An Online Dynamic Analysis for Sound Predictive Data Race Detection *

Ohio State CSE Technical Report #OSU-CISRC-11/16-TR05, November 2016

Jake Roemer
Ohio State University
roemer.37@osu.edu

Michael D. Bond
Ohio State University
mikebond@cse.ohio-state.edu

Abstract

Dynamic analyses struggle to detect all of a program's data races, since some data races do not manifest in observed executions. *Predictive* analyses detect true data races that exist only in *other, unobserved* executions. Smaragdakis et al. introduce the *causally-precedes* (CP) relation and a polynomial-time analysis for *sound* (no false races) predictive data race detection. However, their analysis and other existing sound predictive analyses cannot scale beyond analyzing bounded windows of execution traces.

This work introduces a novel dynamic analysis called *Raptor* that computes CP soundly and completely. Unlike existing analyses, Raptor is inherently an online analysis that analyzes an execution trace in its entirety and finds all of its CP-races. An evaluation of a prototype implementation of Raptor shows that it scales to program executions that existing analyses cannot handle, finding data races that existing analyses cannot find.

1. Introduction

Data races are notorious for causing unexpected and erroneous behaviour in concurrent programs. An execution has a *data race* if two accesses are *conflicting* and *concurrent*, meaning that the accesses are by different threads, at least one is a write, and they are unordered by the *happens-before* relation [2, 34]. Data races commonly lead to atomicity, order, and sequential consistency violations that can cause programs to crash, hang, or corrupt data [6, 13, 15, 26, 31, 33, 35, 38, 45, 48, 53, 55, 57]. Modern shared-memory programming languages such as Java and C++ provide weak, if any, semantics for executions with data races [1, 7–10, 39].

Existing analyses that detect data races tend to be either unsound, meaning that they report false data races,[1] or their coverage is limited to data races that manifest in the current execution. Developers generally avoid unsound analyses (analyses that permit false data races) because each reported race—whether true or false—takes substantial time to investigate [3, 11, 25, 28, 40, 45].

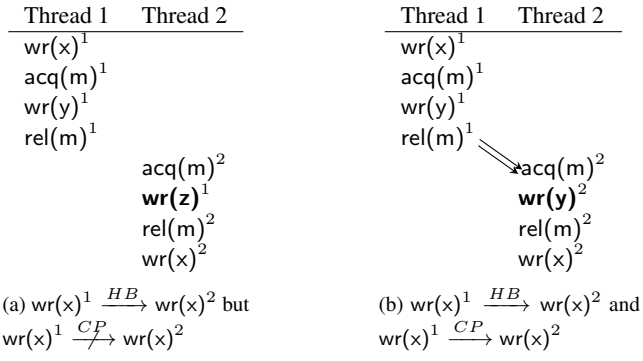

Figure 1. Two executions in which x, y, and z are shared variables, and m is a lock. In each execution, the superscripts differentiate repeat operations. The double arrow represents a "hard" CP edge established by Rule (a) of the CP definition (Section 3).

Happens-before (HB) analysis tracks the happens-before relation in order to detect conflicting, concurrent accesses and report them as data races [21, 25, 49]. However, the coverage of HB analysis is inherently limited to data races that manifest in the current execution. Consider the example executions in Figure 1. In both executions, the two writes to x are ordered by HB, which is the union of program and synchronization order [34]. HB analysis thus would *not* report a data race for either observed execution. However, for Figure 1(a), we can see *from the execution alone* that a data race definitely exists (if Thread 2 acquired m first, then the writes to x would be unordered by HB). In contrast, in Figure 1(b), we cannot say for certain that a data race exists. For example, $\mathsf{wr(x)}^2$'s occurrence might depend on the order of accesses to y. (Assume that $\mathsf{wr(y)}^2$ means not only a write to y but also a possible *read* of y, giving the possibility of a data dependence that could affect control flow.)

Predictive analyses detect data races that are possible in executions *other than* the observed execution. *Sound* predictive analyses report only true data races that are possible in some execution [16, 29, 30, 37, 51, 54, 56] (Section 11). Unfortunately, all existing sound predictive analyses cannot scale to full execution traces. Unlike *online* dynamic analyses that summarize the execution so far in the form of *anal-*

* This material is based upon work supported by the National Science Foundation under Grants CSR-1218695, CAREER-1253703, CCF-1421612, and XPS-1629126.

[1] Following prior work on predictive data race detection (e.g. [56]), an analysis is *sound* if it reports only true data races.

ysis state, these analyses are fundamentally *offline*, needing access to the entire execution trace. To handle nontrivial execution traces, existing sound predictive analyses analyze small, bounded windows of execution. As a result, they miss predictive races involving accesses that are "far apart" in the observed execution.

This paper focuses on the *causally-precedes* (CP) relation introduced by Smaragdakis et al. [56]. CP is a subset of HB that conservatively orders conflicting accesses that *may not* race in some other execution. That is, all conflicting accesses *not* ordered by CP are definitely data races in some execution.[2] In Figure 1(a), although $\mathsf{wr(x)}^1 \xrightarrow{HB} \mathsf{wr(x)}^2$, $\mathsf{wr(x)}^1 \not\xrightarrow{CP} \mathsf{wr(x)}^2$. In contrast, in Figure 1(b), $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ because CP conservatively captures the fact that a data race may *not* occur if the critical sections execute in the reverse order. Smaragdakis et al. show how to compute CP in polynomial time in the execution length. However, their inherently offline analysis cannot scale to full executions, and instead analyzes bounded execution windows of 500 consecutive events [56]. Section 3 motivates why developing an online analysis for CP is challenging.

Our approach. This paper introduces a novel sound predictive dynamic analysis called *Raptor* (Race predictor) that provably computes the CP relation exactly (i.e., soundly and completely). Raptor is inherently an *online* analysis because it summarizes an execution's behavior so far in the form of *analysis state*, rather than needing to look at the entire execution so far. Raptor's key insights lie in how it captures the dependent, recursive nature of the CP relation.

We have implemented Raptor as a dynamic analysis for Java programs. Although our prototype implementation is largely unoptimized, it can analyze executions of real programs with hundreds of thousands or millions of events within an hour or two. In contrast, Smaragdakis et al.'s analysis generally cannot scale beyond bounded windows of thousands of events [56]. As a result, Raptor detects some *CP-races* (conflicting accesses unordered by CP) that are too "far apart" for the offline CP analysis (or other predictive race detection analyses) to detect.

Raptor advances the state of the art by (1) being the first online, sound predictive data race detection analysis and (2) demonstrably scaling to full executions and finding real CP-races that existing analyses cannot detect.

2. Definitions

This section gives definitions of events and of causally-precedes (CP) and other relations. Then Section 3 motivates the challenges of computing CP online.

The initial presentation of definitions and the Raptor analysis considers executions with writes, but not reads, to program variables. Section 9 shows how to extend Raptor to handle reads as well as writes.

[2] More precisely, two conflicting accesses unordered by CP indicate either a data race or a deadlock in some other execution [56].

2.1 Events

An execution consists of a series of events observed in a total order. An event is one of $\mathsf{wr(x)}^i$, $\mathsf{acq(m)}^i$, or $\mathsf{rel(m)}^i$. $\mathsf{wr(x)}^i$ is the ith write to variable x in the observed total order. $\mathsf{acq(m)}^i$ and $\mathsf{rel(m)}^i$ are the ith acquire and release of lock m, respectively, in the observed total order.

We define a total order $\prec_{trace}$ for the observed order of events: for two events e and e', $e \prec_{trace} e'$ if e occurs before e' in the total order. We define a helper function $thr(e)$ that returns the thread identifier that executed event e.

2.2 Relations

The following presentation of relation definitions is closely based on the presentation from prior work [56].

Program order. The *program order* (PO) relation, $\xrightarrow{PO}$, orders events executed by the same thread. For two events e and e', $e \xrightarrow{PO} e'$ if $(e \prec_{trace} e' \wedge thr(e) = thr(e')) \vee e = e'$.

Happens-before. The *happens-before* (HB) relation [34], $\xrightarrow{HB}$, is the smallest relation such that:

- Two events are ordered by HB if they are ordered by PO. That is, for two events e and e', $e \xrightarrow{HB} e'$ if $e \xrightarrow{PO} e'$.
- Release and acquire operations on the same lock (i.e., synchronization order) are ordered by HB. That is, for two events e and e', $e \xrightarrow{HB} e'$ if $e = \mathsf{rel(m)}^i \wedge e' = \mathsf{acq(m)}^j \wedge e \prec_{trace} e'$ (which implies $i < j$).
- HB is closed under composition with itself. That is, for two events e and e', $e \xrightarrow{HB} e'$ if $\exists e'' \mid e \xrightarrow{HB} e'' \wedge e'' \xrightarrow{HB} e'$.

Causally-precedes. The *causally-precedes* (CP) relation [56], $\xrightarrow{CP}$, is the smallest relation such that:

(a) Two critical sections on the same lock are ordered by CP if they contain conflicting accesses (accesses to the same variable by different threads). That is, $\mathsf{rel(m)}^j \xrightarrow{CP} \mathsf{acq(m)}^k$ if there exist events $e = \mathsf{wr(x)}^h$ and $e' = \mathsf{wr(x)}^i$ such that $e \prec_{trace} e' \wedge thr(e) \neq thr(e') \wedge \mathsf{acq(m)}^j \xrightarrow{PO} e \xrightarrow{PO} \mathsf{rel(m)}^j \wedge \mathsf{acq(m)}^k \xrightarrow{PO} e' \xrightarrow{PO} \mathsf{rel(m)}^k$.

(b) Two critical sections on the same lock are ordered by CP if they contain events that are ordered by CP. Because of the next rule, this rule can be expressed simply as follows: $\mathsf{rel(m)}^i \xrightarrow{CP} \mathsf{acq(m)}^j$ if $\mathsf{acq(m)}^i \xrightarrow{CP} \mathsf{rel(m)}^j$.

(c) CP is closed under left and right composition with HB. That is, for two events e and e', $e \xrightarrow{CP} e'$ if $\exists e'' \mid e \xrightarrow{HB} e'' \xrightarrow{CP} e' \vee e \xrightarrow{CP} e'' \xrightarrow{HB} e'$

The rest of this paper refers to the above rules as *Rules (a), (b), and (c)*, respectively, of the CP definition.

Note that CP, which is formed by relations from release to acquire of the same lock and transitively with HB, is a

subset of HB. Unlike PO and HB, CP is not reflexive and thus not a partial order [56].

A *CP-race* exists when two conflicting accesses are not ordered by the CP relation. That is, a CP-race exists between two events $e = \mathsf{wr(x)}^i$ and $e' = \mathsf{wr(x)}^j$ if $e \not\xrightarrow{PO} e' \wedge e' \not\xrightarrow{PO} e \wedge e \not\xrightarrow{CP} e' \wedge e' \not\xrightarrow{CP} e$.

Smaragdakis et al. prove that the CP relation is sound [56]. In particular, if a CP-race exists, then there exists an execution that has (1) a true data race (i.e., two conflicting accesses unordered by HB) or (2) a deadlock.

Example. In both executions of Figure 1 (page 1), $\mathsf{wr(x)}^1 \xrightarrow{HB} \mathsf{wr(x)}^2$ because $\mathsf{rel(m)}^1 \xrightarrow{HB} \mathsf{acq(m)}^2 \wedge \mathsf{wr(x)}^1 \xrightarrow{PO} \mathsf{rel(m)}^1 \wedge \mathsf{acq(m)}^2 \xrightarrow{PO} \mathsf{wr(x)}^2$. In Figure 1(a), $\mathsf{wr(x)}^1 \not\xrightarrow{CP} \mathsf{wr(x)}^2$. In contrast, in Figure 1(b), $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ because $\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2 \wedge \mathsf{wr(x)}^1 \xrightarrow{HB} \mathsf{rel(m)}^1 \wedge \mathsf{acq(m)}^2 \xrightarrow{HB} \mathsf{wr(x)}^2$ ($\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2$ by Rule (a) of the CP definition def; CP composes with HB by Rule (c)).

3. Problem, Background, and Motivation

In addition to introducing the causally-precedes (CP) relation, Smaragdakis et al. introduce an algorithm for detecting CP-races in program executions [56]. Their algorithm encodes the recursive definition of CP in Datalog, guaranteeing polynomial-time execution in the size of the execution trace. The algorithm is inherently *offline* because it fundamentally needs to "look back" at the entire execution trace. Experimentally, Smaragdakis et al. find that their algorithm does not scale to full program traces. Instead, they limit their algorithm's computation to bounded windows of 500 consecutive events [56].

This paper targets the challenge of developing an *online* analysis for tracking the CP relation and detecting CP-races. An online analysis must (1) compute CP soundly and completely; (2) maintain analysis state that summarizes the execution so far, without needing to maintain and refer to the entire execution trace; and (3) analyze real program execution traces using time and space that is "reasonable" for heavyweight in-house testing.

As Smaragdakis et al. explain [56], developing an online analysis for CP is inherently challenging:

> *CP reasoning, based on* [the definition of CP], *is highly recursive. Notably, Rule (c) can feed into Rule (b), which can feed back into Rule (c). As a result, we have not implemented CP using techniques such as vector clocks, nor have we yet discovered a full CP implementation that only does online reasoning (i.e., never needs to "look back" in the execution trace).*

Other existing predictive analyses are either unsound (reporting false races), or (like the existing CP work [56]) are limited to analyzing bounded windows of execution [16, 29, 30, 37, 51, 54]. Section 11 provides more details.

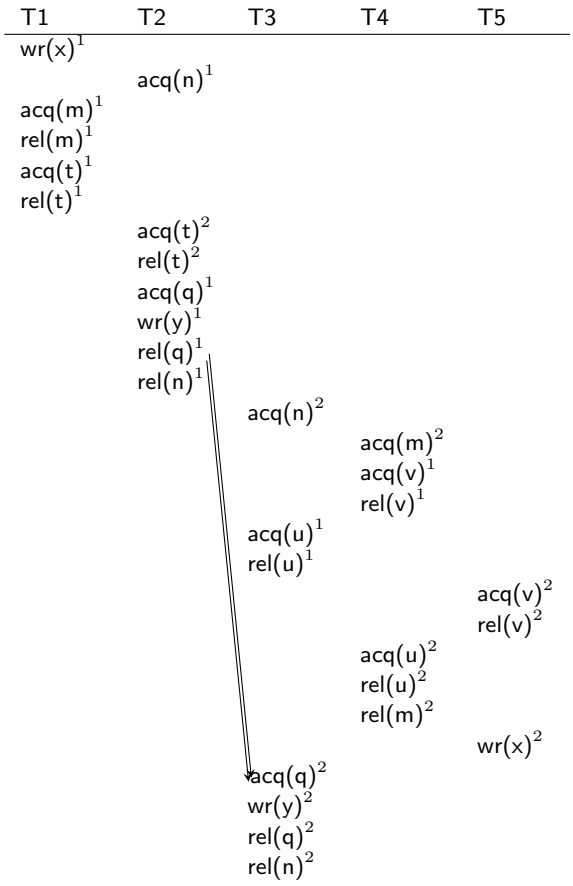

Figure 2. An example execution in which $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$. The double arrow indicates a CP relation established by Rule (a) of the CP definition (called a "hard" CP edge by prior work [56]).

Figure 2 illustrates some of the challenges involved in developing an online analysis for CP. In this example, the writes are CP ordered because the critical sections on m are CP ordered, which is true because the critical sections on n are CP ordered. More precisely, $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ through the following logic: $\mathsf{rel(q)}^1 \xrightarrow{CP} \mathsf{acq(q)}^2$ by Rule (a) implies $\mathsf{acq(n)}^1 \xrightarrow{CP} \mathsf{rel(n)}^2$ by Rule (c), which implies $\mathsf{rel(n)}^1 \xrightarrow{CP} \mathsf{acq(n)}^2$ by Rule (b), which implies $\mathsf{acq(m)}^1 \xrightarrow{CP} \mathsf{rel(m)}^2$ by Rule (c), which implies $\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2$ by Rule (b), which implies $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ by Rule (c).

However, at the event $\mathsf{rel(m)}^2$, an online analysis *cannot* determine that $\mathsf{acq(m)}^1 \xrightarrow{CP} \mathsf{rel(m)}^2$ (and thus $\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2$) because it is *not yet knowable* that $\mathsf{acq(n)}^1 \xrightarrow{CP} \mathsf{rel(n)}^2$ (and thus $\mathsf{rel(n)}^1 \xrightarrow{CP} \mathsf{acq(n)}^2$).

Furthermore, at $\mathsf{wr(x)}^2$, an online analysis *cannot* determine that $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ because it is not yet knowable. Not until $\mathsf{wr(y)}^2$ is it knowable that $\mathsf{acq(n)}^1 \xrightarrow{CP} \mathsf{rel(n)}^2$ and thus $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$.

Now suppose instead that T1 executed its critical section on t *before* its critical section on m. In that subtly different execution, $\mathsf{wr(x)}^1 \overset{CP}{\not\longrightarrow} \mathsf{wr(x)}^2$. A sound and complete online analysis for CP must track analysis state that captures the difference between these two different cases.

4. Raptor Overview

Raptor (Race predictor) is an online dynamic analysis that computes CP ordering soundly and completely. To do so, Raptor also computes HB and PO ordering, since the definition of CP relies on HB, and the definition of CP-race relies on PO. This section overviews key features of Raptor.

Locksets. Most HB analyses represent analysis state using *vector clocks* [25, 41, 49]. In contrast, Raptor's analysis state is in the form of "locksets." Raptor's locksets are most similar to the locksets used by *Goldilocks*, a sound and complete *HB* data race detector [21] (Section 11.2). The Goldilocks algorithm is essentially the same as Raptor's algorithm *for HB only*. (The locksets used by Raptor and Goldilocks are quite different from those used by *lockset analyses*, which check a *locking discipline* and inherently report false data races [17, 18, 46, 47, 52, 59]; Section 11.2.)

The basic idea of Raptor's locksets is that they contain synchronization objects—locks and threads—that are ordered by CP, HB, and PO. For example, if a lock m is an element of the lockset $HB(\mathsf{x}^8)$, it means that any future event that acquires m is HB ordered to $\mathsf{wr(x)}^8$. Similarly, $\mathsf{T2} \in CP(\mathsf{y}^3)$ means that $\mathsf{wr(y)}^3$ is CP ordered with future events by thread T2. Throughout the rest of the paper, we say that a lock m or thread T is CP (or HB) *ordered to* an event e if, for any future event e' that acquires m or is executed by T, respectively, $e \overset{CP}{\longrightarrow} e'$ ($e \overset{HB}{\longrightarrow} e'$).

Locksets for each access to a variable. As implied above, rather than each variable x having CP, HB, and PO locksets, in fact every *access* $\mathsf{wr(x)}^i$ has its own CP, HB, and PO locksets. Per-access locksets are needed for CP locksets in particular because of the nature of the CP relation: at $\mathsf{wr(x)}^{i+1}$, it is not in general knowable whether $\mathsf{wr(x)}^i \overset{CP}{\longrightarrow} \mathsf{wr(x)}^{i+1}$. For example, in Figure 2, even after $\mathsf{wr(x)}^2$ executes, Raptor must continue to maintain information for $\mathsf{wr(x)}^1$ because $\mathsf{wr(x)}^1 \overset{CP}{\longrightarrow} \mathsf{wr(x)}^2$ has not yet been established.

Maintaining locksets for every variable access would seem to require massive time and space (proportional to the length of the execution), making it as unscalable to full execution traces as prior work's offline approach for computing CP [56]. However, as we show, Raptor can safely delete an access $\mathsf{wr(x)}^i$'s locksets as soon as it determines whether $\mathsf{wr(x)}^{i+1}$ is CP ordered to $\mathsf{wr(x)}^i$.

Locksets for lock acquires. Raptor tracks CP, HB, and PO locksets not just for variable accesses, but also for lock acquire operations (e.g., $\mathsf{acq(m)}^i$). The reason is that Raptor needs to detect whether $\mathsf{acq(m)}^i \overset{CP}{\longrightarrow} \mathsf{rel(m)}^j$, which by Rule (b) of the CP definition implies that $\mathsf{rel(m)}^i \overset{CP}{\longrightarrow} \mathsf{acq(m)}^j$. For example, $\mathsf{T3} \in CP(\mathsf{m}^i)$ means that for any future event e, $\mathsf{acq(m)}^i \overset{CP}{\longrightarrow} e$ if $thr(e) = \mathsf{T3}$.

Similar to locksets for variable accesses, maintaining a lockset for each lock acquire might consume high time and space proportional to the execution's length. However, we show how Raptor can safely delete an acquire $\mathsf{acq(m)}^i$'s locksets once they are no longer needed—once no other CP ordering is dependent on the possibility of $\mathsf{acq(m)}^i$ being CP ordered with a future $\mathsf{rel(m)}$.

In contrast, Goldilocks does not need or use locksets for each variable access, nor locksets for lock acquires, since it maintains only HB locksets [21].

Conditional CP locksets. As mentioned above, it is unknowable in general at an event $\mathsf{wr(x)}^{i+1}$ whether $\mathsf{wr(x)}^i \overset{CP}{\longrightarrow} \mathsf{wr(x)}^{i+1}$. This delayed knowledge is due to Rule (b) of the CP definition, which states that $\mathsf{rel(m)}^i \overset{CP}{\longrightarrow} \mathsf{acq(m)}^j$ if $\mathsf{acq(m)}^i \overset{CP}{\longrightarrow} \mathsf{rel(m)}^j$—so a CP ordering might not be known until $\mathsf{rel(m)}^j$ executes. The knowledge of a CP ordering may be delayed even further since Rule (c) can "feed into" Rule (b), which can feed back into Rule (c). This recursive nature of the CP definition prevents immediate detection of CP ordering.

Raptor maintains *conditional CP* (CCP) locksets to capture the fact that, at a given point in an execution, a CP ordering may or may not exist, depending on whether some *other* CP ordering exists. An element $\mathsf{n} : \mathsf{m}^j$ in lockset $CCP(\mathsf{x}^i)$ means that a future $\mathsf{rel(n)}$ event is CP ordered to $\mathsf{wr(x)}^i$ *if* $\mathsf{acq(m)}^j \overset{CP}{\longrightarrow} \mathsf{rel(m)}^k$ for some future $\mathsf{rel(m)}^k$. As another example, $\mathsf{T} : \mathsf{m}^j \in CCP(\mathsf{n}^i)$ means that $\mathsf{acq(n)}^i$ is CP ordered to future events by thread T *if* $\mathsf{acq(m)}^j \overset{CP}{\longrightarrow} \mathsf{rel(m)}^k$ for some future $\mathsf{rel(m)}^k$.

Outline of Raptor presentation. Section 5 describes Raptor's locksets and their elements in detail, and it presents invariants maintained by Raptor's locksets before and after every event from the program execution trace. Section 6 introduces the Raptor analysis that adds (and in a few cases, removes) lockset elements at each execution event. Section 7 proves that the Raptor analysis in fact adheres to the proposed invariants (proof details in submitted supplementary material). Section 8 describes how Raptor removes "obsolete" locksets and detects CP-races. Until Section 9, the paper considers only variable writes but not reads; Section 9 shows how to extend Raptor to support read events.

5. Raptor's Analysis State

This section introduces the analysis state that Raptor maintains. Every *lockset owner* ρ, which can be a variable access instance x^i or lock acquire instance m^i, has the following locksets: $PO(\rho)$, $HB(\rho)$, $CP(\rho)$, and $CCP(\rho)$. In general, elements of locksets are threads T and locks m, with a few caveats: $HB(\rho)$ maintains an index for each lock element (e.g., m^j); and each $CCP(\rho)$ element includes an associated

lock instance upon which CP ordering is conditional (e.g., $\mathsf{m} : \mathsf{n}^j$ or $\mathsf{T} : \mathsf{n}^j$). In addition, each lockset for a variable instance x^i can contain a special element ξ, which indicates ordering between $\mathsf{wr(x)}^i$ and $\mathsf{wr(x)}^{i+1}$.

Figure 3 shows invariants that the Raptor analysis maintains for every lockset owner ρ. The rest of this section explains these invariants in detail.

5.1 Program Order Lockset: $PO(\rho)$

According to the *PO invariant* in Figure 3, $PO(\rho)$ contains all locks and threads that are PO ordered with e_ρ. From the definition of PO, we know that only one thread (the thread that executes e_ρ) will be PO ordered to e_ρ. According to the invariant, no locks will ever be PO ordered to e_ρ.

In addition, for any $\rho = \mathsf{x}^h$, $PO(\mathsf{x}^h)$ may contain the special element ξ, which indicates that $\mathsf{wr(x)}^h$ is PO ordered with the next access to x, i.e., $\mathsf{wr(x)}^h \xrightarrow{PO} \mathsf{wr(x)}^{h+1}$. (Raptor does not strictly need ξ in $PO(\mathsf{x}^h)$ to indicate $\mathsf{wr(x)}^h \xrightarrow{PO} \mathsf{wr(x)}^{h+1}$, since $PO(\mathsf{x}^h)$ does not add new threads or locks after $\mathsf{wr(x)}^h$ executes. However, since Raptor needs ξ for CP and CCP locksets, it adds ξ to PO and HB locksets for consistency.)

5.2 Happens-Before Lockset: $HB(\rho)$

The $HB(\rho)$ lockset contains threads and locks that are HB ordered to e_ρ. Figure 3 states three invariants for $HB(\rho)$: the *HB, HB-index, and HB-critical-section* invariants.

The HB invariant defines which threads and locks are in $HB(\rho)$. Each thread or lock is HB ordered to e_ρ, meaning that some event (before e) by the same thread or release of the same lock, respectively, is HB ordered to e_ρ. This property implies that if any future event (e or a later event) executes on the same thread or acquires the same lock, respectively, it will be HB ordered to e_ρ. In addition, like $PO(\rho)$, $HB(\rho)$ can contain a special element ξ, which indicates, for $\rho = \mathsf{x}^h$ only, that $\mathsf{wr(x)}^h \xrightarrow{HB} \mathsf{wr(x)}^{h+1}$.

According to the HB-index invariant, every lock m in $HB(\rho)$ has a superscript i (i.e., m^i) that specifies the earliest lock release that is HB ordered to e_ρ. For example, $\mathsf{m}^i \in HB(\rho)$ means that $\mathsf{rel(m)}^i$ is HB ordered to e_ρ (i.e., $e_\rho \xrightarrow{HB} \mathsf{rel(m)}^i$), but no earlier release of m is HB ordered to e_ρ. Raptor tracks this property in order to know, at some future event $\mathsf{acq(m)}^j$, which critical section m^i needs to be CP ordered with m^j (i.e., $\mathsf{acq(m)}^i \xrightarrow{CP} \mathsf{rel(m)}^j$) in order to imply that $e_\rho \xrightarrow{CP} \mathsf{acq(m)}^j$ (by Rules (b) and (c) of the CP definition).

According to the HB-critical-section invariant, for $\rho = \mathsf{x}^h$ only, m^i in $HB(\rho)$ may have a subscript $*$ (i.e., m^i_*), which means that not only $e_\rho \xrightarrow{HB} \mathsf{rel(m)}^i$, but also that e_ρ executed *inside* the $\mathsf{acq(m)}^i$–$\mathsf{rel(m)}^i$ critical section. Raptor must track this property in order to apply Rule (a) of the CP definition precisely.

5.3 Causally-Precedes Lockset: $CP(\rho)$

Analogous to $HB(\rho)$ for HB ordering, each $CP(\rho)$ lockset contains locks and threads that are CP ordered to e_ρ. However, at an event e', it is not in general possible to tell whether $e_\rho \xrightarrow{CP} e'$, due to Rule (b) of the CP definition. Thus, a lock or thread σ is *not* necessarily in $CP(\rho)$ even after an event e' such that $appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e'$.

This property of CP presents two main challenges. First, CP ordering may be determined later, dependent on other CP relations, according to Rule (b) of the CP definition. Raptor introduces the $CCP(\rho)$ lockset (described below) to capture potential CP ordering that may be determined later. For every lock and thread that is ordered to e_ρ, Raptor captures that fact either eagerly using $CP(\rho)$ or lazily using $CCP(\rho)$, as the *CP invariant* in Figure 3 shows.

Second, as a result of computing CP lazily, how does Raptor determine whether there is a CP-race between two events $\mathsf{wr(x)}^i$ and $\mathsf{wr(x)}^{i+1}$? In particular, suppose that at some event *after* thread T executes $\mathsf{wr(x)}^{i+1}$, Raptor determines that T is CP ordered to $\mathsf{wr(x)}^i$; then how does Raptor know whether $\mathsf{wr(x)}^i \xrightarrow{CP} \mathsf{wr(x)}^{i+1}$? Raptor addresses this challenge by using the special thread-like element ξ that essentially represents the thread T *up to event* $\mathsf{wr(x)}^{i+1}$ *only*, so $\xi \in CP(\mathsf{x}^i)$ only if $\mathsf{wr(x)}^i \xrightarrow{CP} \mathsf{wr(x)}^{i+1}$.

The *CP-rule-A invariant* (Figure 3) covers a case for which Raptor always computes CP eagerly: when two critical sections on the same lock have conflicting events (e.g., $\mathsf{wr(x)}^j$ and $\mathsf{wr(x)}^k$), according to Rule (a) of the CP definition. In this case, the invariant states that if two critical sections, m^h and m^i, are CP ordered by Rule (a) alone, then $\mathsf{T} \in CP(\mathsf{m}^i)$ as soon as $\mathsf{wr(x)}^k$ executes ($\mathsf{T} = thr(\mathsf{wr(x)}^k)$). The CP-rule-A invariant is useful in proving that Raptor maintains the CP invariant (Section 7).

5.4 Conditionally Causally-Precedes Lockset: $CCP(\rho)$

Rule (b) of the CP definition states that $\mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j$ if $\mathsf{acq(n)}^k \xrightarrow{CP} \mathsf{rel(n)}^j$. To account for this rule, Raptor introduces the $CCP(\rho)$ lockset, which contains elements of the form $\sigma : \mathsf{n}^k$, which means that σ is CP ordered with e_ρ **if** $\mathsf{acq(n)}^k \xrightarrow{CP} \mathsf{rel(n)}^j$, where n^j is the current ongoing critical section of n.

The CP invariant in Figure 3 shows that for every CP ordering, Raptor captures it eagerly in a CP lockset or lazily in a CCP lockset (or both). A further constraint, codified in the *CCP-constraint invariant*, is that $\sigma : \mathsf{n}^k \in CCP(\rho)$ *only if* a critical section on lock n is ongoing. As Section 6 shows, when n's current critical section ends (at $\mathsf{rel(n)}^j$), Raptor either (1) determines whether $\mathsf{acq(n)}^k \xrightarrow{CP} \mathsf{rel(n)}^j$, or (2) it identifies another lock q that has an ongoing critical section such that it is correct to add some $\sigma : \mathsf{q}^f$ to $CCP(\rho)$.

Like $CP(\rho)$, when $\rho = \mathsf{x}^i$, $CCP(\rho)$ can contain the special thread-like element ξ. More precisely, $\xi : \mathsf{n}^k \in$

Let e be any event e in the program trace. The following invariants hold for the point in the trace immediately *before* e.
Let $e^* = \mathsf{wr(x)}^{h+1}$ if $e = \mathsf{wr(x)}^h$; otherwise (e is a lock acquire or release event), e^* is an "invalid event" that matches no real event.
We define the boolean function $appl(\sigma, e')$ that returns whether event e' "applies to" lockset element σ:

$$appl(\sigma, e') \coloneqq \begin{cases} thr(e') = \mathsf{T} & \text{if } \sigma \text{ is a thread } \mathsf{T} \\ \exists i \mid e' = \mathsf{rel(m)}^i & \text{if } \sigma \text{ is a lock } \mathsf{m} \\ e' = e^* & \text{otherwise } (\sigma \text{ is } \xi) \end{cases}$$

The following invariants hold for every lockset owner ρ. For each lockset owner ρ, let e_ρ be the event corresponding to ρ, i.e., $e_\rho = \mathsf{wr(x)}^h$ if $\rho = \mathsf{x}^h$, or $e_\rho = \mathsf{acq(m)}^h$ if $\rho = \mathsf{m}^h$.

[PO] $PO(\rho) = \{\, \sigma \mid \sigma \text{ is not a lock} \wedge \left(\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{PO} e' \prec_{trace} e\right) \}$

[HB] $HB(\rho) = \{\, \sigma \mid \left(\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{HB} e' \prec_{trace} e\right) \}$

[HB-index] $\mathsf{m}^i \in HB(\rho) \iff \left(e_\rho \xrightarrow{HB} \mathsf{rel(m)}^{i-1} \wedge e_\rho \xrightarrow{HB} \mathsf{rel(m)}^i \prec_{trace} e\right)$

[HB-critical-section] $\mathsf{m}^i_* \in HB(\rho) \iff \left(\mathsf{acq(m)}^i \xrightarrow{PO} e_\rho \xrightarrow{PO} \mathsf{rel(m)}^i \wedge e_\rho = \mathsf{wr(x)}^h \wedge e_\rho \prec_{trace} e\right)$

[CP] $CP(\rho) \cup \{\, \sigma \mid \left(\exists \mathsf{n}^k \mid \sigma\!:\!\mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e\right) \} = \{\, \sigma \mid \left(\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e\right) \}$

[CP-rule-A] $\left(\rho = \mathsf{m}^h \wedge \exists \mathsf{x}, \mathsf{T}, i, j, k \mid e_\rho \xrightarrow{PO} \mathsf{wr(x)}^j \xrightarrow{PO} \mathsf{rel(m)}^h \prec_{trace} \mathsf{acq(m)}^i \xrightarrow{PO} \mathsf{wr(x)}^k \xrightarrow{PO} \mathsf{rel(m)}^i \wedge \mathsf{wr(x)}^k \prec_{trace} e \wedge thr(\mathsf{wr(x)}^k) = \mathsf{T}\right) \implies \mathsf{T} \in CP(\rho)$

[CCP-constraint] $\exists \sigma\!:\!\mathsf{n}^k \in CCP(\rho) \implies \left(\exists j \mid \mathsf{acq(n)}^j \prec_{trace} e \wedge \mathsf{rel(n)}^j \not\prec_{trace} e\right)$

Figure 3. The invariants maintained by the Raptor analysis before and after every event in the observed total order.

$CCP(\mathsf{x}^i)$ means that $\mathsf{wr(x)}^i \xrightarrow{CP} \mathsf{wr(x)}^{i+1}$ *if* $\mathsf{acq(n)}^k \xrightarrow{CP} \mathsf{rel(n)}^j$, where n^j is the current ongoing critical section of n.

6. The Raptor Analysis

Raptor is an online dynamic analysis that maintains the invariants shown in Figure 3 and explained in Section 5. For each event e in the observed total order, Raptor updates the analysis state, which consists of the locksets $PO(\rho)$, $HB(\rho)$, $CP(\rho)$, and $CCP(\rho)$ for each lockset owner ρ. Assuming that immediately *before* e, the analysis state satisfies the invariants, then at event e, Raptor modifies the analysis state so that it satisfies the invariants immediately *after* e. Section 7 (with details in supplementary material) proves this claim.

Raptor modifies its analysis state at event e by adding, and in some cases removing, elements from ρ's locksets. To differentiate the state immediately *after* e from the state immediately *before* e, we use the following notation in the analysis. The analysis represents the state immediately *before* e as $PO(\rho)$, $HB(\rho)$, $CP(\rho)$, and $CCP(\rho)$. The analysis represents the state immediately *after* e by adding a superscript $+$, i.e., $PO(\rho)^+$, $HB(\rho)^+$, $CP(\rho)^+$, and $CCP(\rho)^+$.

6.1 Initialization

For every lockset owner ρ, whether it is a variable instance x^i or a lock instance m^i, its locksets are initially empty, i.e., $PO(\rho) = HB(\rho) = CP(\rho) = CCP(\rho) = \emptyset$. This initial state conforms to Figure 3's invariants for the point in execution before any events execute.

In addition, to simplify checking for CP-races, the analysis assumes a "fake" initial access x^0 for every program variable x. The analysis initializes $PO(\mathsf{x}^0)$ to ξ, as Algorithm 1 shows. (The other locksets for x^0 are $\emptyset$.) This initial state ensures that the first access to x, $\mathsf{wr(x)}^1$, will appear to be PO-ordered to the prior access to x, without requiring any logic to handle this corner case.

Algorithm 1 Initialize locksets

```
for all variables x do
    PO(x^0) ← { ξ }
end for
▷ All other locksets are initially empty
```

6.2 Handling Write Events

At a program's write to a potentially shared variable, i.e., $e = \mathsf{wr(x)}^i$, by thread T, the analysis performs the actions in Algorithm 2. The analysis applies Rule (a) of the CP definition (conflicting critical sections are CP ordered); checks for PO, HB, and CP ordering with the prior access $\mathsf{wr(x)}^{i-1}$; and initializes the locksets for x^i.

Applying Rule (a). Lines 2–7 of Algorithm 2 show how the analysis applies Rule (a). The helper function $heldBy(\mathsf{T})$ returns the set of locks currently held by thread T (e.g., locks with active critical sections executed by T). For each lock m held by T, the analysis checks whether a prior access to x executed in a critical section on m, but by a different thread.

If the analysis detects a conflicting critical section m^j, it adds T to $CP(\mathsf{m}^j)$, satisfying the CP-rule-A invariant (Figure 3). Later, when T releases m, the analysis updates other $CP()$ locksets, as Section 6.4 describes.

Checking ordering with prior access. The current event $\mathsf{wr(x)}^i$ may be CP ordered with the prior access $\mathsf{wr(x)}^{i-1}$, but the CP ordering may depend on later events. As Section 5 described, the Raptor analysis handles this situation by using

Algorithm 2 $\mathsf{wr(x)}^i$ by T

1: ▷ Apply Rule (a)
2: **for all** $\mathsf{m} \in heldBy(\mathsf{T})$ **do**
3: **if** $\exists j \exists h \mid \mathsf{m}^j_* \in HB(\mathsf{x}^h) \wedge \mathsf{T} \notin PO(\mathsf{x}^h)$ **then**
4: ▷ Let j be the maximum satisfying index.
5: $CP(\mathsf{m}^j)^+ \leftarrow CP(\mathsf{m}^j)^+ \cup \{\mathsf{T}\}$
6: **end if**
7: **end for**
8: ▷ Add ξ to represent T at $\mathsf{wr(x)}^i$
9: **if** $\mathsf{T} \in PO(\mathsf{x}^{i-1})$ **then**
10: $PO(\mathsf{x}^{i-1})^+ \leftarrow PO(\mathsf{x}^{i-1})^+ \cup \{\xi\}$
11: **end if**
12: **if** $\mathsf{T} \in HB(\mathsf{x}^{i-1})$ **then**
13: $HB(\mathsf{x}^{i-1})^+ \leftarrow HB(\mathsf{x}^{i-1})^+ \cup \{\xi\}$
14: **end if**
15: **if** $\mathsf{T} \in CP(\mathsf{x}^{i-1})$ **then**
16: $CP(\mathsf{x}^{i-1})^+ \leftarrow CP(\mathsf{x}^{i-1})^+ \cup \{\xi\}$
17: **end if**
18: **for all** $\mathsf{m}^j \mid \mathsf{T}:\mathsf{m}^j \in CCP(\mathsf{x}^{i-1})$ **do**
19: $CCP(\mathsf{x}^{i-1})^+ \leftarrow CCP(\mathsf{x}^{i-1})^+ \cup \{\xi:\mathsf{m}^j\}$
20: **end for**
21: ▷ Initialize locksets for x^i
22: $PO(\mathsf{x}^i)^+ \leftarrow \{\mathsf{T}\}$
23: $HB(\mathsf{x}^i)^+ \leftarrow \{\mathsf{T}\} \cup \{\mathsf{m}^j_* \mid \mathsf{m}^j \in heldBy(\mathsf{T})\}$

a special thread-like element ξ in locksets for x^{i-1} that indicates ordering with $\mathsf{wr(x)}^i$.

Lines 9–20 of Algorithm 2 show how the analysis handles this case. If $\mathsf{T} \in CP(\mathsf{x}^{i-1})$, then $\mathsf{wr(x)}^{i-1} \xrightarrow{CP} \mathsf{wr(x)}^i$, and thus the analysis adds ξ to $CP(\mathsf{x}^{i-1})$. The same logic applies for HB, CP, and CCP locksets. Notably, for any m^j such that $\mathsf{T} : \mathsf{m}^j \in CCP(\mathsf{x}^{i-1})$, $\mathsf{wr(x)}^{i-1} \xrightarrow{CP} \mathsf{wr(x)}^i$ if $\mathsf{acq(m)}^j \xrightarrow{CP} \mathsf{rel(m)}^k$ (where m^k is the current critical section on m), and so the analysis adds $\xi:\mathsf{m}^j$ to $CCP(\mathsf{x}^{i-1})$.

Initializing locksets for current access. Lines 22–23 initialize locksets for x^i. (Before this point in the event trace, all locksets for x^i are $\emptyset$; Section 6.1.) In addition to adding T to $PO(\mathsf{x}^i)$ and $HB(\mathsf{x}^i)$, the analysis adds m^j_* to $HB(\mathsf{x}^i)$ for each ongoing critical section on m^j by T, satisfying the HB-critical-section invariant (Figure 3).

6.3 Handling Acquire Events

At a program's acquire of a lock, i.e., $e = \mathsf{acq(m)}^i$ by thread T, the analysis performs the actions in Algorithm 3. The analysis transfers HB and CP ordering from m to T for all ρ; adds new $CCP(\rho)$ elements for potentially CP-ordered critical sections; and initializes the locksets for m^i.

Transferring ordering. HB and CP are both closed under right-composition with HB. Thus, after the current event $e = \mathsf{acq(m)}^i$ by T, any e_ρ that was HB or CP ordered to m is now also HB or CP ordered, respectively, to T. If $\mathsf{m} \in CP(\rho)$, then $e_\rho \xrightarrow{CP} \mathsf{acq(m)}^i$, and thus the analysis

Algorithm 3 $\mathsf{acq(m)}^i$ by T

1: ▷ Transfer ordering from m to T
2: **for all** ρ **do**
3: **if** $\mathsf{m} \in CP(\rho)$ **then**
4: $CP(\rho)^+ \leftarrow CP(\rho)^+ \cup \{\mathsf{T}\}$
5: **end if**
6: **for all** $\mathsf{n}^k \mid \mathsf{m}:\mathsf{n}^k \in CCP(\rho)$ **do**
7: $CCP(\rho)^+ \leftarrow CCP(\rho)^+ \cup \{\mathsf{T}:\mathsf{n}^k\}$
8: ▷ No effect if $\exists k' < k \mid \mathsf{T}:\mathsf{n}^{k'} \in CCP(\rho)^+$
9: **end for**
10: **if** $\exists j \mid \mathsf{m}^j \in HB(\rho) \vee \mathsf{m}^j_* \in HB(\rho)$ **then**
11: $HB(\rho)^+ \leftarrow HB(\rho)^+ \cup \{\mathsf{T}\}$
12: ▷ Add new CCP element
13: $CCP(\rho)^+ \leftarrow CCP(\rho)^+ \cup \{\mathsf{T}:\mathsf{m}^j\}$
14: ▷ No effect if $\exists j' < j \mid \mathsf{T}:\mathsf{m}^{j'} \in CCP(\rho)^+$
15: **end if**
16: **end for**
17: ▷ Initialize locksets for m^i
18: $HB(\mathsf{m}^i)^+ \leftarrow \{\mathsf{T}\}$
19: $PO(\mathsf{m}^i)^+ \leftarrow \{\mathsf{T}\}$

adds T to $CP(\rho)$ (lines 3–5), satisfying the CP invariant. Similarly, lines 10–11 transfer HB ordering from m to T.

The analysis also transfers conditional CP ordering (CCP ordering), in lines 6–9. For any lock critical section n^k, if $\mathsf{m} : \mathsf{n}^k \in CCP(\rho)$, then that means $e_\rho \xrightarrow{CP} \mathsf{rel(m)}^{i-1}$ if $\mathsf{acq(n)}^k \xrightarrow{CP} \mathsf{rel(n)}^j$, where n^j is an ongoing critical section. After the current event $\mathsf{acq(m)}^i$, since HB right-composes with CP, $e_\rho \xrightarrow{CP} \mathsf{acq(m)}^i$ if $\mathsf{acq(n)}^k \xrightarrow{CP} \mathsf{rel(n)}^j$. Thus, the analysis adds $\mathsf{T}:\mathsf{n}^k$ to $CCP(\rho)$.

Adding new CCP ordering. For any e_ρ such that $e_\rho \xrightarrow{HB} \mathsf{rel(m)}^j$ for some j, e_ρ may actually be *CP* ordered to T, i.e., $e_\rho \xrightarrow{CP} \mathsf{acq(m)}^i$, *if* the critical sections on m turn out to be CP ordered. Line 13 of Algorithm 3 handles this case by adding $\mathsf{T}:\mathsf{m}^j$ to $CCP(\rho)$ when $e_\rho \xrightarrow{HB} \mathsf{rel(m)}^j$.

Initializing locksets. Lines 18–19 initialize locksets for m^i. Since any further event executed by T will be PO and HB ordered with $\mathsf{acq(m)}^i$, the analysis adds T to $HB(\mathsf{m}^i)$ and $PO(\mathsf{m}^i)$, satisfying the PO and HB invariants (Figure 3). (The analysis adds T to $PO(\mathsf{m}^i)$ *only* to satisfy Figure 3's PO invariant. It never uses $PO(\mathsf{m}^i)$.)

6.4 Handling Release Events

At a program's release of a lock, i.e., $e = \mathsf{rel(m)}^i$ by thread T, the analysis performs the actions in Algorithm 4, called the "pre-release" algorithm, followed by the actions in Algorithm 5, called the "release" algorithm. We divide Raptor's analysis actions into two algorithms in order to separate out two sets of changes to $CCP(\rho)$ elements: the pre-release algorithm adds elements to $CCP(\rho)$, some of which the release algorithm will use and then remove.

The pre-release algorithm applies Rule (b) of the CP definition, by adding CP ordering and transferring CCP ordering to be dependent on locks *other than* m, to prepare for the removal of all CCP elements dependent on m. The release algorithm operates on the analysis state modified by the pre-release algorithm. The release algorithm transfers CP and HB ordering from T to m, and it removes all CCP elements that are dependent on m.

After Raptor performs the *pre-release* algorithm, Figure 3's invariants still hold—for the execution point *before* e. After Raptor performs the *release* algorithm, Figure 3's invariants hold, for the execution point *after* e.

6.4.1 Pre-release Algorithm (Algorithm 4)

Applying Rule (b) for CP locksets. Lines 3–6 of Algorithm 4 show how the analysis applies Rule (b) of the CP definition. For any $\sigma : \mathsf{m}^j \in CCP(\rho)$, σ is CP ordered to e_ρ if $\mathsf{acq}(\mathsf{m})^j \xrightarrow{CP} \mathsf{rel}(\mathsf{m})^i$.[3] Line 4 checks this condition; if it is true, line 5 adds σ to $CP(\rho)$.

Algorithm 4 Pre-release of m by T

```
1:  for all ρ do
2:      ▷ Trigger CCP according to Rule (b); transfer CCP
3:      for all σ, j | σ : m^j ∈ CCP(ρ) do
4:          if ∃l ≥ j | T ∈ CP(m^l) then
5:              CP(ρ)^+ ← CP(ρ)^+ ∪ {σ}
6:          end if
7:          for all n^k | (∃l ≥ j | T : n^k ∈ CCP(m^l)) do
8:              CCP(ρ)^+ ← CCP(ρ)^+ ∪ {σ : n^k}
9:                      ▷ No effect if ∃k' < k | σ : n^k' ∈ CCP(ρ)^+
10:         end for
11:     end for
12: end for
```

Applying Rule (b) for CCP locksets. Analogous to lines 3–6's handling of CP locksets, lines 7–10 handle CCP locksets. For every $\sigma : \mathsf{m}^j \in CCP(\rho)$ (line 3) and $\mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{m}^j)$ (line 7),[4] σ is CP ordered to e_ρ if $\mathsf{acq}(\mathsf{m})^j \xrightarrow{CP} \mathsf{rel}(\mathsf{m})^i$, which in turn is true if $\mathsf{acq}(\mathsf{n})^k \xrightarrow{CP} \mathsf{rel}(\mathsf{n})^h$ (where n^h is the ongoing critical section on n). Thus, σ is CP ordered to e_ρ if $\mathsf{acq}(\mathsf{n})^k \xrightarrow{CP} \mathsf{rel}(\mathsf{n})^h$, so line 8 adds $\sigma : \mathsf{n}^k$ to $CCP(\rho)$.

We note that although the analysis could in theory detect the above CP and CCP orderings earlier than at $\mathsf{rel}(\mathsf{m})^i$, it is critical that the analysis detect them *no later than* $\mathsf{rel}(\mathsf{m})^i$, since the release algorithm removes all $\sigma : \mathsf{m}^j$ elements.

6.4.2 Release Algorithm (Algorithm 5)

Transferring ordering. HB and CP are both closed under right-composition with HB. Thus, after the current event $e = \mathsf{rel}(\mathsf{m})^i$ by T, if T was HB or CP ordered to e_ρ, m is now HB or CP ordered to e_ρ. Lines 3–12 show how the analysis transfers ordering from T to m. If $\mathsf{T} \in CP(\rho)$, then the analysis adds m to $CP(\rho)$ (lines 3–5). Similar logic transfers HB ordering from T to m (lines 9–12).

Algorithm 5 $\mathsf{rel}(\mathsf{m})^i$ by T

```
1:  for all ρ do
2:      ▷ Transfer ordering from T to m
3:      if T ∈ CP(ρ) then
4:          CP(ρ)^+ ← CP(ρ)^+ ∪ {m}
5:      end if
6:      for all n^k | n ≠ m ∧ T : n^k ∈ CCP(ρ) do
7:          CCP(ρ)^+ ← CCP(ρ)^+ ∪ {m : n^k}
8:      end for
9:      if T ∈ HB(ρ) then
10:         HB(ρ)^+ ← HB(ρ)^+ ∪ {m^i}
11:                 ▷ No effect if ∃i' < i | m^i' ∈ HB(ρ)^+
12:     end if
13:     ▷ Remove CCP elements conditional on m
14:     CCP(ρ)^+ ← CCP(ρ)^+ \ { σ : m^j ∈ CCP(ρ) }
15: end for
```

Lines 6–8 transfer CCP ordering. For any lock instance n^k ($\mathsf{n} \neq \mathsf{m}$) such that $\mathsf{T} : \mathsf{n}^k \in CCP(\rho)$, T is CP ordered to e_ρ *if* $\mathsf{acq}(\mathsf{n})^k \xrightarrow{CP} \mathsf{rel}(\mathsf{n})^j$, where n^j is n's ongoing critical section. Since CP right-composes with HB, m is CP ordered to e_ρ if $\mathsf{acq}(\mathsf{n})^k \xrightarrow{CP} \mathsf{rel}(\mathsf{n})^j$. Thus, the analysis adds $\mathsf{m} : \mathsf{n}^k$ to $CCP(\rho)$, satisfying the CP invariant (Figure 3).

Removing CCP ordering. Line 14 removes all CCP elements dependent on m, i.e., all $\sigma : \mathsf{m}^j$ elements from $CCP(\rho)$, satisfying the CCP-constraint invariant (Figure 3). Removing these elements is necessary: it would be incorrect for the analysis to retain these elements, e.g., $\mathsf{acq}(\mathsf{m})^j \xrightarrow{CP} \mathsf{rel}(\mathsf{m})^{i+1}$ does *not* imply that σ is CP ordered to e_ρ. As described above, before the release algorithm removes all CCP elements dependent on m, the pre-release algorithm detects CP and CCP orderings that are dependent on m.

6.5 Example

This section presents one example of how Raptor maintains its state during a program execution. For additional, more complex examples, please see Appendix A in the submitted supplementary material.

Figure 4 shows a simple execution and how Raptor maintains its analysis state. The last column shows the analysis state (for modified locksets only) *after* each event.

Up to $\mathsf{acq}(\mathsf{m})^2$, the analysis detects only PO and HB ordering. At $\mathsf{acq}(\mathsf{m})^2$, in addition to transferring HB ordering from m to $\mathsf{T2}$, the analysis adds $\mathsf{T2} : \mathsf{m}^1$ to all $CCP(\rho)$ such that $e_\rho \xrightarrow{HB} \mathsf{acq}(\mathsf{m})^2$.

At $\mathsf{wr}(\mathsf{x})^2$, it is not knowable from the events so far that $\mathsf{wr}(\mathsf{x})^1 \xrightarrow{CP} \mathsf{wr}(\mathsf{x})^2$. Raptor adds $\xi : \mathsf{m}^1$ to $CCP(\mathsf{x}^1)$, indicating that $\mathsf{rel}(\mathsf{m})^1 \xrightarrow{CP} \mathsf{acq}(\mathsf{m})^2$ implies $\mathsf{wr}(\mathsf{x})^1 \xrightarrow{CP} \mathsf{wr}(\mathsf{x})^2$. At $\mathsf{wr}(\mathsf{y})^2$, the analysis applies Rule (a) of the CP

[3] More precisely, the analysis checks for any $l \geq j \mid \mathsf{acq}(\mathsf{m})^l \xrightarrow{CP} \mathsf{rel}(\mathsf{m})^i$, since CP is closed under left-composition with HB.

[4] More precisely, the analysis checks for $\mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{m}^l)$ where $l \geq j$, since CP is closed under left-composition with HB.

T1	T2	Locksets after events
$\mathsf{wr(x)}^1$		$PO(\mathsf{x}^1) = \{\mathsf{T1}\}$, $HB(\mathsf{x}^1) = \{\mathsf{T1}\}$
$\mathsf{acq(m)}^1$		$PO(\mathsf{m}^1) = \{\mathsf{T1}\}$, $HB(\mathsf{m}^1) = \{\mathsf{T1}\}$
$\mathsf{wr(y)}^1$		$PO(\mathsf{y}^1) = \{\mathsf{T1}\}$, $HB(\mathsf{y}^1) = \{\mathsf{T1}, \mathsf{m}^1_*\}$
$\mathsf{rel(m)}^1$		$HB(\mathsf{x}^1) = HB(\mathsf{m}^1) = \{\mathsf{T1}, \mathsf{m}^1\}$
⇘	$\mathsf{acq(m)}^2$	$HB(\mathsf{x}^1) = HB(\mathsf{m}^1) = \{\mathsf{T1}, \mathsf{T2}, \mathsf{m}^1\}$
		$CCP(\mathsf{x}^1) = CCP(\mathsf{m}^1) = \{\mathsf{T2}:\mathsf{m}^1\}$
		$HB(\mathsf{y}^1) = \{\mathsf{T1}, \mathsf{T2}, \mathsf{m}^1_*\}$, $CCP(\mathsf{y}^1) = \{\mathsf{T2}:\mathsf{m}^1\}$
		$PO(\mathsf{m}^2) = \{\mathsf{T2}\}$, $HB(\mathsf{m}^2) = \{\mathsf{T2}\}$
	$\mathsf{wr(x)}^2$	$PO(\mathsf{x}^2) = \{\mathsf{T2}\}$, $HB(\mathsf{x}^2) = \{\mathsf{T2}, \mathsf{m}^2_*\}$
		$CCP(\mathsf{x}^1) = \{\mathsf{T2}:\mathsf{m}^1, \xi:\mathsf{m}^1\}$
		$HB(\mathsf{x}^1) = \{\mathsf{T1}, \mathsf{T2}, \mathsf{m}^1, \xi\}$
	$\mathsf{wr(y)}^2$	$CP(\mathsf{m}^1) = \{\mathsf{T2}\}$, $HB(\mathsf{y}^1) = \{\mathsf{T1}, \mathsf{T2}, \mathsf{m}^1_*, \xi\}$
		$PO(\mathsf{y}^2) = \{\mathsf{T2}\}$, $HB(\mathsf{y}^2) = \{\mathsf{T2}, \mathsf{m}^2_*\}$
		$CCP(\mathsf{y}^1) = \{\mathsf{T2}:\mathsf{m}^1, \xi:\mathsf{m}^1\}$
	$\mathsf{rel(m)}^2$	$CP(\mathsf{m}^1) = \{\mathsf{T2}, \mathsf{m}\}$, $HB(\mathsf{m}^2) = \{\mathsf{T2}, \mathsf{m}^2\}$
		$CP(\mathsf{x}^1) = CP(\mathsf{y}^1) = \{\mathsf{T2}, \mathsf{m}, \xi\}$

Figure 4. An execution with no CP-races. The last column shows the changes that Raptor makes to its analysis state after each event.

definition, adding T2 to $CP(\mathsf{m}^1)$. Although it is possible to detect that $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ and $\mathsf{wr(y)}^1 \xrightarrow{CP} \mathsf{wr(y)}^2$ at this point, the analysis defers this logic until $\mathsf{rel(m)}^2$.

At $\mathsf{rel(m)}^2$, the analysis applies Rule (b) of the CP definition, adding ξ to $CP(\mathsf{x}^1)$ and $CP(\mathsf{y}^1)$ because $\mathsf{T2} \in CP(\mathsf{m}^1)$ and $\xi:\mathsf{m}^1 \in CCP(\mathsf{x}^1)$ and $\xi:\mathsf{m}^1 \in CCP(\mathsf{y}^1)$.

7. Correctness

This section overviews our stategy of proving that Raptor soundly and completely tracks CP. The proof details are in the submitted supplementary material.

Theorem 1. *After every event, Raptor (i.e., the analysis in Algorithms 1–5) maintains the invariants in Figure 3.*

We prove this theorem in Appendix B in the submitted supplementary material. The proof is by induction on the total order of events. We only prove that the CP invariant holds, and argue that it is relatively straightforward to see that Raptor maintains the other invariants after each event.

Theorem 2. *An execution has a CP-race if and only if Raptor reports a race for the execution.*

Appendix B in the submitted supplementary material proves this theorem, which follows naturally from Theorem 1.

8. Removing Obsolete Locksets and Detecting CP-Races

Raptor maintains locksets for every variable access and lock acquire. This property makes the analysis state's size proportional to the execution's length, which is unscalable in terms of space as well as time, since the analysis sometimes iterates over all locksets.

Fortunately, for real (non-adversarial) program executions, most locksets become *obsolete*—meaning that they will not be needed again—relatively quickly. Raptor detects and removes obsolete locksets, saving both space and time.

A variable access x^i's locksets becomes obsolete once the analysis determines whether or not the corresponding access ($\mathsf{wr(x)}^i$) is involved in a CP-race with the next access ($\mathsf{wr(x)}^{i+1}$). Thus, detecting CP-races is naturally part of checking for obsolete locksets.

Removing obsolete variable locksets and detecting CP-races. Raptor uses x^i's locksets for two reasons: (1) to apply Rule (a) of the CP definition if $\mathsf{wr(x)}^i$ and $\mathsf{wr(x)}^j$ execute in critical sections on the same lock m and (2) to detect whether $\mathsf{wr(x)}^i \xrightarrow{CP} \mathsf{wr(x)}^{i+1}$. Algorithm 6 shows the conditions for determining whether $\mathsf{wr(x)}^i$ is obsolete or CP-races with $\mathsf{wr(x)}^{i+1}$. If Raptor has determined that $\mathsf{wr(x)}^i \xrightarrow{CP} \mathsf{wr(x)}^{i+1}$ or $\mathsf{wr(x)}^i \xrightarrow{PO} \mathsf{wr(x)}^{i+1}$, then according to the PO and CP invariants (Figure 3), $\xi \in CP(\mathsf{x}^i) \cup PO(\mathsf{x}^i)$; line 2 checks this condition. If true, x^i's locksets are obsolete: they will not be needed again, so the algorithm removes them. The algorithm denotes removal by setting x^i's locksets to $\emptyset$ (line 4)—essentially the same nonexistent state that x^i's locksets had *before* $\mathsf{wr(x)}^i$ executed.

Algorithm 6 Detect and remove obsolete locksets and report CP-races for x^i

```
1: if wr(x)^{i+1} has executed then
2:     if ξ ∈ CP(x^i) ∪ PO(x^i) then
3:         ▷ No CP-race between wr(x)^i and wr(x)^{i+1}
4:         PO(x^i)^+ ← ∅, HB(x^i)^+ ← ∅,
           CP(x^i)^+ ← ∅, CCP(x^i)^+ ← ∅
5:     else if ∄m^j | ξ:m^j ∈ CCP(x^i) then
6:         Report CP-race between wr(x)^i and wr(x)^{i+1}
7:     end if
8: end if
```

Raptor can say for certain that $\mathsf{wr(x)}^i \not\xrightarrow{CP} \mathsf{wr(x)}^{i+1}$ if $\xi \in CP(\mathsf{x}^i)$ is definitely impossible. According to Raptor's analysis (Algorithms 1–5), $\xi \in CP(\mathsf{x}^i)$ is possible only if $\exists \mathsf{m}^j \mid \xi:\mathsf{m}^j \in CCP(\mathsf{x}^i)$ (line 5). If not, the analysis knows $\mathsf{wr(x)}^i \not\xrightarrow{CP} \mathsf{wr(x)}^{i+1}$ and reports a CP-race.

When the execution terminates (i.e., after the last event in the observed total order), we assume that no thread holds any lock.[5] As a result, according to the CCP-constraint invariant (Figure 3), line 5 in Algorithm 6 evaluates to true for every x^i by the time the execution terminates. Thus for every $\mathsf{wr(x)}^i$–$\mathsf{wr(x)}^{i+1}$ pair for which Raptor has not already ruled out a CP-race (via line 2 evaluating to true), Raptor eventually reports a CP-race.

Removing obsolete lock locksets. Raptor's analysis uses locksets for lock instances, such as m^j for $\mathsf{acq(m)}^j$, to detect CP-ordered critical sections, in order to be able to apply Rule (b) of the CP definition. Once m^j's locksets' elements can no longer trigger Rule (b), m^j's locksets are obsolete.

[5] If an execution does not satisfy this condition, Raptor can simulate the release of all held locks, by performing the pre-release and release algorithms (Algorithms 4 and 5) for each held lock.

Algorithm 7 Detect and remove obsolete locksets for m^j

1: **if** $\mathsf{rel}(\mathsf{m})^j$ has executed **then**
2: **if** $\nexists \rho \mid \rho \neq \mathsf{m}^j \wedge \big(\mathsf{m}^j \in HB(\rho) \vee \mathsf{m}^j_* \in HB(\rho) \vee (\exists i \leq j \mid \sigma : \mathsf{m}^i \in CCP(\rho))\big) \wedge \mathsf{m} \notin CP(\rho)$ **then**
3: $PO(\mathsf{m}^j)^+ \leftarrow \emptyset$, $HB(\mathsf{m}^j)^+ \leftarrow \emptyset$, $CP(\mathsf{m}^j)^+ \leftarrow \emptyset$, $CCP(\mathsf{m}^j)^+ \leftarrow \emptyset$
4: **end if**
5: **end if**

Algorithm 7 shows the condition for m^j's locksets being obsolete. m^j's locksets may be needed again if, for any ρ, m^j appears in $HB(\rho)$ or an element dependent on m^i (such that $i \leq j$) appears in $CCP(\rho)$—unless $\mathsf{m} \in CP(\rho)$, in which case m^j is not needed by ρ's locksets. Line 2 shows the exact condition; if it evaluates to true, then the pre-release algorithm (Algorithm 4) definitely will not use m^j's locksets anymore, so the algorithm implicitly removes them by setting them to $\emptyset$.

9. Handling Read Events

So far we have presented Raptor as an analysis that handles only writes. However, in order to detect CP-races soundly and completely, Raptor must handle reads differently from writes, since two reads do not race with each other. This section overviews the changes to Raptor to handle reads correctly: representing locksets for read accesses; detecting write–read and read–write CP-races; and removing locksets in the presence of reads. These changes do not require additional ideas, but they add some complexity. Appendix C in the submitted supplementary material provides detailed algorithms that support reads as well as writes.

Detecting write–read races. An execution's writes are either totally ordered by $\xrightarrow{CP} \cup \xrightarrow{PO}$, or there is a CP-race. In contrast, reads are not totally ordered in a CP-race-free execution. For a write $\mathsf{wr}(\mathsf{x})^i$, Raptor must check if each thread's following read that comes before $\mathsf{wr}(\mathsf{x})^{i+1}$ is CP ordered with $\mathsf{wr}(\mathsf{x})^i$. Raptor extends the special element ξ to one special element per thread, ξ_T, which denotes that T's read of x is ordered with the the preceding write $\mathsf{wr}(\mathsf{x})^i$.

If a thread T performs multiple reads to x between $\mathsf{wr}(\mathsf{x})^i$ and $\mathsf{wr}(\mathsf{x})^{i+1}$, Raptor safely ignores checking for write–read CP-races: if a subsequent read is involved in a write–read CP-race, then the first read is involved in a write–read race.

Since a write may race with each thread's following read, x^i's locksets are not obsolete until Raptor determines that no thread's read between $\mathsf{wr}(\mathsf{x})^i$ and $\mathsf{wr}(\mathsf{x})^{i+1}$ is in a CP-race with $\mathsf{wr}(\mathsf{x})^i$. Thus, we extend the variable removal and CP-race check from Algorithm 6 to check for ξ_T for each thread T that read x between $\mathsf{wr}(\mathsf{x})^i$ and $\mathsf{wr}(\mathsf{x})^{i+1}$.

Detecting read–write races. For each thread that reads x between $\mathsf{wr}(\mathsf{x})^i$ and $\mathsf{wr}(\mathsf{x})^{i+1}$ in the total order, Raptor must check for a read–write CP-race. Raptor uses lockset owners of the form x^i_T for each thread T that reads x between $\mathsf{wr}(\mathsf{x})^i$ and $\mathsf{wr}(\mathsf{x})^{i+1}$. Raptor uses x^i_T's locksets to detect ordering from T's read of x to $\mathsf{wr}(\mathsf{x})^{i+1}$, in order to detect read–write races and to apply Rule (a) of the CP definition.

If a thread T performs multiple reads to x between $\mathsf{wr}(\mathsf{x})^i$ and $\mathsf{wr}(\mathsf{x})^{i+1}$, Raptor only needs to track the latest read: if an earlier read is involved in a read–write race, then the later read is involved in a read–write race. Raptor thus maintains x^i_T's locksets for the latest read by T only, resetting the locksets at each read by T prior to the next write ($\mathsf{wr}(\mathsf{x})^{i+1}$).

10. Evaluation

This section evaluates the performance and CP-race coverage of our implementation of the Raptor analysis.

10.1 Implementation

We have implemented the Raptor analysis, including support for reads, in *RoadRunner*, a dynamic analysis framework for concurrent programs [27]. RoadRunner instruments Java bytecode dynamically at class loading time, generating events of memory accesses (field and array element accesses) and synchronization events (e.g., lock acquire, release, wait, and resume; thread fork, start, terminate, and join; and volatile read and write).

Handling non-lock synchronization. The CP relation handles lock-based synchronization, but not other forms of synchronization such as volatile accesses and thread fork and join. Our implementation conservatively translates each volatile variable access so that a critical section on a lock unique to the variable surrounds the access, which creates CP ordering between accesses to the same volatile variable where at least one is a write. The implementation handles thread events similarly by generating small critical sections containing conflicting accesses, thus creating CP ordering for fork and join. Smaragdakis et al. translate these synchronization operations similarly before feeding them to their Datalog implementation [56].

The implementation similarly establishes HB edges from static class initializers to reads of initialized static fields [36].

Removing obsolete locksets and reporting CP-races. The implementation follows the logic from Algorithms 6 and 7, extended to handle read accesses (Section 9), in order to remove obsolete locksets and report CP-races. However, instead of executing these algorithms directly (e.g., periodically pass over all non-obsolete locksets), the implementation uses *reference counting* to identify obsolete locksets.

For each variable access (e.g., x^i or x^i_T), the implementation keeps track of the number of ξ and ξ_T elements that need to be added to $CP(\mathsf{x}^i) \cup PO(\mathsf{x}^i)$ (or $CP(\mathsf{x}^i_\mathsf{T}) \cup PO(\mathsf{x}^i_\mathsf{T})$) before the analysis can be certain that no CP-race exists. Once this number reaches zero, the implementation concludes that $\mathsf{wr}(\mathsf{x})^i$ (or $\mathsf{rd}(\mathsf{x})^i_\mathsf{T}$) does not race with a following element, and thus its locksets are obsolete.

The implementation tracks the numbers of remaining $\xi : \mathsf{m}^j$ and $\xi_\mathsf{T} : \mathsf{m}^j$ elements in $CCP(\mathsf{x}^i)$ (or $CCP(\mathsf{x}^i_\mathsf{T})$). If any of these numbers drop to zero, and the corresponding ξ or ξ_T

element is not in $CCP(\mathsf{x}^i)$ (or $CCP(\mathsf{x}^i_\mathsf{T})$), the implementation reports a CP-race.

Optimizations. Our prototype implementation of Raptor is largely unoptimized. While we think there is significant opportunity for optimization to reduce unnecessary or redundant work, we also believe that designing and implementing effective optimizations would be a major undertaking (e.g., designing the "right" fast-path lookups while avoiding huge memory overheads).

We have however implemented the following optimization out of necessity. Before the pre-release algorithm (Algorithm 4) iterates over all ρ with active locksets, it precomputes the following information once: (1) the maximum j such that $\exists l \geq j \mid \mathsf{T} \in CP(\mathsf{m}^l)$ and (2) for each j, the set of n^k such that $\exists l \geq j \mid \mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{m}^l)$. This precomputation enables quick lookups at lines 4–10, dramatically outperforming an unoptimized pre-release algorithm.

10.2 Methodology

We have evaluated Raptor on two sets of benchmarks:

- Benchmarks evaluated by Smaragdakis et al. [56] that we were able to obtain and get to run. Of their benchmarks that we do not include, all except StaticBucketMap execute fewer than 1,000 events.
- The DaCapo benchmarks [5], version 9.12 bach, with the small workload size. We exclude several programs that do not run out of the box with RoadRunner. We are working to get results for the other DaCapo programs, based on directions from the RoadRunner authors for running these programs with RoadRunner.

The experiments run on a system with 4 Intel Xeon E5-4620 8-core processors (32 cores total) running Linux 2.6.32.

Datalog CP implementation. We have extended our Raptor implementation to generate a trace of events in a format that can be processed by Smaragdakis et al.'s Datalog CP implementation [56], which they have made available to us. Our experiments run the Datalog CP implementation with a bounded window size of 500 events, which matches prior work's evaluation [56]. The Raptor and Datalog CP implementation analyze identical executions, since Raptor performs its analysis while also generating a trace of events for the Datalog CP implementation. As a further sanity check of Raptor, we have cross-referenced Raptor's reported HB races with RoadRunner's FastTrack implementation's results [25].

10.3 Coverage and Performance

This section considers two empirical questions. (1) How many CP-races does Raptor find in real program executions that HB detectors do not also find? (2) How many CP-races does Raptor find that the Datalog CP implementation cannot find due to its bounded analysis window?

Table 1 reports execution time and HB- and CP-race coverage of Raptor, and compares with Datalog's CP-race coverage. *Time* includes the cost for Raptor to perform its analysis (which dominates execution time) and also generate a trace of events for the Datalog CP implementation, but not the cost of executing Datalog CP. We execute each program until it either terminates normally or executes for two hours. *Events* is the count of events (writes, reads, acquires, and releases) processed by Raptor. Raptor reports significantly different event counts for some of the programs also evaluated by Smaragdakis et al. [56]; this discrepancy makes sense because, at least in several cases, we are using different workloads than the prior work. The Datalog CP implementation runs the execution traces generated by the same run of Raptor that performs Raptor's analysis.

			Raptor		Datalog
	Time	Events	HB-races	CP-races	CP-races
elevator	28 s	29,500	0	0	0
FTPServer	358 s	62,512	371 (28)	395 (22)	27 (6)
hedc	2.5 s	7,886	20 (4)	17 (1)	0
Jigsaw	1.0 h	246,418	5 (4)	0	0
philo	3.1 s	703	0	0	0
tsp	2.0 h	164,167	1 (1)	82 (2)	2 (1)
avrora	2.0 h	310,830	0	0	0
eclipse	2.0 h	320,510	2 (1)	0	0
jython	2.0 h	412,616	0	3 (1)	0
pmd	2.0 h	1,804,994	0	0	0
tomcat	65 s	86,936	0	0	0
xalan	2.0 h	2,638,597	17 (3)	30 (5)	11 (5)

Table 1. Dynamic execution characteristics and reported HB- and CP-races reported for the evaluated programs. *Events* is the number of events processed within the reported execution *Time*. For each HB- and CP-race column, the first number is dynamic races, and the second number (in parentheses) is statically unique races.

	Distance range for each static CP-race
FTPServer	8–1,570; 203–1,472; 253–1,426; 293–1,836; 349–852; 459–2,308; 463–2,309; 495–1,586; 499–2,175; 525–2,415; 510–1,601; 607–1,959; 613–2,505; 760–1,931; 807–2,921; 968–968; 1,023–1,929; 1,397–1,462; 1,453–1,783; 1,736–1,736; 2,063–2,379; 2,079–2,079
hedc	2,067–3,774
tsp	177–14,303; 939–939
jython	157,736–157,739
xalan	4–16; 15–32; 15–48; 18–52; 134–149

Table 2. Event distances for detected CP-races. For each static CP-race reported by Raptor, the table reports a range of event distances for all dynamic occurrences of the static CP-race.

The *Raptor* columns report the *HB-races* and *CP-races* reported by Raptor. For each kind of data race, the first number is dynamic races detected, and the second number (in parentheses) is *distinct static races*. A static race is an unordered pair of static source locations (each source location is a source method and bytecode index).

Every CP-race is also an HB-race. For each detected CP-race, Raptor reports it as "HB" if it is also a HB-race, and otherwise Raptor reports it as "CP." Raptor's *CP-races*

column reports only races reported as "CP," i.e., CP-races that are not also HB-races. The races reported by the two columns can have some overlap in static races, i.e., if the same static race manifests once as an HB-race and again as a (non-HB) CP-race within the same execution.

The *Datalog* column is dynamic and distinct static CP-races reported by the Datalog CP implementation. Datalog CP detects fewer CP-races than Raptor because of its analysis window of 500 events. We have verified that (1) Raptor detects all CP-races detected by Datalog CP, and (2) for all CP-races reported by Raptor that involve accesses separated by fewer than 500 events, Datalog also detects the CP-race.

Table 2 shows the "event distance" for the CP-races detected by Raptor. The event distance of a CP-race is the distance (in events) between two accesses in the observed total order of events. For each static race in the *Raptor CP-races* column in Table 1, Table 2 reports the range of event distances for all dynamic instances of that static race. The table shows that many detected dynamic CP-races have event distances over 500 events; Raptor finds these dynamic CP-races but Datalog CP does not. For some static CP-races, every dynamic occurrence has an event distance exceeding 500 events, so Datalog CP does not detect the static CP-race at all, corresponding to CP-races missed by Datalog CP in Table 1. (A few CP-races have event distances of just less than 500 events, but Datalog CP does not find them because it would need to see events outside of the window in order to determine that no CP ordering exists.)

In summary, Raptor handles execution traces of 100,000s–1,000,000s of events within two hours, and it finds CP-races that existing predictive analyses cannot find.

11. Related Work

To our knowledge, Raptor is the first online, sound (no false races) predictive data race detection analysis. Other predictive analyses are inherently offline and cannot handle execution traces in full. Non-predictive dynamic and static analyses are either unsound (reporting false races) or cannot detect races that did not manifest in the current execution.

11.1 Predictive Analyses

Existing predictive analyses are inherently *offline* because they need to be able to "look back" at the entire execution [16, 29, 30, 37, 51, 54, 56] (Section 3). Because they inherently compute over an entire execution trace, these predictive analyses do not scale well to even modestly sized traces; researchers have handled this issue by restricting analysis to bounded *windows* of consecutive events, e.g., windows of 500 events [56] or 10,000 events [29]. In contrast Raptor is *online* because it summarize an execution's behavior so far in the form of *analysis state*.

Huang et al., Serbanuta et al., and Huang and Rajagopalan increase data race coverage beyond CP by encoding control flow constraints and using an SMT solver [29, 30, 54]. These analyses are technically NP-complete, although in practice they can be solved in polynomial time. In any case, just as for the Datalog CP implementation [56], these analyses scale only to bounded, small windows of events.

11.2 Non-predictive Analyses

Exposing data races. Prior approaches try to expose more data races by exploring multiple thread interleaving schedules, typically systematically or based on heuristics for exposing new behaviors [12, 14, 24, 42, 53]. Somewhat similarly, prior dynamic analyses perturb interleavings by pausing threads, in an effort to expose data races [23, 32]. In contrast, Raptor detects data races that are possible in *other* executions from a *single* observed execution (i.e., from a single schedule). Raptor and these approaches are potentially complementary, since Raptor can detect more data races than non-predictive analysis for a given execution.

Static analysis. Static data race detection analysis can achieve completeness, detecting all true data races possible in all executions [22, 43, 44, 50, 60]. However, it is inherently unsound (reports false data races).

Dynamic analysis. Dynamic analyses that are not predictive inherently cannot detect data races in *other* executions without risking reporting false data races. Happens-before analyses [21, 25, 49] and other race detection approaches that avoid false races [4, 19, 58] cannot predict data races in *other* executions.

As Section 4 mentioned, Raptor's *HB* analysis is closely based on prior work's *Goldilocks* analysis [21]. Raptor extends the Goldilocks HB analysis substantially in order to track CP; in particular, Raptor's CCP locksets add significant complexity beyond Goldilocks.

Goldilocks and our Raptor analysis use per-variable "locksets" to track the HB relation (and in Raptor's case, the CP relation) soundly and completely [21]. In contrast, *lockset analysis* is a different kind of analysis that detects violations of a locking discipline in which two conflicting memory accesses always hold at least one lock in common [17, 18, 46, 47, 52, 59]. Lockset analysis is appealing because, like predictive data race detection, it can detect data races that are ordered by HB in the current execution but manifest in some other execution. However, lockset analysis is inherently unsound (reports false positives) since not all violations of a locking discipline are data races (e.g., accesses ordered by fork, join, or notify–wait synchronization). Prior work hybridizes happens-before and lockset analyses for performance or accuracy reasons [47, 49, 61], but it inherently cannot report data races that manifest only in other executions without also risking false positives.

Prioritizing data races. Prior work seeks to expose erroneous *behaviors* due to data races, in order to prioritize races that are demonstrably harmful [13, 15, 23, 26, 31, 45, 53]. However, as researchers have argued convincingly, *all* data races are problematic because they lead to ill-defined semantics [1, 7, 8, 10, 39, 55]. In any case, prioritizing data races

is complementary to our work, which tries to detect as many (true) data races as possible.

12. Conclusion

To our knowledge, Raptor is the first online, sound (no false races) predictive analysis for detecting data races. Raptor provably tracks the causally-precedes (CP) relation soundly and completely. An evaluation of our implementation of Raptor shows that it can analyze full program executions, in contrast with existing predictive analyses that analyze bounded execution windows. This advantage allows Raptor to find CP-races that existing predictive analyses cannot find.

Acknowledgments

We thank Steve Freund for help with using and modifying RoadRunner; Yannis Smaragdakis for help with Datalog CP and experimental infrastructure; and Yannis Smaragdakis, Jaeheon Yi, Swarnendu Biswas, and Man Cao for helpful discussions, suggestions, and other feedback.

References

[1] S. V. Adve and H.-J. Boehm. Memory Models: A Case for Rethinking Parallel Languages and Hardware. *CACM*, 53:90–101, 2010.

[2] S. V. Adve, M. D. Hill, B. P. Miller, and R. H. B. Netzer. Detecting Data Races on Weak Memory Systems. In *ISCA*, pages 234–243, 1991.

[3] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler. A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World. *CACM*, 53(2):66–75, 2010.

[4] S. Biswas, M. Zhang, M. D. Bond, and B. Lucia. Valor: Efficient, Software-Only Region Conflict Exceptions. In *OOPSLA*, pages 241–259, 2015.

[5] S. M. Blackburn, R. Garner, C. Hoffman, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo Benchmarks: Java Benchmarking Development and Analysis. In *OOPSLA*, pages 169–190, 2006.

[6] H.-J. Boehm. How to miscompile programs with "benign" data races. In *HotPar*, 2011.

[7] H.-J. Boehm. Position paper: Nondeterminism is Unavoidable, but Data Races are Pure Evil. In *RACES*, pages 9–14, 2012.

[8] H.-J. Boehm and S. V. Adve. Foundations of the C++ Concurrency Memory Model. In *PLDI*, pages 68–78, 2008.

[9] H.-J. Boehm and S. V. Adve. You Don't Know Jack about Shared Variables or Memory Models. *CACM*, 55(2):48–54, 2012.

[10] H.-J. Boehm and B. Demsky. Outlawing Ghosts: Avoiding Out-of-Thin-Air Results. In *MSPC*, pages 7:1–7:6, 2014.

[11] M. D. Bond, K. E. Coons, and K. S. McKinley. Pacer: Proportional Detection of Data Races. In *PLDI*, pages 255–268, 2010.

[12] S. Burckhardt, P. Kothari, M. Musuvathi, and S. Nagarakatte. A Randomized Scheduler with Probabilistic Guarantees of Finding Bugs. In *ASPLOS*, pages 167–178, 2010.

[13] J. Burnim, K. Sen, and C. Stergiou. Testing Concurrent Programs on Relaxed Memory Models. In *ISSTA*, pages 122–132, 2011.

[14] Y. Cai and L. Cao. Effective and Precise Dynamic Detection of Hidden Races for Java Programs. In *ESEC/FSE*, pages 450–461, 2015.

[15] M. Cao, J. Roemer, A. Sengupta, and M. D. Bond. Prescient Memory: Exposing Weak Memory Model Behavior by Looking into the Future. In *ISMM*, pages 99–110, 2016.

[16] F. Chen, T. F. Serbanuta, and G. Rosu. jPredictor: A Predictive Runtime Analysis Tool for Java. In *ICSE*, pages 221–230, 2008.

[17] J.-D. Choi, K. Lee, A. Loginov, R. O'Callahan, V. Sarkar, and M. Sridharan. Efficient and Precise Datarace Detection for Multithreaded Object-Oriented Programs. In *PLDI*, pages 258–269, 2002.

[18] A. Dinning and E. Schonberg. Detecting Access Anomalies in Programs with Critical Sections. In *PADD*, pages 85–96, 1991.

[19] L. Effinger-Dean, B. Lucia, L. Ceze, D. Grossman, and H.-J. Boehm. IFRit: Interference-Free Regions for Dynamic Data-Race Detection. In *OOPSLA*, pages 467–484, 2012.

[20] T. Elmas, S. Qadeer, and S. Tasiran. Precise Race Detection and Efficient Model Checking Using Locksets. Technical report, Microsoft Research, 2005.

[21] T. Elmas, S. Qadeer, and S. Tasiran. Goldilocks: A Race and Transaction-Aware Java Runtime. In *PLDI*, pages 245–255, 2007.

[22] D. Engler and K. Ashcraft. RacerX: Effective, Static Detection of Race Conditions and Deadlocks. In *SOSP*, pages 237–252, 2003.

[23] J. Erickson, M. Musuvathi, S. Burckhardt, and K. Olynyk. Effective Data-Race Detection for the Kernel. In *OSDI*, pages 1–16, 2010.

[24] M. Eslamimehr and J. Palsberg. Race Directed Scheduling of Concurrent Programs. In *PPoPP*, pages 301–314, 2014.

[25] C. Flanagan and S. N. Freund. FastTrack: Efficient and Precise Dynamic Race Detection. In *PLDI*, pages 121–133, 2009.

[26] C. Flanagan and S. N. Freund. Adversarial Memory for Detecting Destructive Races. In *PLDI*, pages 244–254, 2010.

[27] C. Flanagan and S. N. Freund. The RoadRunner Dynamic Analysis Framework for Concurrent Programs. In *PASTE*, pages 1–8, 2010.

[28] P. Godefroid and N. Nagappan. Concurrency at Microsoft – An Exploratory Survey. In EC^2, 2008.

[29] J. Huang, P. O. Meredith, and G. Rosu. Maximal Sound Predictive Race Detection with Control Flow Abstraction. In *PLDI*, pages 337–348, 2014.

[30] J. Huang and A. K. Rajagopalan. Precise and Maximal Race Detection from Incomplete Traces. In *OOPSLA*, pages 462–476, 2016.

[31] B. Kasikci, C. Zamfir, and G. Candea. Data Races vs. Data Race Bugs: Telling the Difference with Portend. In *ASPLOS*, pages 185–198, 2012.

[32] B. Kasikci, C. Zamfir, and G. Candea. RaceMob: Crowdsourced Data Race Detection. In *SOSP*, pages 406–422, 2013.

[33] B. Kasikci, C. Zamfir, and G. Candea. Automated Classification of Data Races Under Both Strong and Weak Memory Models. *TOPLAS*, 37(3):8:1–8:44, May 2015.

[34] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *CACM*, 21(7):558–565, 1978.

[35] N. G. Leveson and C. S. Turner. An Investigation of the Therac-25 Accidents. *IEEE Computer*, 26(7):18–41, 1993.

[36] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Prentice Hall PTR, 2nd edition, 1999.

[37] P. Liu, O. Tripp, and X. Zhang. IPA: Improving Predictive Analysis with Pointer Analysis. In *ISSTA*, pages 59–69, 2016.

[38] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from Mistakes: A Comprehensive Study on Real World Concurrency Bug Characteristics. In *ASPLOS*, pages 329–339, 2008.

[39] J. Manson, W. Pugh, and S. V. Adve. The Java Memory Model. In *POPL*, pages 378–391, 2005.

[40] D. Marino, M. Musuvathi, and S. Narayanasamy. LiteRace: Effective Sampling for Lightweight Data-Race Detection. In *PLDI*, pages 134–143, 2009.

[41] F. Mattern. Virtual Time and Global States of Distributed Systems. In *Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1988.

[42] M. Musuvathi and S. Qadeer. Iterative Context Bounding for Systematic Testing of Multithreaded Programs. In *PLDI*, pages 446–455, 2007.

[43] M. Naik and A. Aiken. Conditional Must Not Aliasing for Static Race Detection. In *POPL*, pages 327–338, 2007.

[44] M. Naik, A. Aiken, and J. Whaley. Effective Static Race Detection for Java. In *PLDI*, pages 308–319, 2006.

[45] S. Narayanasamy, Z. Wang, J. Tigani, A. Edwards, and B. Calder. Automatically Classifying Benign and Harmful Data Races Using Replay Analysis. In *PLDI*, pages 22–31, 2007.

[46] H. Nishiyama. Detecting Data Races using Dynamic Escape Analysis based on Read Barrier. In *VMRT*, pages 127–138, 2004.

[47] R. O'Callahan and J.-D. Choi. Hybrid Dynamic Data Race Detection. In *PPoPP*, pages 167–178, 2003.

[48] PCWorld. Nasdaq's facebook glitch came from race conditions, 2012. `http://www.pcworld.com/article/255911/nasdaqs_facebook_glitch_came_from_race_conditions.html`.

[49] E. Pozniansky and A. Schuster. MultiRace: Efficient On-the-Fly Data Race Detection in Multithreaded C++ Programs. *CCPE*, 19(3):327–340, 2007.

[50] P. Pratikakis, J. S. Foster, and M. Hicks. LOCKSMITH: Context-Sensitive Correlation Analysis for Race Detection. In *PLDI*, pages 320–331, 2006.

[51] M. Said, C. Wang, Z. Yang, and K. Sakallah. Generating Data Race Witnesses by an SMT-based Analysis. In *NFM*, pages 313–327, 2011.

[52] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson. Eraser: A Dynamic Data Race Detector for Multi-Threaded Programs. In *SOSP*, pages 27–37, 1997.

[53] K. Sen. Race Directed Random Testing of Concurrent Programs. In *PLDI*, pages 11–21, 2008.

[54] T. F. Şerbănuţă, F. Chen, and G. Roşu. Maximal Causal Models for Sequentially Consistent Systems. In S. Qadeer and S. Tasiran, editors, *RV*, pages 136–150, 2013.

[55] J. Ševčík and D. Aspinall. On Validity of Program Transformations in the Java Memory Model. In *ECOOP*, pages 27–51, 2008.

[56] Y. Smaragdakis, J. Evans, C. Sadowski, J. Yi, and C. Flanagan. Sound Predictive Race Detection in Polynomial Time. In *POPL*, pages 387–400, 2012.

[57] U.S.–Canada Power System Outage Task Force. Final Report on the August 14th Blackout in the United States and Canada. Technical report, Department of Energy, 2004.

[58] K. Veeraraghavan, P. M. Chen, J. Flinn, and S. Narayanasamy. Detecting and Surviving Data Races using Complementary Schedules. In *SOSP*, pages 369–384, 2011.

[59] C. von Praun and T. R. Gross. Object Race Detection. In *OOPSLA*, pages 70–82, 2001.

[60] J. W. Voung, R. Jhala, and S. Lerner. RELAY: Static Race Detection on Millions of Lines of Code. In *ESEC/FSE*, pages 205–214, 2007.

[61] Y. Yu, T. Rodeheffer, and W. Chen. RaceTrack: Efficient Detection of Data Race Conditions via Adaptive Tracking. In *SOSP*, pages 221–234, 2005.

A. Examples Executions Analyzed by Raptor

Example involving CP established before accesses execute. In Figure 5, $\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2$ through the following logic: $\mathsf{rel(n)}^1 \xrightarrow{CP} \mathsf{acq(n)}^2$ by Rule (a), which implies $\mathsf{acq(m)}^1 \xrightarrow{CP} \mathsf{rel(m)}^2$ by Rule (c) (since $\mathsf{acq(m)}^1 \xrightarrow{HB} \mathsf{rel(n)}^1$ and $\mathsf{acq(n)}^2 \xrightarrow{HB} \mathsf{rel(m)}^2$), which implies $\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2$ by Rule (b). Thus $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ by applying Rule (c) to $\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2$ (since $\mathsf{wr(x)}^1 \xrightarrow{HB} \mathsf{rel(m)}^1$ and $\mathsf{acq(m)}^2 \xrightarrow{HB} \mathsf{wr(x)}^2$).

Since the critical section on n^2 finishes before $\mathsf{wr(x)}^1$ executes, Raptor must keep track of the ordering on critical sections of n, m, and other locks—not just analysis state for accesses such as x. After $\mathsf{acq(q)}^2$, $\mathsf{T3} : \mathsf{q}^1 \in CCP(\mathsf{m}^1)$. The CCP ordering composes with HB, so that $\mathsf{T4} : \mathsf{n}^1 \in CCP(\mathsf{m}^1)$ before $\mathsf{rel(n)}^2$, and thus $\mathsf{T4} \in CP(\mathsf{m}^1)$ after $\mathsf{rel(n)}^2$. In a similar vein, Raptor adds $\mathsf{T4} : \mathsf{m}^1$ to $CCP(\mathsf{x}^1)$ at $\mathsf{acq(m)}^2$, which allows the analysis to add T4 to $CP(\mathsf{x}^1)$ at $\mathsf{rel(m)}^2$.

Example involving CP ordering established only after CP-ordered variable accesses have executed. In Figure 6,

T1	T2	T3	T4	Lockset after event
	$\mathsf{acq(m)}^1$			$HB(\mathsf{m}^1) = \{\mathsf{T2}\}$
	$\mathsf{acq(q)}^1$			$HB(\mathsf{q}^1) = \{\mathsf{T2}\}$
	$\mathsf{rel(q)}^1$			$HB(\mathsf{m}^1) = HB(\mathsf{q}^1) = \{\mathsf{q}^1, \mathsf{T2}\}$
		$\mathsf{acq(q)}^2$		$HB(\mathsf{m}^1) = HB(\mathsf{q}^1) = \{\mathsf{q}^1, \mathsf{T2}, \mathsf{T3}\}$
				$CCP(\mathsf{m}^1) = CCP(\mathsf{q}^1) = \{\mathsf{T3}:\mathsf{q}^1\}$
				$HB(\mathsf{q}^2) = \{\mathsf{T3}\}$
		$\mathsf{rel(q)}^2$		$HB(\mathsf{q}^2) = \{\mathsf{q}^2, \mathsf{T3}\}$
		$\mathsf{acq(n)}^1$		$HB(\mathsf{n}^1) = \{\mathsf{T3}\}$
		$\mathsf{wr(y)}^1$		$HB(\mathsf{y}^1) = \{\mathsf{n}^1_*, \mathsf{T3}\}$
		$\mathsf{rel(n)}^1$		$HB(\mathsf{q}^2) = \{\mathsf{q}^2, \mathsf{n}^1, \mathsf{T3}\}$, $HB(\mathsf{n}^1) = \{\mathsf{n}^1, \mathsf{T3}\}$
				$HB(\mathsf{m}^1) = HB(\mathsf{q}^1) = \{\mathsf{q}^1, \mathsf{n}^1, \mathsf{T2}, \mathsf{T3}\}$
			$\mathsf{acq(n)}^2$	$HB(\mathsf{y}^1) = \{\mathsf{n}^1_*, \mathsf{T3}, \mathsf{T4}\}$
				$CCP(\mathsf{m}^1) = CCP(\mathsf{q}^1) = CCP(\mathsf{q}^2) = CCP(\mathsf{n}^1) = CCP(\mathsf{y}^1) = \{\mathsf{T4}:\mathsf{n}^1\}$
				$HB(\mathsf{q}^2) = \{\mathsf{q}^2, \mathsf{n}^1, \mathsf{T3}, \mathsf{T4}\}$, $HB(\mathsf{n}^1) = \{\mathsf{n}^1, \mathsf{T3}, \mathsf{T4}\}$
				$HB(\mathsf{m}^1) = HB(\mathsf{q}^1) = \{\mathsf{q}^1, \mathsf{n}^1, \mathsf{T2}, \mathsf{T3}, \mathsf{T4}\}$, $HB(\mathsf{n}^2) = \{\mathsf{T4}\}$
			$\mathsf{wr(y)}^2$	$CP(\mathsf{n}^1) = \{\mathsf{T4}\}$, $CCP(\mathsf{y}^1) = \{\mathsf{T4}:\mathsf{n}^1, \xi:\mathsf{n}^1\}$
				$HB(\mathsf{y}^1) = \{\mathsf{n}^1_*, \mathsf{T3}, \mathsf{T4}, \xi\}$
				$HB(\mathsf{y}^2) = \{\mathsf{n}^2_*, \mathsf{T4}\}$
			$\mathsf{rel(n)}^2$	$HB(\mathsf{n}^2) = \{\mathsf{n}^2, \mathsf{T4}\}$, $CP(\mathsf{y}^1) = \{\mathsf{n}, \mathsf{T4}, \xi\}$
				$CP(\mathsf{m}^1) = CP(\mathsf{q}^1) = CP(\mathsf{q}^2) = CP(\mathsf{n}^1) = \{\mathsf{n}, \mathsf{T4}\}$
$\mathsf{wr(x)}^1$				$HB(\mathsf{x}^1) = \{\mathsf{T1}\}$
$\mathsf{acq(p)}^1$				$HB(\mathsf{p}^1) = \{\mathsf{T1}\}$
$\mathsf{rel(p)}^1$				$HB(\mathsf{p}^1) = \{\mathsf{p}^1, \mathsf{T1}\}$, $HB(\mathsf{x}^1) = \{\mathsf{p}^1, \mathsf{T1}\}$
	$\mathsf{acq(p)}^2$			$HB(\mathsf{p}^1) = HB(\mathsf{x}^1) = \{\mathsf{p}^1, \mathsf{T1}, \mathsf{T2}\}$
				$CCP(\mathsf{p}^1) = CCP(\mathsf{x}^1) = \{\mathsf{T2}:\mathsf{p}^1\}$
				$HB(\mathsf{p}^2) = \{\mathsf{T2}\}$
	$\mathsf{rel(p)}^2$			$HB(\mathsf{m}^1) = HB(\mathsf{q}^1) = \{\mathsf{q}^1, \mathsf{p}^2, \mathsf{T2}, \mathsf{T3}\}$
				$HB(\mathsf{p}^2) = \{\mathsf{p}^2, \mathsf{T2}\}$
	$\mathsf{rel(m)}^1$			$HB(\mathsf{p}^1) = \{\mathsf{p}^1, \mathsf{m}^1, \mathsf{T1}, \mathsf{T2}\}$
				$HB(\mathsf{m}^1) = HB(\mathsf{q}^1) = \{\mathsf{q}^1, \mathsf{p}^2, \mathsf{m}^1, \mathsf{T2}, \mathsf{T3}\}$
				$HB(\mathsf{p}^2) = \{\mathsf{p}^2, \mathsf{m}^1, \mathsf{T2}\}$, $HB(\mathsf{x}^1) = \{\mathsf{p}^1, \mathsf{m}^1, \mathsf{T1}, \mathsf{T2}\}$
			$\mathsf{acq(m)}^2$	$HB(\mathsf{p}^1) = HB(\mathsf{x}^1) = \{\mathsf{p}^1, \mathsf{m}^1, \mathsf{T1}, \mathsf{T2}, \mathsf{T4}\}$
				$HB(\mathsf{p}^2) = \{\mathsf{p}^2, \mathsf{m}^1, \mathsf{T2}, \mathsf{T4}\}$, $HB(\mathsf{m}^2) = \{\mathsf{T4}\}$
				$CCP(\mathsf{p}^1) = CCP(\mathsf{p}^2) = CCP(\mathsf{x}^1) = \{\mathsf{T4}:\mathsf{m}^1\}$
				$CCP(\mathsf{m}^1) = CCP(\mathsf{q}^1) = \{\mathsf{T4}:\mathsf{m}^1\}$
			$\mathsf{rel(m)}^2$	$HB(\mathsf{m}^2) = \{\mathsf{m}^2, \mathsf{T4}\}$, $HB(\mathsf{n}^2) = \{\mathsf{n}^2, \mathsf{m}^2, \mathsf{T4}\}$
				$CP(\mathsf{p}^1) = CP(\mathsf{p}^2) = CP(\mathsf{x}^1) = \{\mathsf{m}, \mathsf{T4}\}$
				$CP(\mathsf{m}^1) = CP(\mathsf{q}^1) = CP(\mathsf{q}^2) = CP(\mathsf{n}^1) = \{\mathsf{n}, \mathsf{m}, \mathsf{T4}\}$
				$CP(\mathsf{y}^1) = \{\mathsf{n}, \mathsf{m}, \mathsf{T4}, \xi\}$, $HB(\mathsf{y}^1) = \{\mathsf{n}^1_*, \mathsf{m}^2, \mathsf{T3}, \mathsf{T4}, \xi\}$
				$HB(\mathsf{y}^2) = \{\mathsf{n}^2_*, \mathsf{m}^2, \mathsf{T4}\}$, $HB(\mathsf{n}^1) = \{\mathsf{n}^1, \mathsf{m}^2, \mathsf{T3}, \mathsf{T4}\}$
			$\mathsf{wr(x)}^2$	$HB(\mathsf{x}^2) = \{\mathsf{T4}\}$
				$CP(\mathsf{x}^1) = \{\mathsf{m}, \mathsf{T4}, \xi\}$, $HB(\mathsf{x}^1) = \{\mathsf{p}^1, \mathsf{m}^1, \mathsf{T1}, \mathsf{T2}, \mathsf{T3}, \xi\ \}$

Figure 5. Example execution and Raptor's analysis state. $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ because $\mathsf{rel(n)}^1 \xrightarrow{CP} \mathsf{acq(n)}^2$, yet $\mathsf{wr(x)}^1$ executes only after $\mathsf{rel(n)}^1 \xrightarrow{CP} \mathsf{acq(n)}^2$ has been established and after $\mathsf{rel(n)}^2$ has executed.

$\mathsf{wr(z)}^1 \xrightarrow{CP} \mathsf{wr(z)}^2$ because $\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2$ (via Rule (c)), which is true because $\mathsf{acq(m)}^1 \xrightarrow{CP} \mathsf{rel(m)}^2$ (via Rule (b)), which is true by Rule (c) because $\mathsf{rel(n)}^1 \xrightarrow{CP} \mathsf{acq(n)}^2$ by Rule (a). However, these CP orderings have not yet been (and cannot be) determined at $\mathsf{wr(z)}^2$.

Raptor handles this situation as follows. At $\mathsf{wr(z)}^2$, Raptor adds $\xi : \mathsf{m}^1$ to $CCP(\mathsf{z}^1)$, which captures the fact that $\mathsf{wr(z)}^1 \xrightarrow{CP} \mathsf{wr(z)}^2$ if $\mathsf{acq(m)}^1 \xrightarrow{CP} \mathsf{rel(m)}^2$. When the execution finally executes $\mathsf{rel(m)}^2$, Raptor concludes from $\xi:\mathsf{m}^1 \in CCP(\mathsf{z}^1)$ and $\mathsf{T3} \in CP(\mathsf{m}^1)$ that it should add ξ to $CP(\mathsf{z}^1)$.

Example that is subtlely different from the prior example. Figure 7 differs from Figure 6 by changing the order in which T1 executes its critical sections on m and q. This subtle change means that $\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2$ and thus $\mathsf{wr(z)}^1 \xrightarrow{CP} \mathsf{wr(z)}^2$.

In Figure 7, after $\mathsf{rel(q)}^1$, $\mathsf{m}^1 \in HB(\mathsf{q}^1)$, instead of $\mathsf{q}^1 \in HB(\mathsf{m}^1)$ as in Figure 6. As a result, $\mathsf{T3} : \mathsf{n}^1 \notin CCP(\mathsf{m}^1)$ after $\mathsf{acq(n)}^2$. That is, Raptor correctly captures the fact that $\mathsf{acq(n)}^1 \xrightarrow{CP} \mathsf{rel(n)}^2$ does *not* imply $\mathsf{acq(m)}^1 \xrightarrow{CP} \mathsf{rel(m)}^2$ (if true).

Example involving a chain of CCP dependencies. In Figure 8, $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ through the following logic: $\mathsf{rel(q)}^1 \xrightarrow{CP} \mathsf{acq(q)}^2$ by Rule (a), which implies $\mathsf{acq(o)}^1 \xrightarrow{CP} \mathsf{rel(o)}^2$ by Rule (c), which implies $\mathsf{rel(o)}^1 \xrightarrow{CP} \mathsf{acq(o)}^2$ by Rule (b), which implies $\mathsf{acq(m)}^1 \xrightarrow{CP} \mathsf{rel(m)}^2$ by Rule (c), which implies $\mathsf{rel(m)}^1 \xrightarrow{CP} \mathsf{acq(m)}^2$ by Rule (b), which

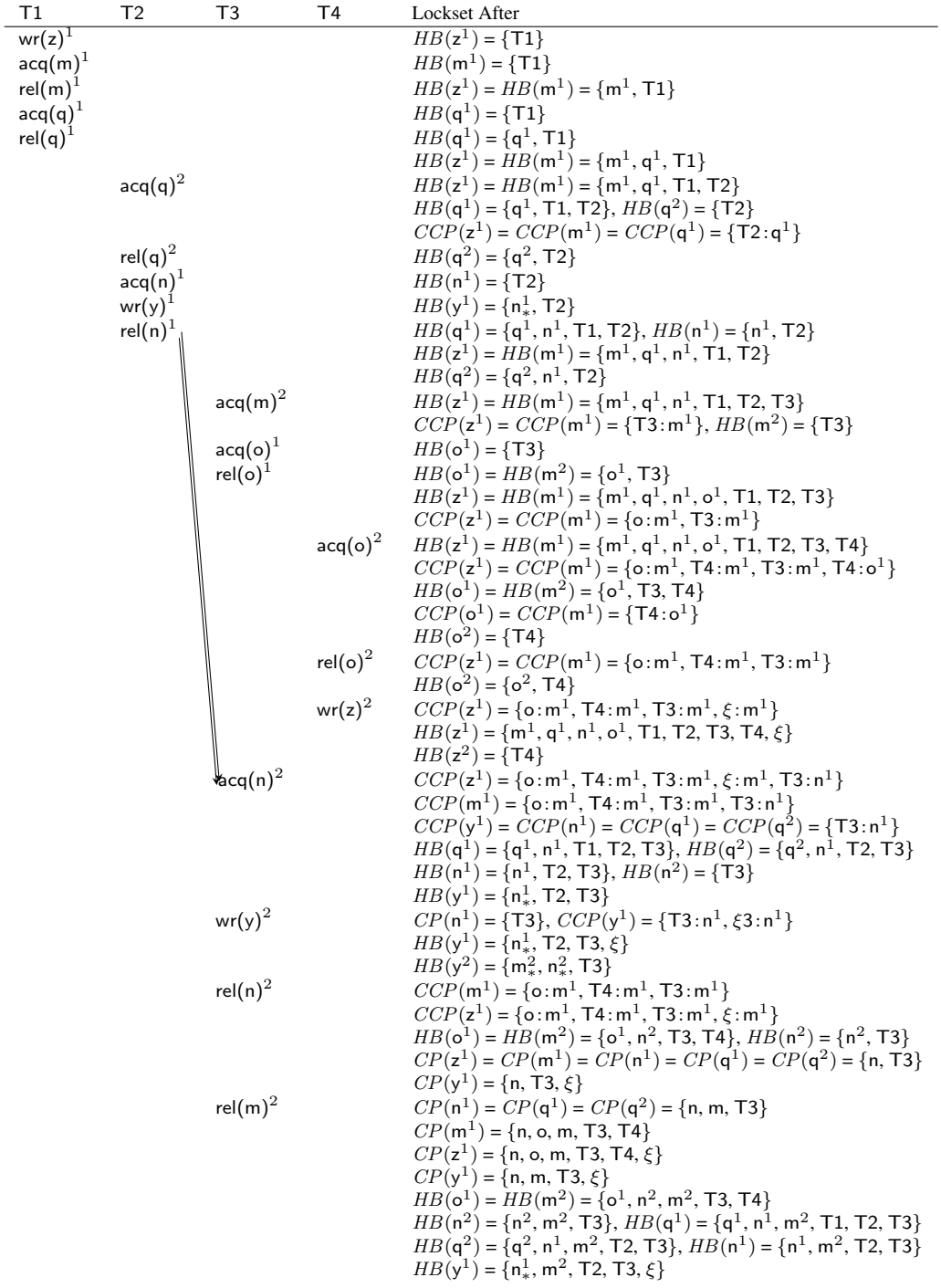

Figure 6. In this execution, $\mathsf{wr(z)}^1 \xrightarrow{CP} \mathsf{wr(z)}^2$ because of a CP ordering that has not yet been established when $\mathsf{wr(z)}^2$ executes.

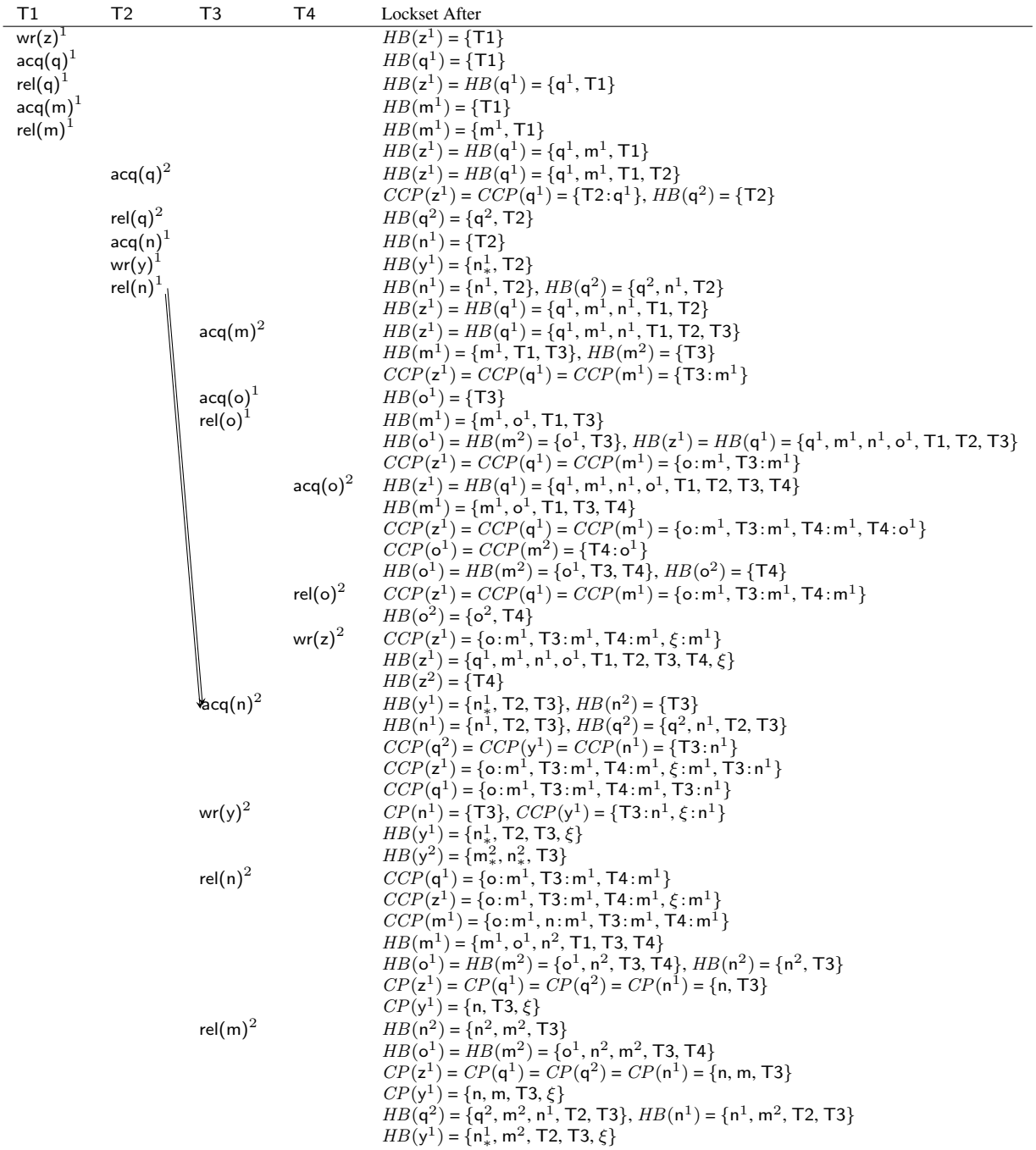

Figure 7. This example is subtlely different from the example in Figure 6: it flips the order of T1's critical sections on m and q. As a result, $\mathsf{acq(m)}^1 \overset{CP}{\not\longrightarrow} \mathsf{rel(m)}^2$.

implies $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ by Rule (c). The example shows that even when the critical section on m^2 ends, i.e., after $\mathsf{rel(m)}^2$ executes, it is not yet possible to determine that $\mathsf{acq(m)}^1 \xrightarrow{CP} \mathsf{rel(m)}^2$. This fact is not knowable until after $\mathsf{acq(q)}^2$ executes. In particular, at $\mathsf{rel(m)}^2$, $\mathsf{acq(m)}^1 \xrightarrow{CP} \mathsf{rel(m)}^2$ is dependent on whether $\mathsf{rel(o)}^1 \xrightarrow{CP} \mathsf{acq(o)}^2$, as represented by $\mathsf{T3{:}o}^1 \in CCP(\mathsf{m}^1)$ after $\mathsf{rel(m)}^2$.

B. Proof Details

This section proves the theorems from Section 7.

Theorem 1. *After every event, Raptor (i.e., the analysis in Algorithms 1–5) maintains the invariants in Figure 3.*

To prove this theorem, we use the following lemmas:

Lemma 1. *Let e be any* write *event, i.e., $e = \mathit{wr(x)}^i$ by thread T. Let e^+ be the event immediately after e in the observed total order ($\prec_{trace}$). If Figure 3's invariants hold for e before Raptor's write algorithm (Algorithm 2) executes, then they hold for e^+ after the write algorithm executes.*

Lemma 2. *Let e be any* acquire *event, i.e., $e = \mathit{acq(m)}^i$ by thread T. Let e^+ be the event immediately after e in the observed total order ($\prec_{trace}$). If Figure 3's invariants hold for e before Raptor's acquire algorithm (Algorithm 3) executes, then they hold for e^+ after the acquire algorithm executes.*

Lemma 3. *Let e be any* release *event, i.e., $e = \mathit{rel(m)}^i$ by thread T. If Figure 3's invariants hold for e before Raptor's pre-release algorithm (Algorithm 4) executes, then they hold for e after the pre-release algorithm executes.*

Lemma 4. *Let e be any* release *event, i.e., $e = \mathit{rel(m)}^i$ by thread T. Let e^+ be the event immediately after e in the observed total order ($\prec_{trace}$). If Figure 3's invariants hold for e before Raptor's release algorithm (Algorithm 5) executes, then they hold for e^+ after the release algorithm executes.*

We introduce four lemmas, instead of one combined lemma, in order to divide the proof steps into four separate proofs.

We prove these lemmas *for the CP invariant only*. For the other invariants, it is comparatively straightforward to see that they hold:

- The PO, HB, HB-index, and HB-critical-section invariants hold for a modest extension of the Goldilocks algorithm [21]. The Goldilocks authors have proved that the Goldilocks algorithm soundly and precisely computes HB [20].
- The analysis maintains the CP-rule-A invariant by updating CP lockset(s) as soon as a conflicting write executes (lines 2–7 of Algorithm 2).
- The CCP-constraint invariant holds because CCP elements of the form $\sigma : \mathsf{m}^j$ only exist during critical sections on m:
  1. At $\mathsf{rel(m)}^i$, Algorithm 5 removes all elements $\sigma : \mathsf{m}^j$ from every $CCP(\rho)^+$.
  2. When $\nexists \sigma' \mid \sigma' : \mathsf{m}^j \in CCP(\rho)$, only Algorithm 3 (for $\mathsf{acq(m)}^i$) adds elements of the form $\sigma : \mathsf{m}^j$ to $CCP(\rho)^+$.

Proof of Lemma 1.

Let $e = \mathsf{wr(x)}^i$ by thread T. Let e^+ be the event immediately after e in the observed total order ($\prec_{trace}$). Let ρ be any lockset owner. Let e_ρ be the event corresponding to ρ, i.e., $e_\rho = \mathsf{wr(x)}^h$ if $\rho = \mathsf{x}^h$, or $e_\rho = \mathsf{acq(m)}^h$ if $\rho = \mathsf{m}^h$.

We define the following abbreviations for the left- and right-hand sides of the CP invariant:

Let $LHS = CP(\rho) \cup \{\, \sigma \mid (\exists \mathsf{n}^k \mid \sigma{:}\mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e) \,\}$.

Let $LHS^+ = CP(\rho)^+ \cup \{\, \sigma \mid (\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+) \,\}$.

Let $RHS = \{\, \sigma \mid (\exists e' \mid \mathit{appl}(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e) \,\}$

Let $RHS^+ = \{\, \sigma \mid (\exists e' \mid \mathit{appl}(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e^+) \,\}$.

Suppose Figure 3's invariants hold before e, i.e., suppose $LHS = RHS$. We call this assumption the "inductive hypothesis" because we use this lemma's result in the proof of Theorem 1, which is by induction.

To show $LHS^+ = RHS^+$, we first show $LHS^+ \subseteq RHS^+$ (subset direction) and then $LHS^+ \supseteq RHS^+$ (superset direction).

Subset direction: Let $\sigma \in LHS^+$.

Either $\sigma \in CP(\rho)^+$ or $\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$ (or both):

Case 1: $\sigma \in CP(\rho)^+$

If $\sigma \in CP(\rho)$, then by the inductive hypothesis, $\sigma \in RHS$, i.e., $\exists e' \mid \mathit{appl}(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. Since $e \prec_{trace} e^+$, $\exists e' \mid e_\rho \xrightarrow{CP} e' \prec_{trace} e^+$, so $\sigma \in RHS^+$.

Otherwise ($\sigma \notin CP(\rho)$), Algorithm 2 must add σ to $CP(\rho)^+$, which can happen only at line 5 or 16:

- Line 5 executes within line 2's **for** loop, so let $\mathsf{m} \in \mathit{heldBy}(\mathsf{T})$. Since line 5 executes, line 3 evaluates to true, so (1) $\mathsf{T} \notin PO(\mathsf{x}^{i-1})$, and (2) let j be such that $\mathsf{m}^j_* \in HB(\mathsf{x}^{i-1})$. Line 5 adds T to $CP(\mathsf{m}^j)^+$, so $\rho = \mathsf{m}^j$ and $\sigma = \mathsf{T}$.

  By the inductive hypothesis on $\mathsf{m}^j_* \in HB(\mathsf{x}^{i-1})$ (HB-critical-section invariant), $\mathsf{acq(m)}^j \xrightarrow{PO} \mathsf{wr(x)}^{i-1} \xrightarrow{PO} \mathsf{rel(m)}^j \wedge \mathsf{wr(x)}^{i-1} \prec_{trace} e$. Because $\mathsf{m} \in \mathit{heldBy}(\mathsf{T})$ and $\mathsf{T} \notin PO(\mathsf{x}^{i-1})$, let $k > j$ be such that $\mathsf{acq(m)}^k \xrightarrow{PO} e \xrightarrow{PO} \mathsf{rel(m)}^k$. By the definition of CP (Rule (a)), $\mathsf{rel(m)}^j \xrightarrow{CP} \mathsf{acq(m)}^k$. By the definition of CP

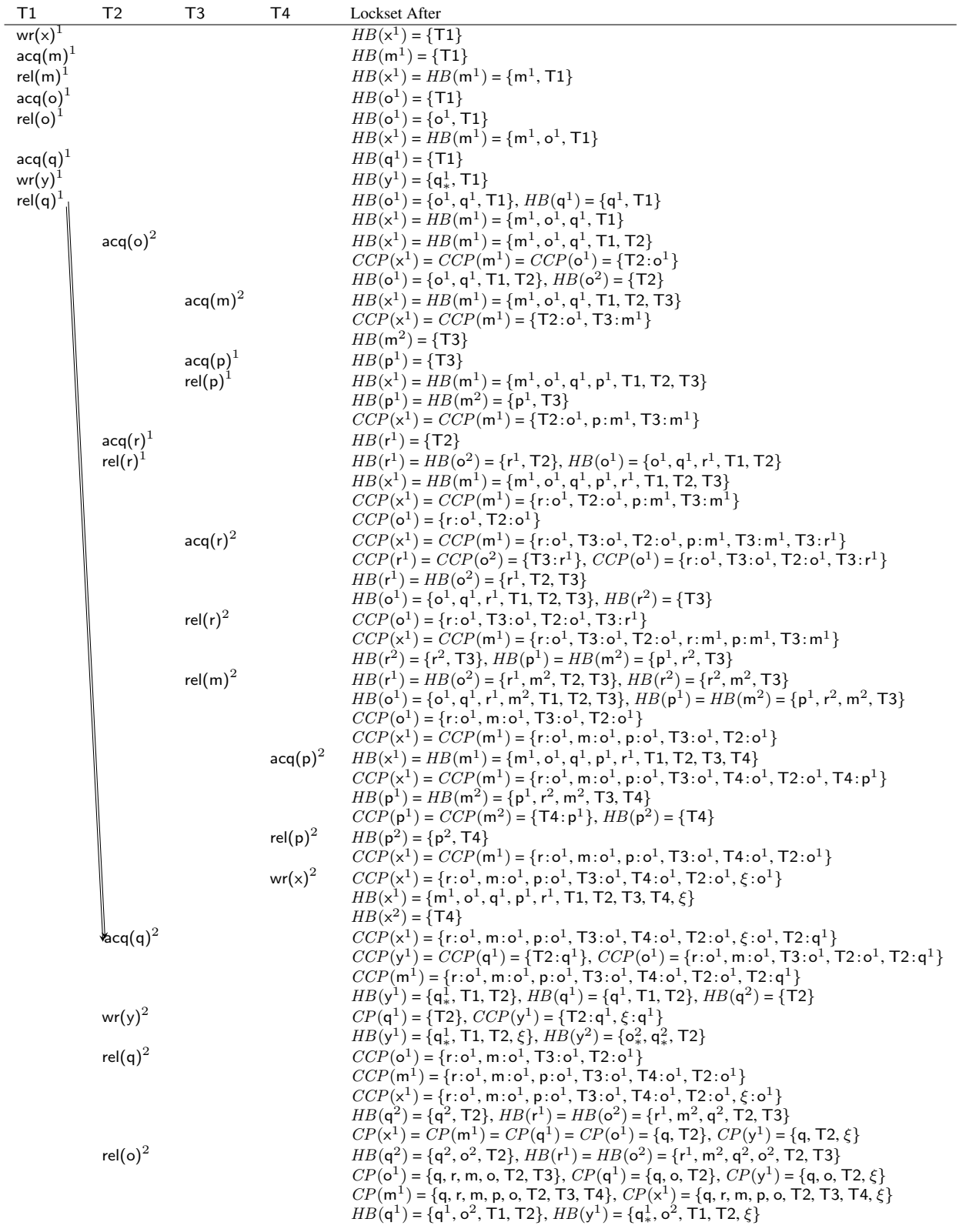

Figure 8. $\mathsf{wr(x)}^1 \xrightarrow{CP} \mathsf{wr(x)}^2$ due to CCP dependencies that Raptor necessarily resolves lazily.

(Rule (c)), $\mathsf{acq(m)}^j \xrightarrow{CP} e$. Since $thr(e) = \mathsf{T} \wedge \mathsf{acq(m)}^j \xrightarrow{CP} e \prec_{trace} e^+$, therefore $\mathsf{T} \in RHS^+$.

- Line 16 adds ξ to $CP(\rho)^+$, $\rho = \mathsf{x}^{i-1}$, $e_\rho = \mathsf{wr(x)}^{i-1}$ and $\sigma = \xi$. Since line 16 executes, line 15 evaluates to true, so $\mathsf{T} \in CP(\rho)$. By the inductive hypothesis (CP invariant), let e' be such that $thr(e') = \mathsf{T} \wedge \mathsf{wr(x)}^{i-1} \xrightarrow{CP} e' \prec_{trace} e$. By the definition of CP (Rule (c)), since $thr(e') = \mathsf{T} = thr(e)$, $\mathsf{wr(x)}^{i-1} \xrightarrow{CP} e \prec_{trace} e^+$. Note that $e = \mathsf{wr(x)}^i$ is e^* from Figure 3. Therefore $\xi \in RHS^+$.

Case 2: Let n^k and j be such that $\sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$.

If $\sigma : \mathsf{n}^k \in CCP(\rho) \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$, then by the inductive hypothesis (CP invariant), $\sigma \in RHS$, i.e., $\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. Since $e \prec_{trace} e^+$, $e_\rho \xrightarrow{CP} e' \prec_{trace} e^+$, and thus $\sigma \in RHS^+$.

Otherwise, $\neg\big(\sigma : \mathsf{n}^k \in CCP(\rho) \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e\big)$. However, $\mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$ because $\mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$ and $e \neq \mathsf{acq(n)}^j$. Therefore, $\sigma : \mathsf{n}^k \notin CCP(\rho)$, which means that Algorithm 2 must add $\sigma : \mathsf{n}^k$ to $CCP(\rho)^+$. This addition can happen only at line 19, which adds $\xi : \mathsf{n}^k$ to $CCP(\rho)^+$, so $\rho = \mathsf{x}^{i-1}$, $e_\rho = \mathsf{wr(x)}^{i-1}$, and $\sigma = \xi$. Because line 19 executes, $\mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{x}^{i-1})$ according to line 18. Since $\mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e \wedge \mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{x}^{i-1})$, by the inductive hypothesis (CP invariant), let e' be such that $thr(e') = \mathsf{T} \wedge \mathsf{wr(x)}^{i-1} \xrightarrow{CP} e' \prec_{trace} e$. By the definitions of HB and CP, since $thr(e') = \mathsf{T} = thr(e)$, $\mathsf{wr(x)}^{i-1} \xrightarrow{CP} e \prec_{trace} e^+$. Note that $e = \mathsf{wr(x)}^i$ is e^* from Figure 3. Therefore $\xi \in RHS^+$.

Superset direction: Let $\sigma \in RHS^+$.

If $\sigma \in RHS$, then by the inductive hypothesis, $\sigma \in LHS$. Since Algorithm 2 maintains $CP(\rho)^+ \supseteq CP(\rho)$ and $CCP(\rho)^+ \supseteq CCP(\rho)$, $\sigma \in LHS^+$.

Otherwise ($\sigma \in RHS^+$ but $\sigma \notin RHS$), $\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e^+$ but $\nexists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$, which implies $appl(\sigma, e) \wedge e_\rho \xrightarrow{CP} e$. By the definition of CP, since $e = \mathsf{wr(x)}^i$ is not an acquire operation, $\exists e' \mid thr(e') = \mathsf{T} \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$, i.e., $\mathsf{T} \in RHS$. According to the definition of $appl(\sigma, e)$, either (1) $\sigma = \mathsf{T}$ or (2) $\sigma = \xi$ (note that $e = \mathsf{wr(x)}^i$ is e^* from Figure 3). However, $\sigma \notin RHS$, so $\sigma = \xi$. Therefore $\rho = \mathsf{x}^{i-1}$ and $e_\rho = \mathsf{wr(x)}^{i-1}$, and $\mathsf{wr(x)}^{i-1} \xrightarrow{CP} e$. By the inductive hypothesis on $\mathsf{T} \in RHS$ (CP invariant), $\mathsf{T} \in CP(\mathsf{x}^{i-1})$ or $\exists \mathsf{n}^k \mid \mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{x}^{i-1}) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$ (or both).

Case 1: $\mathsf{T} \in CP(\mathsf{x}^{i-1})$

Then line 15 in Algorithm 2 evaluates to true, and line 16 adds ξ to $CP(\mathsf{x}^{i-1})^+$. Thus $\xi \in LHS^+$.

Case 2: Let n^k be such that $\mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{x}^{i-1}) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$.

Since $\mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{x}^{i-1})$ matches line 18, line 19 adds $\xi : \mathsf{n}^j$ to $CCP(\mathsf{x}^{i-1})^+$. Since $e \prec_{trace} e^+$, $\exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$. Thus $\xi \in LHS^+$.

Since $LHS^+ \subseteq RHS^+ \wedge LHS^+ \supseteq RHS^+$, $LHS^+ = RHS^+$. □

Proof of Lemma 2.

Let $e = \mathsf{acq(m)}^i$ by thread T. Let e^+ be the event immediately after e in the observed total order ($\prec_{trace}$). Let ρ be any lockset owner. Let e_ρ be the event corresponding to ρ, i.e., $e_\rho = \mathsf{wr(x)}^h$ if $\rho = \mathsf{x}^h$, or $e_\rho = \mathsf{acq(m)}^h$ if $\rho = \mathsf{m}^h$.

We define the following abbreviations for the left- and right-hand sides of the CP invariant:

Let $LHS = CP(\rho) \cup \big\{ \sigma \mid \big(\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e\big) \big\}$.

Let $LHS^+ = CP(\rho)^+ \cup \big\{ \sigma \mid \big(\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+\big) \big\}$.

Let $RHS = \big\{ \sigma \mid \big(\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e\big) \big\}$

Let $RHS^+ = \big\{ \sigma \mid \big(\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e^+\big) \big\}$.

Suppose Figure 3's invariants hold before e, i.e., suppose $LHS = RHS$. We call this assumption the "inductive hypothesis" because we use this lemma's result in the proof of Theorem 1, which is by induction.

To show $LHS^+ = RHS^+$, we first show $LHS^+ \subseteq RHS^+$ (subset direction) and then $LHS^+ \supseteq RHS^+$ (superset direction).

Subset direction: Let $\sigma \in LHS^+$.

Either $\sigma \in CP(\rho)^+$ or $\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$ (or both):

Case 1: $\sigma \in CP(\rho)^+$

If $\sigma \in CP(\rho)$, then by the inductive hypothesis, $\sigma \in RHS$, i.e., $\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. Since $e \prec_{trace} e^+$ then $\exists e' \mid e_\rho \xrightarrow{CP} e' \prec_{trace} e^+$, so $\sigma \in RHS^+$.

Otherwise ($\sigma \notin CP(\rho)$), Algorithm 3 must add σ to $CP(\rho)^+$, which can happen only at line 4. Line 4 adds T to $CP(\rho)^+$, so $\sigma = \mathsf{T}$. Since line 4 executes, line 3 must evaluate to true for ρ, so $\mathsf{m} \in CP(\rho)$. By the inductive hypothesis (CP invariant), $\exists j \mid e_\rho \xrightarrow{CP} \mathsf{rel(m)}^j \prec_{trace} e$. By the definition of CP, $e_\rho \xrightarrow{CP} e$ (since $e = \mathsf{acq(m)}^i$). Because $thr(e) = \mathsf{T} \wedge e_\rho \xrightarrow{CP} e \prec_{trace} e^+$, therefore $\mathsf{T} \in RHS^+$.

Case 2: Let n^k and j be such that $\sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$.

If $\sigma : \mathsf{n}^k \in CCP(\rho) \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$, then by the inductive hypothesis, $\sigma \in RHS$, i.e., $\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. Since $e \prec_{trace} e^+$ then $e_\rho \xrightarrow{CP} e' \prec_{trace} e^+$, so $\sigma \in RHS^+$.

Otherwise, $\neg\big(\sigma : \mathsf{n}^k \in CCP(\rho) \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e\big)$. In fact, we can conclude that $\sigma : \mathsf{n}^k \notin CCP(\rho)$ using the following reasoning. If $\neg\big(\mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e\big)$, then $\mathsf{acq(n)}^j \not\prec_{trace} e$ (since $\mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j$), and thus $e = \mathsf{acq(n)}^j$ (since $\mathsf{acq(n)}^j \prec_{trace} e^+$ and e immediately precedes e^+), which by the inductive hypothesis (CCP-constraint invariant) gives $\sigma : \mathsf{n}^k \notin CCP(\rho)$. Since $\sigma : \mathsf{n}^k \notin CCP(\rho)$ regardless, Algorithm 3 must add $\sigma : \mathsf{n}^k$ to $CCP(\rho)^+$, which can happen only at line 7 or 13 (or both).

- If line 7 adds $\sigma : \mathsf{n}^k$ to $CCP(\rho)^+$, then $\sigma = \mathsf{T}$ and $\mathsf{m} : \mathsf{n}^k \in CCP(\rho)$ (line 6). By the inductive hypothesis (CP invariant), $\exists l \mid e_\rho \xrightarrow{CP} \mathsf{rel(m)}^l \prec_{trace} e$. By the definition of CP, $e_\rho \xrightarrow{CP} e$ since $e = \mathsf{acq(m)}^i$. Since $thr(e) = \mathsf{T} \wedge e_\rho \xrightarrow{CP} e \prec_{trace} e^+$, therefore $\mathsf{T} \in RHS^+$.
- If line 13 adds $\sigma : \mathsf{n}^k$ to $CCP(\rho)^+$, then $\mathsf{n}^k \in HB(\rho)$ or $\mathsf{n}^k_* \in HB(\rho)$ according to line 10. Furthermore, $\mathsf{n} = \mathsf{m}$ and $\sigma = \mathsf{T}$. By the inductive hypothesis (HB, HB-index, and HB-critical-section invariants), $e_\rho \xrightarrow{HB} \mathsf{rel(n)}^k \prec_{trace} e$. Since we know $\mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$ (from the beginning of Case 2) and $e = \mathsf{acq(n)}^i$ (since $\mathsf{n} = \mathsf{m}$), therefore $e_\rho \xrightarrow{CP} e$ by the definition of CP (regardless of whether $j = i$ or $j < i$). Since $thr(e) = \mathsf{T} \wedge e_\rho \xrightarrow{CP} e \prec_{trace} e^+$, therefore $\mathsf{T} \in RHS^+$.

Superset direction: Let $\sigma \in RHS^+$.

If $\sigma \in RHS$, then by the inductive hypothesis, $\sigma \in LHS$. Since Algorithm 3 maintains $CP(\rho)^+ \supseteq CP(\rho)$ and $CCP(\rho)^+ \supseteq CCP(\rho)$, $\sigma \in LHS^+$.

Otherwise ($\sigma \in RHS^+$ but $\sigma \notin RHS$), $\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e^+$ but $\nexists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$, which implies $appl(\sigma, e) \wedge e_\rho \xrightarrow{CP} e$ and thus $\sigma = \mathsf{T}$. Since $e_\rho \xrightarrow{CP} e$ and $\nexists e' \mid thr(e') = \mathsf{T} \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$, by the definition of CP, $\exists l \mid e_\rho \xrightarrow{CP} \mathsf{rel(m)}^l \xrightarrow{HB} e$ or $\exists l \mid e_\rho \xrightarrow{HB} \mathsf{rel(m)}^l \xrightarrow{CP} e$ (or both).

Case 1: $\exists l \mid e_\rho \xrightarrow{CP} \mathsf{rel(m)}^l \xrightarrow{HB} e$

By the inductive hypothesis on $e_\rho \xrightarrow{CP} \mathsf{rel(m)}^l$ (CP invariant), $\mathsf{m} \in CP(\rho)$ or $\exists \mathsf{n}^k \mid \mathsf{m} : \mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$ (or both).

Case 1a: $\mathsf{m} \in CP(\rho)$

Then line 3 of Algorithm 3 evaluates to true for ρ, so line 4 executes and thus $\mathsf{T} \in CP(\rho)^+$. Therefore $\mathsf{T} \in LHS^+$.

Case 1b: Let n^k be such that $\mathsf{m} : \mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$.

Then according to line 6, line 7 executes and adds $\mathsf{T} : \mathsf{n}^k$ to $CCP(\rho)^+$. Since $e \prec_{trace} e^+$, $\exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$. Therefore $\mathsf{T} \in LHS^+$.

Case 2: $\exists l \mid e_\rho \xrightarrow{HB} \mathsf{rel(m)}^l \xrightarrow{CP} e$

Let j be the minimum value such that $e_\rho \xrightarrow{HB} \mathsf{rel(m)}^j$. By the inductive hypothesis (HB, HB-index, and HB-critical-section invariants), $\mathsf{m}^j \in HB(\rho)$ or $\mathsf{m}^j_* \in HB(\rho)$. Thus line 10 evaluates to true for ρ, so line 13 executes and $\mathsf{T} : \mathsf{m}^j \in CCP(\rho)^+$. Furthermore, $\mathsf{rel(m)}^j \xrightarrow{CP} \mathsf{acq(m)}^i \prec_{trace} e^+$ ($e = \mathsf{acq(m)}^i$). Therefore $\mathsf{T} \in LHS^+$.

Since $LHS^+ \subseteq RHS^+ \wedge LHS^+ \supseteq RHS^+$, $LHS^+ = RHS^+$. □

Proof of Lemma 3.

Let $e = \mathsf{rel(m)}^i$ by thread T. Let ρ be any lockset owner. Let e_ρ be the event corresponding to ρ, i.e., $e_\rho = \mathsf{wr(x)}^h$ if $\rho = \mathsf{x}^h$, or $e_\rho = \mathsf{acq(m)}^h$ if $\rho = \mathsf{m}^h$.

We define the following abbreviations for the left- and right-hand sides of the CP invariant:

Let $LHS = CP(\rho) \cup \big\{\, \sigma \mid \big(\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho) \wedge \exists l \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \prec_{trace} e\big) \,\big\}$.

Let $LHS^+ = CP(\rho)^+ \cup \big\{\, \sigma \mid \big(\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \exists l \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \prec_{trace} e\big) \,\big\}$.

Let $RHS = \big\{\, \sigma \mid \big(\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e\big) \,\big\}$.

Let $RHS^+ = RHS$.

Unlike for the other algorithms, the pre-release algorithm aims to maintain the invariants for the program point *before* e, so we define LHS^+ and RHS^+ with respect to e (not some event e^+).

Suppose Figure 3's invariants hold, i.e., $LHS = RHS$, before the pre-release algorithm executes. We call this assumption the "inductive hypothesis" because we use this lemma's result in the proof of Theorem 1, which is by induction.

To show $LHS^+ = RHS^+$, we first show $LHS^+ \subseteq RHS^+$ (subset direction) and then $LHS^+ \supseteq RHS^+$ (superset direction).

Subset direction: Let $\sigma \in LHS^+$.

Either $\sigma \in CP(\rho)^+$ or $\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \exists l \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \prec_{trace} e$ (or both):

Case 1: $\sigma \in CP(\rho)^+$

If $\sigma \in CP(\rho)$, then by the inductive hypothesis, $\sigma \in RHS$. Since $RHS = RHS^+$, $\sigma \in RHS^+$.

Otherwise ($\sigma \notin CP(\rho)$), Algorithm 4 adds σ to $CP(\rho)^+$, which can happen only at line 5. Let j be such that $\sigma : \mathsf{m}^j \in CCP(\rho)$ (line 3) and $\mathsf{T} \in CP(\mathsf{m}^q) \mid q \geq j$ (line 4). By the inductive hypothesis on $\mathsf{T} \in CP(\mathsf{m}^q)$ (CP invariant), let e' be such that $thr(e') = \mathsf{T} \wedge \mathsf{acq(m)}^q \xrightarrow{CP} e' \prec_{trace} e$. Since $e' \prec_{trace} e$ and $thr(e') = thr(e)$, $e' \xrightarrow{HB} e$ by the definition of HB. Thus $\mathsf{acq(m)}^q \xrightarrow{CP} e$ and therefore $\mathsf{rel(m)}^q \xrightarrow{CP} \mathsf{acq(m)}^i$ by the definition of CP (recall that $e = \mathsf{rel(m)}^i$). If $q = j$, then $\mathsf{rel(m)}^j \xrightarrow{CP} \mathsf{acq(m)}^i$. Otherwise, $q > j$ so $\mathsf{acq(m)}^j \xrightarrow{HB} \mathsf{acq(m)}^q$ by the definition of HB. Thus $\mathsf{acq(m)}^j \xrightarrow{CP} e$ and therefore $\mathsf{rel(m)}^j \xrightarrow{CP} \mathsf{acq(m)}^i$ by the definition of CP. Because of those facts, together with $\sigma : \mathsf{m}^j \in CCP(\rho)$, by the inductive hypothesis (CP invariant), $\sigma \in RHS$. Since $RHS = RHS^+$, $\sigma \in RHS^+$.

Case 2: Let n^k and l be such that $\sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \prec_{trace} e$.

If $\sigma : \mathsf{n}^k \in CCP(\rho)$, then $\sigma \in LHS$, so by the inductive hypothesis, $\sigma \in RHS$. Since $RHS = RHS^+$, $\sigma \in RHS^+$.

Otherwise, $\neg\big(\sigma : \mathsf{n}^k \in CCP(\rho) \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \prec_{trace} e\big)$. We know $\mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \prec_{trace} e$, so therefore $\sigma : \mathsf{n}^k \notin CCP(\rho)$. Thus Algorithm 4 adds $\sigma : \mathsf{n}^k$ to $CCP(\rho)^+$, which can only happen at line 8, which adds $\sigma : \mathsf{n}^k$ to $CCP(\rho)^+$. Let j be such that $\sigma : \mathsf{m}^j \in CCP(\rho)$ (line 3) and $\mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{m}^q) \mid q \geq j$ (line 7). By the inductive hypothesis on $\mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{m}^q)$ and the fact that $\mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \prec_{trace} e$ (CP invariant), let e' be such that $thr(e') = \mathsf{T} \wedge \mathsf{acq(m)}^q \xrightarrow{CP} e' \prec_{trace} e$. Since $e' \prec_{trace} e$ and $thr(e') = thr(e)$, $e' \xrightarrow{HB} e$ by the definition of HB. Thus $\mathsf{acq(m)}^q \xrightarrow{CP} e$ and therefore $\mathsf{rel(m)}^q \xrightarrow{CP} \mathsf{acq(m)}^i$ by the definition of CP (recall $e = \mathsf{rel(m)}^i$). If $q = j$, then $\mathsf{rel(m)}^j \xrightarrow{CP} \mathsf{acq(m)}^i$. Otherwise, $q > j$ so $\mathsf{acq(m)}^j \xrightarrow{HB} \mathsf{acq(m)}^q$ by the definition of HB. Thus $\mathsf{acq(m)}^j \xrightarrow{CP} e$ and therefore $\mathsf{rel(m)}^j \xrightarrow{CP} \mathsf{acq(m)}^i$ by the definition of CP. We have thus determined that $\sigma : \mathsf{m}^j \in CCP(\rho) \wedge \mathsf{rel(m)}^l \xrightarrow{CP} \mathsf{acq(m)}^i \prec_{trace} e$, i.e., $\sigma \in LHS$. By the inductive hypothesis, $\sigma \in RHS$. Since $RHS = RHS^+$, $\sigma \in RHS^+$.

Superset direction: Let $\sigma \in RHS^+$.

Since $RHS = RHS^+$, $\sigma \in RHS$. By the inductive hypothesis, $\sigma \in LHS$. Since Algorithm 4 maintains $CP(\rho)^+ \supseteq CP(\rho)$ and $CCP(\rho)^+ \supseteq CCP(\rho)$, therefore $\sigma \in LHS^+$.

Since $LHS^+ \subseteq RHS^+ \wedge, LHS^+ \supseteq RHS^+$, $LHS^+ = RHS^+$. □

Proof of Lemma 4.

Let $e = \mathsf{rel(m)}^i$ by thread T. Let e^+ be the event immediately after e in the observed total order ($\prec_{trace}$). Let ρ be any lockset owner. Let e_ρ be the event corresponding to ρ, i.e., $e_\rho = \mathsf{wr(x)}^h$ if $\rho = \mathsf{x}^h$, or $e_\rho = \mathsf{acq(m)}^h$ if $\rho = \mathsf{m}^h$.

We define the following abbreviations for the left- and right-hand sides of the CP invariant:

Let $LHS = CP(\rho) \cup \big\{\, \sigma \mid \big(\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e\big) \,\big\}$.

Let $LHS^+ = CP(\rho)^+ \cup \big\{\, \sigma \mid \big(\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+\big) \,\big\}$.

Let $RHS = \big\{\, \sigma \mid \big(\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e\big) \,\big\}$

Let $RHS^+ = \big\{\, \sigma \mid \big(\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e^+\big) \,\big\}$.

Suppose Figure 3's invariants hold before e, i.e., suppose $LHS = RHS$. We call this assumption the "inductive hypothesis" because we use this lemma's result in the proof of Theorem 1, which is by induction.

According to Lemma 3, the inductive hypothesis holds after pre-release algorithm (Algorithm 4). Thus, we need to show that $LHS^+ = RHS^+$ holds after the release algorithm (Algorithm 5).

To show $LHS^+ = RHS^+$, we first show $LHS^+ \subseteq RHS^+$ (subset direction) and then $LHS^+ \supseteq RHS^+$ (superset direction).

Subset direction: Let $\sigma \in LHS^+$.

Either $\sigma \in CP(\rho)^+$ or $\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$ (or both):

Case 1: $\sigma \in CP(\rho)^+$.

If $\sigma \in CP(\rho)$, then by the inductive hypothesis (CP invariant), $\sigma \in RHS$, i.e., $\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. Since $e \prec_{trace} e^+$, $\exists e' \mid e_\rho \xrightarrow{CP} e' \prec_{trace} e^+$, so $\sigma \in RHS^+$.

Otherwise ($\sigma \notin CP(\rho)$), Algorithm 5 must add σ to $CP(\rho)^+$. Algorithm 5 adds to $CP(\rho)^+$ only at line 4, so $\sigma = \mathsf{m}$ and $\mathsf{m} \in CP(\rho)^+$. Line 3 thus evaluates to true for ρ, so $\mathsf{T} \in CP(\rho)$. By the inductive hypothesis (CP invariant), let e' be such that $thr(e') = \mathsf{T} \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. By the definition of HB, $e' \xrightarrow{HB} e$, and thus $e_\rho \xrightarrow{CP} e \prec_{trace} e^+$ by the definition of CP. Therefore $\mathsf{m} \in RHS^+$.

Case 2: Let n^k and j be such that $\sigma : \mathsf{n}^k \in CCP(\rho)^+ \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$.

If $\sigma \in CCP(\rho) \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$, then by the inductive hypothesis (CP invariant), $\sigma \in RHS$, i.e., $\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. Since $e \prec_{trace} e^+$, then $e_\rho \xrightarrow{CP} e' \prec_{trace} e^+$, so $\sigma \in RHS^+$.

Otherwise $\neg\big(\sigma : \mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e\big)$. However, $\exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace}$

e because $\exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$ and $e \neq \mathsf{acq(n)}^j$. Therefore $\sigma : \mathsf{n}^k \notin CCP(\rho)$, so Algorithm 5 adds $\sigma : \mathsf{n}^k$ to $CCP(\rho)^+$, which must happen at line 7, and thus $\sigma = \mathsf{m}$. Therefore $\mathsf{T} : \mathsf{n}^k \in CCP(\rho)$ according to line 6. By the inductive hypothesis (CP invariant), $\exists e' \mid thr(e') = \mathsf{T} \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. By the definition of CP, $e_\rho \xrightarrow{CP} e \prec_{trace} e^+$ since $e = \mathsf{rel(m)}^i$ and $thr(e) = \mathsf{T}$. Therefore $\mathsf{m} \in RHS^+$.

Superset direction: Let $\sigma \in RHS^+$.

Either $\sigma \in RHS$ or not:

Case 1: $\sigma \in RHS$

Then $\sigma \in LHS$ by the inductive hypothesis. Either $\sigma \in CP(\rho)$ or $\exists \mathsf{n}^k \mid \sigma : \mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$ (or both):

Case 1a: If $\sigma \in CP(\rho)$, then $\sigma \in CP(\rho)^+$ because Algorithm 5 maintains $CP(\rho)^+ \supseteq CP(\rho)$.

Case 1b: Let n^k be such that $\sigma : \mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$.

If $\mathsf{n} \neq \mathsf{m}$, then $\sigma : \mathsf{n}^k \in CCP(\rho)^+$ since Algorithm 5 only removes $\sigma : \mathsf{n}^k$ from $CCP(\rho)^+$ if $\mathsf{n} = \mathsf{m}$.

Otherwise, $\mathsf{n} = \mathsf{m}$. Since $\exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$ and $e = \mathsf{rel(m)}^i$, therefore $\mathsf{acq(m)}^k \xrightarrow{CP} \mathsf{acq(m)}^i$ by the definition of CP. By the inductive hypothesis (CP invariant) and Lemma 5 (introduced and proved later in this section), $\mathsf{T} \in CP(\mathsf{m}^k)$ or $\exists \mathsf{o}^l \mid \mathsf{o} \neq \mathsf{m} \wedge \mathsf{T} : \mathsf{o}^l \in CCP(\mathsf{m}^k) \wedge \exists h \mid \mathsf{rel(o)}^l \xrightarrow{CP} \mathsf{rel(o)}^h \prec_{trace} e$.

Case 1b(i): $\mathsf{T} \in CP(\mathsf{m}^k)$

In the pre-release algorithm (Algorithm 4), line 4 evaluated to true for $\sigma : \mathsf{m}^l \in CCP(\rho) \mid l \leq k$ (line 3). Thus line 5 executed, so $\sigma \in CP(\rho)$. Since the release algorithm (Algorithm 5) maintains $CP(\rho)^+ \supseteq CP(\rho)$, therefore $\sigma \in LHS^+$.

Case 1b(ii): Let o^l be such that $\mathsf{o} \neq \mathsf{m} \wedge \mathsf{T} : \mathsf{o}^l \in CCP(\mathsf{m}^k) \wedge \exists h \mid \mathsf{rel(o)}^l \xrightarrow{CP} \mathsf{rel(o)}^h \prec_{trace} e$.

Then the pre-release algorithm (Algorithm 4) executed line 8 for $\sigma : \mathsf{m}^k \in CCP(\rho)$ (line 3) and $\mathsf{T} : \mathsf{o}^l \in CCP(\mathsf{m}^k)$ (line 7). Line 8 added $\sigma : \mathsf{o}^l$ to $CCP(\rho)$. The release algorithm (Algorithm 5) does *not* remove $\sigma : \mathsf{o}^l$ because $\mathsf{o} \neq \mathsf{m}$ (line 14). Therefore $\sigma : \mathsf{o}^l \in CCP(\rho)^+$ and thus $\sigma \in LHS^+$.

Case 2: $\sigma \notin RHS$

Thus $\exists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e^+$ but $\nexists e' \mid appl(\sigma, e') \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. Therefore $e_\rho \xrightarrow{CP} e$. Since $e = \mathsf{rel(m)}^i$, $\sigma = \mathsf{m}$ or $\sigma = \mathsf{T}$. But $\sigma = \mathsf{T}$ would imply $\exists e' \mid thr(e') = \mathsf{T} \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$, contradicting $\sigma \notin RHS$. Therefore $\sigma = \mathsf{m}$.

By the definition of CP and the fact that $e = \mathsf{rel(m)}^i$, let e' be such that $thr(e') = \mathsf{T} \wedge e_\rho \xrightarrow{CP} e' \prec_{trace} e$. By the inductive hypothesis (CP invariant), $\mathsf{T} \in CP(\rho)$ or $\exists \mathsf{n}^k \mid \mathsf{T} : \mathsf{n}^k \in CCP(\rho) \wedge \exists j \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$.

Case 2a: $\mathsf{T} \in CP(\rho)$

Then line 3 in Algorithm 5 evaluates to true for ρ. Therefore line 4 executes, so $\mathsf{m} \in CP(\rho)^+$ and thus $\mathsf{m} \in LHS^+$.

Case 2b: Let n^k and j be such that $\mathsf{T} : \mathsf{n}^k \in CCP(\rho) \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e$.

If $\mathsf{n} \neq \mathsf{m}$, then because $\mathsf{T} : \mathsf{n}^k \in CCP(\rho)$ matches line 6 of Algorithm 5, line 7 adds $\mathsf{m} : \mathsf{n}^k$ to $CCP(\rho)^+$. Since $e \prec_{trace} e^+$, therefore $\mathsf{m} : \mathsf{n}^k \in CCP(\rho) \wedge \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^j \prec_{trace} e^+$, i.e., $\mathsf{m} \in LHS^+$.

Otherwise ($\mathsf{n} = \mathsf{m}$), $\mathsf{m} : \mathsf{m}^k \notin CCP(\rho)^+$ because Algorithm 5 removes $\mathsf{m} : \mathsf{m}^k$ from $CCP(\rho)^+$ at line 14. Since $\mathsf{rel(m)}^k \xrightarrow{CP} \mathsf{acq(m)}^j \prec_{trace} e$, $\mathsf{acq(m)}^k \xrightarrow{CP} \mathsf{acq(m)}^i \prec_{trace} e$ by the definition of CP (since $e = \mathsf{rel(m)}^i$). By the inductive hypothesis (CP invariant) and Lemma 5, $\mathsf{T} \in CP(\mathsf{m}^k)$ or $\exists \mathsf{o}^l \mid \mathsf{o} \neq \mathsf{m} \wedge \mathsf{T} : \mathsf{o}^l \in CCP(\mathsf{m}^k) \wedge \exists h \mid \mathsf{rel(o)}^l \xrightarrow{CP} \mathsf{acq(o)}^h \prec_{trace} e$ (or both).

Case 2b(i): $\mathsf{T} \in CP(\mathsf{m}^k)$

In the pre-release algorithm (Algorithm 4), line 4 evaluated to true for $\mathsf{T} : \mathsf{m}^k \in CP(\rho)$ (line 3). Thus line 5 executed, so $\mathsf{T} \in CP(\rho)$. In the *release* algorithm (Algorithm 5), line 3 evaluates to true for ρ. Therefore Algorithm 5's line 4 executes, so $\mathsf{m} \in CP(\rho)^+$ and thus $\mathsf{m} \in LHS^+$.

Case 2b(ii): Let o^l and h be such that $\mathsf{T} : \mathsf{o}^l \in CCP(\mathsf{m}^k) \wedge \mathsf{rel(o)}^l \xrightarrow{CP} \mathsf{acq(o)}^h \prec_{trace} e$.

In the pre-release algorithm (Algorithm 4), line 8 executed for $\mathsf{T} : \mathsf{m}^k \in CCP(\rho)$ (line 3) and $\mathsf{T} : \mathsf{o}^l \in CCP(\mathsf{m}^k)$ (line 7). Line 8 added $\mathsf{T} : \mathsf{o}^l$ to $CCP(\rho)$. In the *release* algorithm (Algorithm 5), line 7 executes for $\mathsf{T} : \mathsf{o}^l \in CCP(\rho)$ (line 6), adding $\mathsf{m} : \mathsf{o}^l$ to $CCP(\rho)^+$. In addition, $\mathsf{rel(o)}^l \xrightarrow{CP} \mathsf{acq(o)}^h \prec_{trace} e^+$ since $e \prec_{trace} e^+$. Therefore $\mathsf{m} \in LHS^+$.

Since $LHS^+ \subseteq RHS^+ \wedge, LHS^+ \supseteq RHS^+$, $LHS^+ = RHS^+$. □

The proof of Lemma 4 relied on an additional lemma, which we state and prove below. First we state a new definition.

According to the definition of CP, CP-ordered critical sections on lock m are ordered either directly by Rule (a), or indirectly through Rule (b) and Rule (c). We define the *CP-distance* of m^i and j, $d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i)$, as the following:

- 0 if $\exists \mathsf{x}, g, h, i', j' \mid j \leq j' < i' \leq i \wedge \mathsf{acq(m)}^{j'} \xrightarrow{PO} \mathsf{wr(x)}^g \xrightarrow{PO} \mathsf{rel(m)}^{j'} \wedge \mathsf{acq(m)}^{i'} \xrightarrow{PO} \mathsf{wr(x)}^h \xrightarrow{PO} \mathsf{rel(m)}^{i'}$
- $1 + \min d(\mathsf{n}^k \rightsquigarrow \mathsf{n}^l) \mid \mathsf{acq(m)}^j \xrightarrow{HB} \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \xrightarrow{HB} \mathsf{rel(m)}^i$

Lemma 5. *Let e be any release event, i.e., $e = \mathsf{rel(m)}^i$ executed by thread T. If there exists j such that $\mathsf{acq(m)}^j \xrightarrow{CP} e$, and Figure 3's invariants hold for all execution points up to and including the point immediately before e, then after the pre-release algorithm (Algorithm 4) executes but before the release algorithm (Algorithm 5) executes, one or both of the following hold:*

- $\exists j' \geq j \mid \in CP(\mathsf{m}^{j'})$
- $\exists j' \geq j \mid \exists \mathsf{n}^k \mid \mathsf{n} \neq \mathsf{m} \wedge \mathsf{T} : \mathsf{n}^k \in CCP(\mathsf{m}^{j'}) \wedge \exists l \mid \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \prec_{trace} e \wedge d(\mathsf{n}^k \rightsquigarrow \mathsf{n}^l) < d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i)$

Proof of Lemma 5.

Let e be any release event, i.e., $e = \mathsf{rel(m)}^i$ executed by thread T. Let j be such that $\mathsf{acq(m)}^j \xrightarrow{CP} e$. Suppose Figure 3's invariants hold for all execution points up to and including the point immediately before e.

We prove the lemma by induction on the CP-distance $d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i)$.

Base case. If $d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i) = 0$, then by the definition of CP-distance, $\exists \mathsf{x}, g, h, i', j' \mid j \leq j' < i' \leq i \wedge \mathsf{acq(m)}^{j'} \xrightarrow{PO} \mathsf{wr(x)}^g \xrightarrow{PO} \mathsf{rel(m)}^{j'} \wedge \mathsf{acq(m)}^{i'} \xrightarrow{PO} \mathsf{wr(x)}^h \xrightarrow{PO} \mathsf{rel(m)}^{i'}$.

By the CP-rule-A invariant, $\mathsf{T'} \in CP(\mathsf{m}^{j'})$ at $\mathsf{rel(m)}^{i'}$, where $\mathsf{T'} = thr(\mathsf{rel(m)}^{i'})$. Because $\mathsf{rel(m)}^{i'} \xrightarrow{HB} \mathsf{rel(m)}^i$ and the algorithms propogate HB-ordered events through CP locksets, $\mathsf{T} \in CP(\mathsf{m}^{j'})$ at e.

Inductive step. $d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i) > 0$

Inductive hypothesis: Suppose the lemma holds true for all n^l and k such that $d(\mathsf{n}^k \rightsquigarrow \mathsf{n}^l) < d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i)$.

By the definitions of CP and CP-distance, there exist n^l and k such that $\mathsf{acq(m)}^j \xrightarrow{HB} \mathsf{rel(n)}^k \xrightarrow{CP} \mathsf{acq(n)}^l \xrightarrow{HB} \mathsf{rel(m)}^i$, $d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i) = 1 + d(\mathsf{n}^k \rightsquigarrow \mathsf{n}^l)$, and $\mathsf{n} \neq \mathsf{m}$. Either $\mathsf{rel(m)}^i \prec_{trace} \mathsf{rel(n)}^l$ or $\mathsf{rel(n)}^l \prec_{trace} \mathsf{rel(m)}^i$:

Case 1: $\mathsf{rel(m)}^i \prec_{trace} \mathsf{rel(n)}^l$

Since $\mathsf{acq(m)}^j \xrightarrow{HB} \mathsf{rel(n)}^k$, at $\mathsf{acq(n)}^l$, there exists $k' \leq k$ such that Algorithm 3 adds $\mathsf{T'} : \mathsf{n}^{k'}$ to $CCP(\mathsf{m}^j)$, where $\mathsf{T'} = thr(\mathsf{acq(n)}^l)$. Since $\mathsf{acq(n)}^l \xrightarrow{HB} \mathsf{rel(m)}^i$, and the algorithms propagate CCP through HB-ordered events, at $e = \mathsf{rel(m)}^i$, $\mathsf{T} : \mathsf{n}^{k'} \in CCP(\mathsf{m}^j)$.

Case 2: $\mathsf{rel(n)}^l \prec_{trace} \mathsf{rel(m)}^i$

Thus we *cannot* show that, at $e = \mathsf{rel(m)}^i$, $\mathsf{T} : \mathsf{n}^{k'} \notin CCP(\mathsf{m}^j)$. To prove this case, we are interested in the release of a lock like n with "minimum distance." Let o^f and g be such that

- $d(\mathsf{o}^f \rightsquigarrow \mathsf{o}^g)$ is as minimal as possible;
- $\mathsf{rel(o)}^g \prec_{trace} \mathsf{rel(m)}^i$; and
- there exists $f' \geq f$ and σ such that, at $\mathsf{rel(o)}^g$, $\exists j' \geq j \mid \sigma : \mathsf{o}^{f'} \in CCP(\mathsf{m}^{j'}) \wedge \exists e' \mid \mathsf{acq(m)}^j \xrightarrow{CP} e' \wedge appl(\sigma, e') \wedge e' \xrightarrow{HB} e$.

Note that o may be n, or else o is some other "lower-distance" lock, and thus $d(\mathsf{o}^f \rightsquigarrow \mathsf{o}^g) < d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i)$ in any case. Note that this includes the possibility that $\mathsf{o} = \mathsf{m}$, in which case $f < j$ and $g < i$.

Since $d(\mathsf{o}^f \rightsquigarrow \mathsf{o}^g) < d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i)$, by the inductive hypothesis, at $\mathsf{rel(o)}^g$ (after the pre-release algorithm but before the release algorithm), there exists $f' \geq f$ such that either

- $\mathsf{T'} \in CP(\mathsf{o}^{f'})$ or
- $\exists \mathsf{q}^c \mid \mathsf{q} \neq \mathsf{o} \wedge \mathsf{T'} : \mathsf{q}^c \in CCP(\mathsf{o}^{f'}) \wedge \exists d \mid \mathsf{rel(q)}^c \xrightarrow{CP} \mathsf{acq(q)}^d \prec_{trace} \mathsf{rel(o)}^g \wedge d(\mathsf{q}^c \rightsquigarrow \mathsf{q}^d) < d(\mathsf{o}^f \rightsquigarrow \mathsf{o}^g)$

where $\mathsf{T'} = thr(\mathsf{rel(o)}^g)$.

Case 2a: $\mathsf{T'} \in CP(\mathsf{o}^{f'})$

At $\mathsf{rel(o)}^g$, the pre-release algorithm (Algorithm 4) adds σ to $CP(\mathsf{m}^{j'})$ at line 5 because $\sigma : \mathsf{o}^{f'} \in CCP(\mathsf{m}^{j'})$ (matching line 3) and $\mathsf{T'} \in CP(\mathsf{o}^{f'})$ (matching line 4). Since $appl(\sigma, e') \wedge e' \xrightarrow{HB} e$, and the algorithms propagate CP through HB-ordered events, at $e = \mathsf{rel(m)}^i$, $\mathsf{T} \in CP(\mathsf{m}^{j'})$.

Case 2b: Let q^c and d be such that, at $\mathsf{rel(o)}^g$, $\mathsf{q} \neq \mathsf{o} \wedge \mathsf{T'} : \mathsf{q}^c \in CCP(\mathsf{o}^{f'}) \wedge \mathsf{rel(q)}^c \xrightarrow{CP} \mathsf{acq(q)}^d \prec_{trace} \mathsf{rel(o)}^g \wedge d(\mathsf{q}^c \rightsquigarrow \mathsf{q}^d) < d(\mathsf{o}^f \rightsquigarrow \mathsf{o}^g)$.

Thus $\mathsf{acq(q)}^d \prec_{trace} \mathsf{rel(o)}^g \prec_{trace} \mathsf{rel(q)}^d$.

If $\mathsf{rel(q)}^d \prec_{trace} \mathsf{rel(m)}^i$, then that would violate the stipulation above that $d(\mathsf{o}^f \rightsquigarrow \mathsf{o}^g)$ is minimal. Thus either $\mathsf{rel(m)}^i = \mathsf{rel(q)}^d$ or $\mathsf{rel(m)}^i \prec_{trace} \mathsf{rel(q)}^d$.

Case 2b(i): $\mathsf{rel(m)}^i = \mathsf{rel(q)}^d$

Thus $\mathsf{m} = \mathsf{q}$ and $i = d$, but $c < j$ since $d(\mathsf{m}^c \rightsquigarrow \mathsf{m}^i) = d(\mathsf{q}^c \rightsquigarrow \mathsf{q}^d) < d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i)$.

At $\mathsf{rel(o)}^g$, the pre-release algorithm (Algorithm 4) adds $\sigma : \mathsf{m}^c$ to $CCP(\mathsf{m}^{j'})$ at line 8 because $\sigma : \mathsf{o}^{f'} \in CCP(\mathsf{m}^{j'})$ (matching line 3) and $\mathsf{T'} : \mathsf{m}^c \in CCP(\mathsf{o}^{f'})$ (matching line 7). Since $appl(\sigma, e') \wedge e' \xrightarrow{HB} e$, and the algorithms propagate CCP through HB-ordered events, at $e = \mathsf{rel(m)}^i$, $\mathsf{T} : \mathsf{m}^{c'} \in CCP(\mathsf{m}^{j'})$.

By the inductive hypothesis on m^i and c, since $d(\mathsf{m}^c \rightsquigarrow \mathsf{m}^i) < d(\mathsf{m}^j \rightsquigarrow \mathsf{m}^i)$, there exists $c' \geq c$ such that, at $\mathsf{rel(m)}^i$, one or both of the following hold:

- $\mathsf{T} \in CP(\mathsf{m}^{c'})$
  The pre-release algorithm (Algorithm 4) adds T to $CP(\mathsf{m}^{j'})$ at line 5 because $\mathsf{T} : \mathsf{m}^c \in CCP(\mathsf{m}^{j'})$ (matching line 3) and $\mathsf{T} \in CP(\mathsf{m}^{c'})$ (matching line 4).
- $\exists \mathsf{r}^a \mid \mathsf{r} \neq \mathsf{m} \wedge \mathsf{T} : \mathsf{r}^a \in CCP(\mathsf{m}^{c'}) \wedge \exists b \mid \mathsf{rel(r)}^a \xrightarrow{CP} \mathsf{acq(r)}^b \prec_{trace} \mathsf{rel(m)}^i \wedge d(\mathsf{r}^a \rightsquigarrow \mathsf{r}^b) < d(\mathsf{m}^c \rightsquigarrow \mathsf{m}^i)$
  The pre-release algorithm (Algorithm 4) adds $\mathsf{T} : \mathsf{r}^a$ to $CCP(\mathsf{m}^{j'})$ at line 8 because

$\mathsf{T} : \mathsf{m}^{c'} \in CCP(\mathsf{m}^{j'})$ (matching line 3) and $\mathsf{T}:\mathsf{r}^{a} \in CCP(\mathsf{m}^{c'})$ (matching line 7).

Case 2b(ii): $\mathsf{rel}(\mathsf{m})^{i} \prec_{trace} \mathsf{rel}(\mathsf{q})^{d}$

Note that $\mathsf{m} \neq \mathsf{q}$ because $\mathsf{acq}(\mathsf{q})^{d} \prec_{trace} \mathsf{rel}(\mathsf{o})^{g} \prec_{trace} \mathsf{rel}(\mathsf{m})^{i} \prec_{trace} \mathsf{rel}(\mathsf{q})^{d}$.

At $\mathsf{rel}(\mathsf{o})^{g}$, the pre-release algorithm (Algorithm 4) adds $\sigma : \mathsf{q}^{c}$ to $CCP(\mathsf{m}^{j'})$ at line 8 because $\sigma : \mathsf{o}^{f'} \in CCP(\mathsf{m}^{j'})$ (matching line 3) and $\mathsf{T}' : \mathsf{q}^{c} \in CCP(\mathsf{o}^{f'})$ (matching line 7). Since $appl(\sigma, e') \wedge e' \xrightarrow{HB} e$, and the algorithms propagate CCP through HB-ordered events, at $e = \mathsf{rel}(\mathsf{m})^{i}$, $\mathsf{T}:\mathsf{q}^{c'} \in CCP(\mathsf{m}^{j'})$.

Thus, the lemma's conclusions hold for each case. □

We are now ready to prove Theorem 1 from Section 7.

Theorem 1. *After every event, Raptor (i.e., the analysis in Algorithms 1–5) maintains the invariants in Figure 3.*

Proof of Theorem 1. By induction on the events in the observed total order of events.

Base case. Let e be the first event in the total order. Before e, all locksets are empty, so the LHS of each invariant is empty or false (depending on the invariant). The RHS of each invariant is also empty or false, either because there is no earlier event $e' \prec_{trace} e$ (PO, HB, HB-index, HB-critical-section, CP, and CCP-constraint invariants), or all locksets are empty (CP-rule-A invariant).

The exception to the above is that $PO()$ for "fake" initial accesses is non-empty, i.e., $PO(\mathsf{x}^{0}) = \{\xi\}$ for every variable x (Section 6.1). The PO invariant holds because we assume that the first real access is PO ordered to the fake initial access, i.e., $\mathsf{wr}(\mathsf{x})^{0} \xrightarrow{PO} \mathsf{wr}(\mathsf{x})^{1}$.

Inductive step. Suppose the invariants in Figure 3 hold immediately before an event e. If $e = \mathsf{wr}(\mathsf{x})^{i}$, the invariants hold after e, by Lemma 1. If $e = \mathsf{acq}(\mathsf{m})^{i}$, the invariants hold after e, by Lemma 2. If $e = \mathsf{rel}(\mathsf{m})^{i}$, the invariants hold after e, by Lemmas 3 and 4. □

Finally, we prove the soundness and completeness theorem from Section 7.

Theorem 2. *An execution has a CP-race if and only if Raptor reports a race for the execution.*

Proof of Theorem 2. We prove the forward direction ($\Longrightarrow$) and backward direction ($\Longleftarrow$) in turn:

Forward direction (completeness). Suppose a CP-race exists between two accesses x^{i} and x^{i+1}, but Raptor does not report a CP-race. Thus $\mathsf{wr}(\mathsf{x})^{i} \not\xrightarrow{CP} \mathsf{wr}(\mathsf{x})^{i+1} \wedge \mathsf{wr}(\mathsf{x})^{i} \not\xrightarrow{PO} \mathsf{wr}(\mathsf{x})^{i+1}$, but $\xi \in CP(\mathsf{x}^{i}) \cup PO(\mathsf{x}^{i})$ at program termination. According to Theorem 1, by the CP and PO invariants, $\xi \notin CP(\mathsf{x}^{i}) \vee \xi \notin PO(\mathsf{x}^{i})$ at program termination, which is a contradiction.

Backward direction (soundness). Suppose Raptor reports a CP-race between two accesses x^{i} and x^{i+1}, but no CP-race exists. Thus $\xi \notin CP(\mathsf{x}^{i}) \cup PO(\mathsf{x}^{i})$ at program termination, but $\mathsf{wr}(\mathsf{x})^{i} \xrightarrow{CP} \mathsf{wr}(\mathsf{x})^{i+1} \vee \mathsf{wr}(\mathsf{x})^{i} \xrightarrow{PO} \mathsf{wr}(\mathsf{x})^{i+1}$.

Since $\xi \notin PO(\mathsf{x}^{i})$ at program termination, according to Theorem 1, by the PO invariant, $\mathsf{wr}(\mathsf{x})^{i} \not\xrightarrow{PO} \mathsf{wr}(\mathsf{x})^{i+1}$. Thus $\mathsf{wr}(\mathsf{x})^{i} \xrightarrow{CP} \mathsf{wr}(\mathsf{x})^{i+1}$. By the CP invariant, at program termination,

$$\xi \in CP(\mathsf{x}^{i}) \vee \big(\exists \mathsf{n}^{k} \mid \xi:\mathsf{n}^{k} \in CCP(\mathsf{x}^{i}) \wedge \exists j \mid \mathsf{rel}(\mathsf{n})^{k} \xrightarrow{CP} \mathsf{acq}(\mathsf{n})^{j} \prec_{trace} e^{\Omega}\big)$$

where e^{Ω} represents a final "program termination" event. Since $\xi \notin CP(\mathsf{x}^{i})$, $\exists \mathsf{n}^{k} \mid \xi : \mathsf{n}^{k} \in CCP(\mathsf{x}^{i})$. By the CCP-constraint invariant, $\exists l \mid \mathsf{acq}(\mathsf{n})^{l} \prec_{trace} e^{\Omega} \wedge \mathsf{rel}(\mathsf{n})^{l} \not\prec_{trace} e^{\Omega}$. But a thread releases all of its held locks before program termination (Section 8), so $\mathsf{rel}(\mathsf{n})^{l} \not\prec_{trace} e^{\Omega}$ is a contradiction. □

C. Detailed Analysis Changes for Reads

This section presents Raptor's analysis that handles not only writes but also reads, corresponding to the changes described in Section 9. Algorithm 8 adds analysis steps to handle a read event. Algorithm 9 modifies the analysis for a write event (Algorithm 2) to handle reads.

Algorithm 8 $\mathsf{rd}(\mathsf{x})^i_{\mathsf{T}}$ by T

▷ Apply Rule (a)
▷ Write-Read Race
for all $\mathsf{m} \in heldBy(\mathsf{T})$ **do**
  if $\exists j \exists h \mid \mathsf{m}^j_* \in HB(\mathsf{x}^h) \wedge \mathsf{T} \notin PO(\mathsf{x}^h)$ **then**
    $CP(\mathsf{m}^j)^+ \leftarrow CP(\mathsf{m}^j)^+ \cup \{\mathsf{T}\}$
  end if
end for
▷ Add termination CCP elements
if $\mathsf{T} \in PO(\mathsf{x}^i)$ **then**
  $PO(\mathsf{x}^i)^+ \leftarrow PO(\mathsf{x}^i)^+ \cup \{\xi_{\mathsf{T}}\}$
end if
if $\mathsf{T} \in HB(\mathsf{x}^i)$ **then**
  $HB(\mathsf{x}^i)^+ \leftarrow HB(\mathsf{x}^i)^+ \cup \{\xi_{\mathsf{T}}\}$
end if
if $\mathsf{T} \in CP(\mathsf{x}^i)$ **then**
  $CP(\mathsf{x}^i)^+ \leftarrow CP(\mathsf{x}^i)^+ \cup \{\xi_{\mathsf{T}}\}$
end if
for all $\mathsf{m} \mid \mathsf{T}:\mathsf{m} \in CCP(\mathsf{x}^i)$ **do**
  $CCP(\mathsf{x}^i)^+ \leftarrow CCP(\mathsf{x}^i)^+ \cup \{\xi_{\mathsf{T}}:\mathsf{m}\}$
end for
▷ Initialize locksets for x^i_{T}.
▷ Reset locksets if x^i_{T} already exists.
if $\mathsf{T} \in PO(\mathsf{x}^i_{\mathsf{T}})$ **then**
  $PO(\mathsf{x}^i_{\mathsf{T}}) \leftarrow \emptyset, HB(\mathsf{x}^i_{\mathsf{T}}) \leftarrow \emptyset$
  $CP(\mathsf{x}^i_{\mathsf{T}}) \leftarrow \emptyset, CCP(\mathsf{x}^i_{\mathsf{T}}) \leftarrow \emptyset$
end if
$PO(\mathsf{x}^i_{\mathsf{T}}) \leftarrow \{\mathsf{T}\}$
$HB(\mathsf{x}^i_{\mathsf{T}}) \leftarrow \{\mathsf{T}\} \cup \{\mathsf{m}^v_* \mid \mathsf{m}^v \in heldBy(\mathsf{T})\}$

Algorithm 9 Modified $\mathsf{wr}(\mathsf{x})^i$ by T

▷ First apply Rule (a)
▷ Write-Write and Read-Write Race
for all $\mathsf{m} \in heldBy(\mathsf{T})$ **do**
  if $\exists j \exists h \mid \mathsf{m}^j_* \in HB(\mathsf{x}^h) \wedge \mathsf{T} \notin PO(\mathsf{x}^h)$ **then**
    $CP(\mathsf{m}^j)^+ \leftarrow CP(\mathsf{m}^j)^+ \cup \{\mathsf{T}\}$
  end if
  for all Threads t **do**
    if $\exists j \exists h \mid \mathsf{m}^j_* \in HB(\mathsf{x}^h_{\mathsf{t}}) \wedge \mathsf{T} \notin PO(\mathsf{x}^h_{\mathsf{t}})$ **then**
      $CP(\mathsf{m}^j)^+ \leftarrow CP(\mathsf{m}^j)^+ \cup \{\mathsf{T}\}$
    end if
  end for
end for
▷ Add termination CCP elements
if $\mathsf{T} \in PO(\mathsf{x}^{i-1})$ **then**
  $PO(\mathsf{x}^{i-1})^+ \leftarrow PO(\mathsf{x}^{i-1})^+ \cup \{\xi\}$
end if
if $\mathsf{T} \in HB(\mathsf{x}^{i-1})$ **then**
  $HB(\mathsf{x}^{i-1})^+ \leftarrow HB(\mathsf{x}^{i-1})^+ \cup \{\xi\}$
end if
if $\mathsf{T} \in CP(\mathsf{x}^{i-1})$ **then**
  $CP(\mathsf{x}^{i-1})^+ \leftarrow CP(\mathsf{x}^{i-1})^+ \cup \{\xi\}$
end if
for all $\mathsf{m} \mid \mathsf{T}:\mathsf{m} \in CCP(\mathsf{x}^{i-1})$ **do**
  $CCP(\mathsf{x}^{i-1})^+ \leftarrow CCP(\mathsf{x}^{i-1})^+ \cup \{\xi:\mathsf{m}\}$
end for
for all Threads t **do**
  if $\mathsf{T} \in PO(\mathsf{x}^{i-1}_{\mathsf{t}})$ **then**
    $PO(\mathsf{x}^{i-1}_{\mathsf{t}})^+ \leftarrow PO(\mathsf{x}^{i-1}_{\mathsf{t}})^+ \cup \{\xi\}$
  end if
  if $\mathsf{T} \in HB(\mathsf{x}^{i-1}_{\mathsf{t}})$ **then**
    $HB(\mathsf{x}^{i-1}_{\mathsf{t}})^+ \leftarrow HB(\mathsf{x}^{i-1}_{\mathsf{t}})^+ \cup \{\xi\}$
  end if
  if $\mathsf{T} \in CP(\mathsf{x}^{i-1}_{\mathsf{t}})$ **then**
    $CP(\mathsf{x}^{i-1}_{\mathsf{t}})^+ \leftarrow CP(\mathsf{x}^{i-1}_{\mathsf{t}})^+ \cup \{\xi\}$
  end if
  for all $\mathsf{m} \mid \mathsf{T}:\mathsf{m} \in CCP(\mathsf{x}^{i-1}_{\mathsf{t}})$ **do**
    $CCP(\mathsf{x}^{i-1}_{\mathsf{t}})^+ \leftarrow CCP(\mathsf{x}^{i-1}_{\mathsf{t}})^+ \cup \{\xi:\mathsf{m}\}$
  end for
end for
▷ Initialize locksets for x^i
$PO(\mathsf{x}^i) \leftarrow \{\mathsf{T}\}$
$HB(\mathsf{x}^i) \leftarrow \{\mathsf{T}\} \cup \{\mathsf{m}^v_* \mid \mathsf{m}^v \in heldBy(\mathsf{T})\}$