complexity*. Springer. https://doi.org/10.1007/978-1-4612-0515-9

[13] Michael Emmi and Constantin Enea. 2017. Sound, Complete, and Tractable Linearizability Monitoring for Concurrent Collections. *Proc. ACM Program. Lang.* 2, POPL, Article Article 25 (Dec. 2017), 27 pages. https://doi.org/10.1145/3158113

[14] Shimon Even, Alan L. Selman, and Yacov Yacobi. 1984. The Complexity of Promise Problems with Applications to Public-Key Cryptography. *Inf. Control* 61, 2 (May 1984), 159–173. https://doi.org/10.1016/S0019-9958(84)80056-X

[15] Cormac Flanagan and Stephen N. Freund. 2009. FastTrack: Efficient and Precise Dynamic Race Detection. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '09)*. ACM, New York, NY, USA, 121–133. https://doi.org/10.1145/1542476.1542490

[16] Phillip B. Gibbons and Ephraim Korach. 1997. Testing Shared Memories. *SIAM J. Comput.* 26, 4 (Aug. 1997), 1208–1244. https://doi.org/10.1137/S0097539794279614

[17] Maurice P. Herlihy and Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (July 1990), 463–492. https://doi.org/10.1145/78969.78972

[18] Jeff Huang, Patrick O'Neil Meredith, and Grigore Rosu. 2014. Maximal Sound Predictive Race Detection with Control Flow Abstraction. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*. ACM, New York, NY, USA, 337–348. https://doi.org/10.1145/2594291.2594315

[19] Baris Kasikci, Cristian Zamfir, and George Candea. 2013. RaceMob: Crowdsourced Data Race Detection. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 406–422. https://doi.org/10.1145/2517349.2522736

[20] Dileep Kini, Umang Mathur, and Mahesh Viswanathan. 2017. Dynamic Race Prediction in Linear Time. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*. ACM, New York, NY, USA, 157–170. https://doi.org/10.1145/3062341.3062374

[21] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (July 1978), 558–565. https://doi.org/10.1145/359545.359563

[22] Umang Mathur, Dileep Kini, and Mahesh Viswanathan. 2018. What Happens-after the First Race? Enhancing the Predictive Power of Happens-before Based Dynamic Race Detection. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 145 (Oct. 2018), 29 pages. https://doi.org/10.1145/3276515

[23] Umang Mathur, Andreas Pavlogiannis, and Mahesh Viswanathan. 2020. The Complexity of Dynamic Data Race Prediction. (2020). arXiv:cs.PL/2004.14931

[24] Friedemann Mattern. 1989. Virtual Time and Global States of Distributed Systems. In *Parallel and Distributed Algorithms: proceedings of the International Workshop on Parallel & Distributed Algorithms*, M. Cosnard et. al. (Ed.). Elsevier Science Publishers B. V., 215–226.

[25] A Mazurkiewicz. 1987. Trace Theory. In *Advances in Petri Nets 1986, Part II on Petri Nets: Applications and Relationships to Other Models of Concurrency*. Springer-Verlag New York, Inc., 279–324.

[26] Mayur Naik, Alex Aiken, and John Whaley. 2006. Effective Static Race Detection for Java. In *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '06)*. ACM, New York, NY, USA, 308–319. https://doi.org/10.1145/1133981.1134018

[27] Satish Narayanasamy, Zhenghao Wang, Jordan Tigani, Andrew Edwards, and Brad Calder. 2007. Automatically Classifying Benign and Harmful Data Races Using Replay Analysis. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '07)*. Association for Computing Machinery, New York, NY, USA, 22–31. https://doi.org/10.1145/1250734.1250738

[28] Robert H.B. Netzer and Barton P. Miller. 1990. On the Complexity of Event Ordering for Shared-Memory Parallel Program Executions. In *In Proceedings of the 1990 International Conference on Parallel Processing*. 93–97.

[29] Robert H. B. Netzer and Barton P. Miller. 1992. What Are Race Conditions? Some Issues and Formalizations. *ACM Lett. Program. Lang. Syst.* 1, 1 (March 1992), 74–88. https://doi.org/10.1145/130616.130623

[30] Robert Netzer Netzer and Barton P. Miller. 1989. Detecting Data Races in Parallel Program Executions. In *In Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*. MIT Press, 109–129.

[31] Christos H. Papadimitriou. 1979. The Serializability of Concurrent Database Updates. *J. ACM* 26, 4 (Oct. 1979), 631–653. https://doi.org/10.1145/322154.322158

[32] Andreas Pavlogiannis. 2019. Fast, Sound, and Effectively Complete Dynamic Race Prediction. *Proc. ACM Program. Lang.* 4, POPL, Article Article 17 (Dec. 2019), 29 pages. https://doi.org/10.1145/3371085

[33] Eli Pozniansky and Assaf Schuster. 2003. Efficient On-the-fly Data Race Detection in Multithreaded C++ Programs. *SIGPLAN Not.* 38, 10 (June 2003), 179–190. https://doi.org/10.1145/966049.781529

[34] Polyvios Pratikakis, Jeffrey S. Foster, and Michael Hicks. 2011. LOCKSMITH: Practical Static Race Detection for C. *ACM Trans. Program. Lang. Syst.* 33, 1, Article 3 (Jan. 2011), 55 pages. https://doi.org/10.1145/1889997.1890000

[35] Jake Roemer, Kaan Genç, and Michael D. Bond. 2018. High-coverage, Unbounded Sound Predictive Race Detection. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2018)*. ACM, New York, NY, USA, 374–389. https://doi.org/10.1145/3192366.3192385

[36] Mahmoud Said, Chao Wang, Zijiang Yang, and Karem Sakallah. 2011. Generating Data Race Witnesses by an SMT-based Analysis. In *Proceedings of the Third International Conference on NASA Formal Methods (NFM'11)*. Springer-Verlag, Berlin, Heidelberg, 313–327. http://dl.acm.org/citation.cfm?id=1986308.1986334

[37] Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, and Thomas Anderson. 1997. Eraser: A Dynamic Data Race Detector for Multithreaded Programs. *ACM Trans. Comput. Syst.* 15, 4 (Nov. 1997), 391–411. https://doi.org/10.1145/265924.265927

[38] Koushik Sen, Grigore Roşu, and Gul Agha. 2005. Detecting Errors in Multithreaded Programs by Generalized Predictive Analysis of Executions. In *Formal Methods for Open Object-Based Distributed Systems*, Martin Steffen and Gianluigi Zavattaro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 211–226.

[39] Konstantin Serebryany and Timur Iskhodzhanov. 2009. ThreadSanitizer: Data Race Detection in Practice. In *Proceedings of the Workshop on Binary Instrumentation and Applications (WBIA '09)*. Association for Computing Machinery, New York, NY, USA, 62–71. https://doi.org/10.1145/1791194.1791203

[40] Dennis Shasha and Marc Snir. 1988. Efficient and Correct Execution of Parallel Programs That Share Memory. *ACM Trans. Program. Lang. Syst.* 10, 2 (April 1988), 282–312. https://doi.org/10.1145/42190.42277

[41] Yannis Smaragdakis, Jacob Evans, Caitlin Sadowski, Jaeheon Yi, and Cormac Flanagan. 2012. Sound Predictive Race Detection in Polynomial Time. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '12)*. ACM, New York, NY, USA, 387–400. https://doi.org/10.1145/2103656.2103702

[42] Chao Wang, Sudipta Kundu, Malay Ganai, and Aarti Gupta. 2009. Symbolic Predictive Analysis for Concurrent Programs. In *Proceedings of the 2Nd World Congress on Formal Methods (FM '09)*. Springer-Verlag, Berlin, Heidelberg, 256–272. https://doi.org/10.1007/978-3-642-05089-3_17

[43] Ryan Williams. 2005. A New Algorithm for Optimal 2-constraint Satisfaction and Its Implications. *Theor. Comput. Sci.* 348, 2 (Dec. 2005), 357–365. https://doi.org/10.1016/j.tcs.2005.09.023

[44] M. Zhivich and R. K. Cunningham. 2009. The Real Cost of Software Errors. *IEEE Security and Privacy* 7, 2 (March 2009), 87–90. https://doi.org/10.1109/MSP.2009.56