An Online Dynamic Analysis for Sound Predictive Data Race Detection


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 behavior 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, 53, 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].

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, wr(x)²’s occurrence might depend on the order of accesses to y. (Assume that 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-

---

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

---

* This material is based upon work supported by the National Science Foundation under Grants CSR-1218695, CAREER-1253703, CCF-1421612, and XPS-1629126.
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. In Figure 1(a), although wr(x)$^1$ $\xrightarrow{HB}$ wr(x)$^2$, wr(x)$^1$ $\xrightarrow{CP}$ wr(x)$^2$. In contrast, in Figure 1(b), wr(x)$^1$ $\xrightarrow{CP}$ 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.

2.1 Events

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

We define a total order $\prec_{\text{trace}}$ for the observed order of events: for two events $e$ and $e'$, $e \prec_{\text{trace}} e'$ if $e$ occurs before $e'$ in the total order. We define a helper function $\text{thr}(e)$ 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 $\prec_{\text{trace}}$ e' $\land$ thr(e) = thr(e')) $\lor$ 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 $e = \text{rel(m)}^i$ $\land$ e' = acq(m)$^j$ $\land$ $e \prec_{\text{trace}}$ 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 $\exists e'' \mid e \xrightarrow{PO} e''$ $\land$ $e'' \xrightarrow{HB}$ 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, $e \xrightarrow{CP}$ acq(m)$^j$ if there exist events $e = \text{wr(x)}^k$ and $e' = \text{wr(x)}^j$ such that $e \prec_{\text{trace}}$ e' $\land$ thr(e) $\neq$ thr(e') $\land$ $\text{acq(m)}^j$ $\xrightarrow{PO}$ $e \xrightarrow{PO}$ rel(m)$^j$ $\land$ acq(m)$^k$ $\xrightarrow{PO}$ e' $\xrightarrow{PO}$ 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: $\text{rel(m)}^j$ $\xrightarrow{CP}$ acq(m)$^j$ if acq(m)$^j$ $\xrightarrow{CP}$ 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' $\lor$ 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

---

2 More precisely, two conflicting accesses unordered by CP indicate either a data race or a deadlock in some other execution [56].
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 $e = \text{wr}(x)^1$ and $e' = \text{wr}(x)^2$ if $e \xrightarrow{PO} e' \wedge e' \xrightarrow{PO} e$. 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), $\text{wr}(x)^1 \xrightarrow{HB} \text{wr}(x)^2$ because $\text{rel}(m)^1 \xrightarrow{HB} \text{acq}(m)^2 \land \text{wr}(x)^1 \xrightarrow{PO} \text{rel}(m)^1 \land \text{acq}(m)^2 \xrightarrow{PO} \text{wr}(x)^2$. In Figure 1(a), $\text{wr}(x)^1 \xrightarrow{CP} \text{wr}(x)^2$. In contrast, in Figure 1(b), $\text{wr}(x)^1 \xrightarrow{CP} \text{wr}(x)^2$ because $\text{rel}(m)^1 \xrightarrow{CP} \text{acq}(m)^2 \land \text{wr}(x)^1 \xrightarrow{HB} \text{rel}(m)^1 \land \text{acq}(m)^2 \xrightarrow{HB} \text{wr}(x)^2 (\text{rel}(m)^1 \xrightarrow{CP} \text{acq}(m)^2)$ 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 heavy-weight 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.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{wr}(x)^1$</td>
<td>$\text{acq}(n)^1$</td>
<td>$\text{acq}(m)^1$</td>
<td>$\text{acq}(v)^1$</td>
<td>$\text{acq}(u)^1$</td>
</tr>
<tr>
<td>$\text{acq}(m)^1$</td>
<td>$\text{rel}(u)^1$</td>
<td>$\text{rel}(m)^1$</td>
<td>$\text{rel}(v)^1$</td>
<td>$\text{rel}(u)^2$</td>
</tr>
<tr>
<td>$\text{rel}(t)^1$</td>
<td>$\text{rel}(q)^1$</td>
<td>$\text{rel}(n)^2$</td>
<td>$\text{acq}(q)^2$</td>
<td>$\text{acq}(q)^1$</td>
</tr>
</tbody>
</table>

**Figure 2.** An example execution in which $\text{wr}(x)^1 \xrightarrow{CP} \text{wr}(x)^2$. 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, $\text{wr}(x)^1 \xrightarrow{CP} \text{wr}(x)^2$ through the following logic: $\text{rel}(q)^1 \xrightarrow{CP} \text{acq}(q)^2$ by Rule (a) implies $\text{acq}(n)^1 \xrightarrow{CP} \text{rel}(n)^2$ by Rule (c), which implies $\text{rel}(m)^1 \xrightarrow{CP} \text{acq}(m)^2$ by Rule (b), which implies $\text{acq}(m)^1 \xrightarrow{CP} \text{rel}(m)^2$ by Rule (c), which implies $\text{rel}(m)^1 \xrightarrow{CP} \text{acq}(m)^2$ by Rule (b), which implies $\text{wr}(x)^1 \xrightarrow{CP} \text{wr}(x)^2$ by Rule (c).

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

Furthermore, at $\text{wr}(x)^2$, an online analysis cannot determine that $\text{wr}(x)^1 \xrightarrow{CP} \text{wr}(x)^2$ because it is not yet knowable. Not until $\text{wr}(y)^2$ is it knowable that $\text{acq}(n)^1 \xrightarrow{CP} \text{rel}(n)^2$ and thus $\text{wr}(x)^1 \xrightarrow{CP} \text{wr}(x)^2$. 
Now suppose instead that $T_1$ executed its critical section on $t$ before its critical section on $m$. In that subtle different execution, $wr(x)^1 \xrightarrow{CP} wr(x)^2$. 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 $HB(x^i)$, it means that any future event that acquires $m$ is HB ordered to $wr(x)^i$. Similarly, $T_2 \in CP(y^i)$ means that $wr(y)^i$ is CP ordered with future events by thread $T_2$. 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, $e \xrightarrow{CP} e'$ ($e \xrightarrow{HB} e'$).

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 $wr(x)^i$ 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 $wr(x)^{i+1}$, it is not in general knowable whether $wr(x)^i \xrightarrow{CP} wr(x)^{i+1}$. For example, in Figure 2, even after $wr(x)^2$ executes, Raptor must continue to maintain information for $wr(x)^i$ because $wr(x)^1 \xrightarrow{CP} wr(x)^2$ 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 scalable to full execution traces as prior work’s offline approach for computing CP [56]. However, as we show, Raptor can safely delete an access $wr(x)^i$’s locksets as soon as it determines whether $wr(x)^{i+1}$ is CP ordered to $wr(x)^i$.

Locksets for lock acquires. Raptor tracks CP, HB, and PO locksets not just for variable accesses, but also for lock acquire operations (e.g., $acq(m)^i$). The reason is that Raptor needs to detect whether $acq(m)^i \xrightarrow{CP} rel(m)^i$, which by Rule (b) of the CP definition implies that $rel(m)^i \xrightarrow{CP}$...
lock instance upon which CP ordering is conditional (e.g., \(m : n')\) or \(T : n\). In addition, each lockset for a variable instance \(x^h\) contains a special element \(\xi\), which indicates ordering between \(wr(x)^i\) and \(wr(x)^{i+1}\).

Figure 3 shows invariants that the Raptor analysis maintains for every lockset owner \(\rho\). 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 \(e_\rho\). From the definition of PO, we know that only one thread (the thread that executes \(e_\rho\)) will be PO ordered to \(e_\rho\). To the invariant, no locks will ever be PO ordered to \(e_\rho\).

In addition, for any \(\rho = x^h\), \(PO(x^h)\) may contain the special element \(\xi\), which indicates that \(wr(x)^h\) is PO ordered with the next access to \(x\), i.e., \(wr(x)^h \xrightarrow{PO} wr(x)^{h+1}\). (Raptor does not strictly need \(\xi\) in \(PO(x^h)\)) to indicate \(wr(x)^h \xrightarrow{PO} wr(x)^{h+1}\), since \(PO(x^h)\) does not add new threads or locks after \(wr(x)^h\) executes. However, since Raptor needs \(\xi\) for CP and CCP locksets, it adds \(\xi\) 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 \(e_\rho\). 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 \(e_\rho\), meaning that some event (before \(e\)) by the same thread or release of the same lock, respectively, is HB ordered to \(e_\rho\). 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 \(e_\rho\). In addition, like \(PO(\rho)\), \(HB(\rho)\) can contain a special element \(\xi\), which indicates, for \(\rho = x^h\) only, that \(wr(x)^h \xrightarrow{HB} wr(x)^{h+1}\).

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

According to the HB-critical-section invariant, for \(\rho = x^h\) only, \(m^i \in HB(\rho)\) may have a subscript \(\ast\) (i.e., \(m^i_\ast\)), which means that not only \(e_\rho \xrightarrow{HB} rel(m)^i\), but also that \(e_\rho\) executed inside the \(acq(m)^i - rel(m)^i\) 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 \(e_\rho\). 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 \(\sigma\) is not necessarily in \(CP(\rho)\) even after an event \(e'\) such that \(app!((\sigma, e') \land e_\rho \xrightarrow{CP} e')\).

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 \(e_\rho\), 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 \(wr(x)^i\) and \(wr(x)^{i+1}\)? In particular, suppose that at some event after \(T\) executes \(wr(x)^{i+1}\), Raptor determines that \(T\) is CP ordered to \(wr(x)^i\); then how does Raptor know whether \(wr(x)^i \xrightarrow{CP} wr(x)^{i+1}\)? Raptor addresses this challenge by using the special thread-like element \(\xi\) that essentially represents the thread \(T\) up to event \(wr(x)^{i+1}\) only, so \(\xi \in CP(x^i)\) only if \(wr(x)^i \xrightarrow{CP} wr(x)^{i+1}\).

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., \(wr(x)^j\) and \(wr(x)^k\)), 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 \(T \in CP(m^i)\) as soon as \(wr(x)^i\) executes (\(T = thr(wr(x)^i)\)). 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 \(rel(n)^k \xrightarrow{CP} acq(n)^j\) if \(acq(n)^k \xrightarrow{CP} rel(n)^j\). To account for this rule, Raptor introduces the CCP(\(\rho)\) lockset, which contains elements of the form \(\sigma : n^k\), which means that \(\sigma\) is CP ordered with \(e_\rho\) if \(acq(n)^k \xrightarrow{CP} rel(n)^j\), 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 \(\sigma : n^k \in CCP(\rho)\) only if a critical section on lock \(n\) is ongoing. As Section 6 shows, when \(n\)'s current critical section ends (at \(rel(n)^j\)), Raptor either (1) determines whether \(acq(n)^k \xrightarrow{CP} rel(n)^j\), or (2) it identifies another lock \(q\) that has an ongoing critical section such that it is correct to add some \(\sigma : q^j\) to \(CCP(\rho)\).

Like \(CP(\rho)\), when \(\rho = x^i\), \(CCP(\rho)\) can contain the special thread-like element \(\xi\). More precisely, \(\xi : n^k \in\)
Let \( e \) be any event in the program trace. The following invariants hold for the point in the trace immediately before \( e \).

Let \( e^* = \text{wr}(x)^{k+1} \) if \( e = \text{wr}(x)^{k} \); otherwise \( e \) is a lock acquire or release event, \( e^* \) is an "invalid event" that matches no real event.

We define the boolean function \( \text{appl}(\sigma, e') \) that returns whether event \( e' \) "applies to" lockset element \( \sigma \):

\[
\text{appl}(\sigma, e') := \begin{cases} 
\exists i \mid e' = \text{rel}(m)^i & \text{if } \sigma \text{ is a thread } T \\
\sigma = \text{acq}(m) & \text{if } \sigma \text{ is a lock } m \\
\text{otherwise } (\sigma = \xi) & 
\end{cases}
\]

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

**[PO]** \( \text{PO}(\rho) = \{ \sigma \mid \sigma \text{ is not a lock } \land \exists e' \mid \text{appl}(\sigma, e') \land e_\rho \xrightarrow{\text{PO}} e' \xrightarrow{\text{trace}} e \} \)

**[HB]** \( \text{HB}(\rho) = \{ \sigma \mid \exists e' \mid \text{appl}(\sigma, e') \land e_\rho \xrightarrow{\text{HB}} e' \xrightarrow{\text{trace}} e \} \)

**[HB-critical-section]** \( \text{m}^i \xrightarrow{\text{HB}} \text{HB}(\rho) \iff (e_\rho \xrightarrow{\text{PO}} \text{rel}(m)^{j-1} \land e_\rho \xrightarrow{\text{HB}} \text{rel}(m)^j \land e_\rho = \text{wr}(x)^{h} \land e_\rho \xrightarrow{\text{trace}} e) \)

**[CP]** \( \text{CP}(\rho) \cup \{ \sigma \mid \exists n^k \mid \exists n^k \in \text{CCP}(\rho) \land \exists j \mid \text{rel}(n)^{j} \xrightarrow{\text{CP}} \text{acq}(m)^{j} \xrightarrow{\text{trace}} e \} = \{ \sigma \mid \exists e' \mid \text{appl}(\sigma, e') \land e_\rho \xrightarrow{\text{CP}} e' \xrightarrow{\text{trace}} e \} \)

**[CCP-constraint]** \( \exists e' \mid \text{CCP}(\rho) \implies (\exists j \mid \text{acq}(n)^{j} \xrightarrow{\text{trace}} e \land \text{rel}(n)^{j} \neq e) \)

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

\( \text{CCP}(x^i) \) means that \( \text{wr}(x)^{i} \xrightarrow{\text{CP}} \text{wr}(x)^{i+1} \) if \( \text{acq}(n)^{k} \xrightarrow{\text{CP}} \text{rel}(n)^{j} \), 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 \( \text{PO}(\rho), \text{HB}(\rho), \text{CP}(\rho), \) and \( \text{CCP}(\rho) \) for each lockset owner \( \rho \). Assuming that immediately before \( e \), the analysis state satisfies the invariants, then at \( e \), 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 \( \rho \)'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 \( \text{PO}(\rho), \text{HB}(\rho), \text{CP}(\rho), \) and \( \text{CCP}(\rho) \). The analysis represents the state immediately after \( e \) by adding a superscript +, i.e., \( \text{PO}(\rho)^+, \text{HB}(\rho)^+, \text{CP}(\rho)^+, \text{CCP}(\rho)^+ \).

### 6.1 Initialization

For every lockset owner \( \rho \), whether it is a variable instance \( x^l \) or a lock instance \( m^i \), its locksets are initially empty, i.e., \( \text{PO}(\rho) = \text{HB}(\rho) = \text{CP}(\rho) = \text{CCP}(\rho) = \emptyset \). 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 \( \text{PO}(x^0) \) to \( \xi \), as Algorithm 1 shows. (The other locksets for \( x^0 \) are \( \emptyset \).) This initial state ensures that the first access to \( x^0 \), \( \text{wr}(x^0)^1 \), will appear to be PO-ordered to the prior access to \( x^0 \), without requiring any logic to handle this corner case.

**Algorithm 1**

Initialize locksets

```
for all variables x do
    \text{PO}(x^0) \leftarrow \{ \xi \}
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., \( e = \text{wr}(x)^{l} \) 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 \( \text{wr}(x)^{l-1} \); and initializes the locksets for \( x^l \).

**Applying Rule (a).** Lines 2–7 of Algorithm 2 show how the analysis applies Rule (a). The helper function \( \text{heldBy}(T) \) 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 \( x^{l} \) 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 \( \text{CP}(m^j) \), satisfying the CP-rule-A invariant (Figure 3). Later, when \( T \) releases \( m \), the analysis updates other \( \text{CP}(\cdot) \) locksets, as Section 6.4 describes.

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

At a program’s acquire of a lock, i.e., \( e = \text{acq}(m) \) by thread \( T \), the analysis performs the actions in Algorithm 3. The analysis transfers HB and CP ordering from \( m \) to \( T \) for all \( \rho \); adds new \( \text{CCP}(\rho) \) 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 \( e = \text{acq}(m) \) by \( T \), any \( e_\rho \) that was HB or CP ordered to \( m \) is now also HB or CP ordered, respectively, to \( T \). If \( m \in \text{CP}(\rho) \), then \( e_\rho \xrightarrow{\text{CP}} \text{acq}(m) \), and thus the analysis adds \( T \) to \( \text{CP}(\rho) \) (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 \( m : n^k \in \text{CCP}(\rho) \), then that means \( e_\rho \xrightarrow{\text{CP}} \text{rel}(m)^i \), if \( \text{acq}(m)^j \xrightarrow{\text{CCP}} \text{rel}(n)^j \), where \( n^j \) is an ongoing critical section. After the current event \( \text{acq}(m)^j \), since HB right-composes with CP, \( e_\rho \xrightarrow{\text{CP}} \text{acq}(m)^j \) if \( \text{acq}(n)^k \xrightarrow{\text{CCP}} \text{rel}(n)^j \). Thus, the analysis adds \( T : n^k \) to \( \text{CCP}(\rho) \).

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

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

6.4 Handling Release Events

At a program’s release of a lock, i.e., \( e = \text{rel}(m)^i \) 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 \( \text{CCP}(\rho) \) elements: the pre-release algorithm adds elements to \( \text{CCP}(\rho) \), 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 \( \sigma : m^j \in CP(\rho) \), \( \sigma \) is CP ordered to \( e_\rho \) if \( acq(m)^j \xrightarrow{CP} rel(m)^j \).

Line 4 checks this condition; if it is true, line 5 adds \( \sigma \) to \( CP(\rho) \).

Algorithm 4

Pre-release of \( m \) by \( T \)

1: for all \( \rho \) do
2: \( \triangleright \) Transfer CP ordering according to Rule (b); transfer CP
3: for all \( \sigma, j \mid \sigma : m^j \in CP(\rho) \) do
4: if \( \exists l \geq j \mid T : n^k \in CP(m^l) \) then
5: \( CP(\rho)^+ \leftarrow CP(\rho)^+ \cup \{\sigma\} \)
6: end if
7: \( CP(\rho)^+ \leftarrow CP(\rho)^+ \cup \{\sigma : n^k\} \)
8: \( \triangleright \) No effect if \( \exists k' < k \mid \sigma : n^{k'} \in CP(\rho)^+ \)
9: end for
10: 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 \( \sigma : m^j \in CP(\rho) \) (line 3) and \( T : n^k \in CP(m^l) \) (line 7), \( \sigma \) is CP ordered to \( e_\rho \) if \( acq(m)^j \xrightarrow{CP} rel(m)^j \), which in turn is true if \( acq(n)^k \xrightarrow{CP} rel(n)^h \) (where \( n^h \) is the ongoing critical section on \( n \)). Thus, \( \sigma \) is CP ordered to \( e_\rho \) if \( acq(n)^k \xrightarrow{CP} rel(n)^h \), so line 8 adds \( \sigma : n^k \) to \( CP(\rho) \).

We note that although the analysis could in theory detect the above CP and CCP orderings earlier than at \( rel(m)^j \), it is critical that the analysis detect them no later than \( rel(m)^j \), since the release algorithm removes all \( \sigma : m^j \) 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 \( e = rel(m)^j \) by \( T \), if \( T = HB \) or CP ordered to \( e_\rho \), m

is now HB or CP ordered to \( e_\rho \). Lines 3–12 show how the analysis transfers ordering from \( T \) to \( m \). If \( T = CP(\rho) \), 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

Pre-release of \( m \) by \( T \)

1: for all \( \rho \) do
2: \( \triangleright \) Transfer ordering from \( T \) to \( m \)
3: if \( T \in CP(\rho) \) then
4: \( CP(\rho)^+ \leftarrow CP(\rho)^+ \cup \{m\} \)
5: end if
6: for all \( n^k \mid n \neq m \land T : n^k \in CP(\rho) \) do
7: \( CP(\rho)^+ \leftarrow CP(\rho)^+ \cup \{m : n^k\} \)
8: end for
9: if \( T \in HB(\rho) \) then
10: \( HB(\rho)^+ \leftarrow HB(\rho)^+ \cup \{m\} \)
11: \( \triangleright \) No effect if \( \exists i < j \mid m^i \in HB(\rho)^+ \)
12: end if
13: \( \triangleright \) Remove CCP elements conditional on \( m \)
14: \( CP(\rho)^+ \leftarrow CP(\rho)^+ \setminus \{\{\sigma : m^j \in CP(\rho]\} \)
15: end for

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

Removing CCP ordering. Line 14 removes all CCP elements dependent on \( m \), i.e., all \( \sigma : m^j \) elements from \( CP(\rho) \), 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., \( acq(m)^j \xrightarrow{CP} rel(m)^j \) does not imply that \( \sigma \) is CP ordered to \( e_\rho \). 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 \( acq(m)^2 \), the analysis detects only PO and HB ordering. At \( acq(m)^2 \), in addition to transferring HB ordering from \( m \) to \( T_2 \), the analysis adds \( T_2 : m^1 \) to all \( CP(\rho) \) such that \( e_\rho \xrightarrow{HB} acq(m)^2 \).

At \( wr(x)^2 \), it is not knowable from the events so far that \( wr(x)^1 \xrightarrow{CP} wr(x)^2 \). Raptor adds \( x : m^1 \) to \( CP(x)^1 \), indicating that \( rel(m)^1 \xrightarrow{CP} acq(m)^2 \) implies \( wr(x)^1 \xrightarrow{CP} wr(x)^2 \). At \( wr(y)^2 \), the analysis applies Rule (a) of the CP
Removing obsolete variable locksets and detecting CP-races.

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

Algorithm 6 Detect and remove obsolete locksets and report CP-races for \( x' \)

1: if \( \text{wr}(x') \) has executed then
2: if \( \xi \in \text{CP}(x') \cup \text{PO}(x') \) then
3: \( \triangleright \) No CP-race between \( \text{wr}(x') \) and \( \text{wr}(x') \)
4: \( \text{PO}(x') \) \( \rightarrow \emptyset \)
5: \( \text{CP}(x') \) \( \rightarrow \emptyset \)
7: Report CP-race between \( \text{wr}(x') \) and \( \text{wr}(x') \)
8: end if
8: end

Raptor can say for certain that \( \text{wr}(x') \rightarrow \text{wr}(x') \) if \( \xi \in \text{CP}(x') \) is definitely impossible. According to Raptor's analysis (Algorithms 1–5), \( \xi \in \text{CP}(x') \) is possible only if \( \exists m \in \text{CP}(x') \cup \text{PO}(x') \) (line 5). If not, the analysis knows \( \text{wr}(x') \rightarrow \text{wr}(x') \) 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. As a result, according to the CCP-constraint invariant (Figure 3), line 6 in Algorithm 6 evaluates to true for every \( x' \) by the time the execution terminates. Thus for every \( \text{wr}(x') \rightarrow \text{wr}(x') \) pair for which Raptor has not already ruled out a CP-race (via line 2 evaluating to true), Raptor eventually reports a CP-race.

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.
Algorithm 7 Detect and remove obsolete locksets for $m^j$

1: if $rel(m^j)$ has executed then
2: \( \begin{align*}
\text{if } & \exists \rho \quad | \quad \rho \neq m^j \land (m^j \in HB(\rho) \lor m^j \in HB(\rho) \lor \\
& (3i \leq j \mid \sigma : m^i \in CCP(\rho)) \land m \notin CP(\rho) \text{ then} \\
\end{align*} \)
3: \( PO(m^j)^+ \leftarrow \emptyset , \) $HB(m^j)^+ \leftarrow \emptyset ,$
4: \( CP(m^j)^+ \leftarrow \emptyset , \) $CCP(m^j)^+ \leftarrow \emptyset$
5: end if
6: 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 $\rho$, $m^j$ appears in $HB(\rho)$ or an element dependent on $m^j$ (such that $i \leq j$) appears in $CCP(\rho)$—unless $m \in CP(\rho)$, in which case $m^j$ is not needed by $\rho$’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 $CP$ or $PO$, or there is a CP-race. In contrast, reads are not totally ordered in a CP-race-free execution. For a write $wr(x)^i$, Raptor must check if each thread’s following read that comes before $wr(x)^{i+1}$ is CP ordered with $wr(x)^i$. Raptor extends the special element $\xi$ to one special element per thread, $\xi_T$, which denotes that $T$’s read of $x$ is ordered with the the preceding writing $wr(x)^i$.

If a thread $T$ performs multiple reads to $x$ between $wr(x)^i$ and $wr(x)^{i+1}$, 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 $wr(x)^i$ and $wr(x)^{i+1}$ is in a CP-race with $wr(x)^i$. Thus, we extend the variable removal and CP-race check from Algorithm 6 to check for $\xi_T$ for each thread $T$ that reads $x$ between $wr(x)^i$ and $wr(x)^{i+1}$.

Detecting read–write races. For each thread that reads $x$ between $wr(x)^i$ and $wr(x)^{i+1}$ 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 $wr(x)^i$ and $wr(x)^{i+1}$. Raptor uses $x^i_T$’s locksets to detect ordering from $T$’s read of $x$ to $wr(x)^{i+1}$, 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 $wr(x)^i$ and $wr(x)^{i+1}$, 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 ($wr(x)^{i+1}$).

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 $\xi$ and $\xi_T$ elements that need to be added to $CP(x^i) \cup PO(x^i)$ (or $CP(x^i_T) \cup PO(x^i_T)$) before the analysis can be certain that no CP-race exists. Once this number reaches zero, the implementation concludes that $wr(x)^i$ (or $rd(x)^i_T$) does not race with a following element, and thus its locksets are obsolete.

The implementation tracks the numbers of remaining $\xi : m^i$ and $\xi_T : m^i$ elements in $CP(x^i)$ (or $CP(x^i_T)$). If any of these numbers drop to zero, and the corresponding $\xi$ or $\xi_T$
element is not in \( CCP(x_i) \) (or \( CCP(x_{i+}) \)), 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 \( \rho \) with active locksets, it precomputes the following information once: (1) the maximum \( j \) such that \( \exists I \geq j | T \in CP(m^l) \) and (2) for each \( j \), the set of \( n^k \) such that \( \exists I \geq j | T : n^k \in CP(m^l) \). 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 9.12bach, 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.

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 statically unique races.

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

**Table 1.** Dynamic execution characteristics and reported HB- and CP-races 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.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Distance range for each static CP-race</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTPServer</td>
<td>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</td>
</tr>
<tr>
<td>hedc</td>
<td>2,067–3,774</td>
</tr>
<tr>
<td>tsp</td>
<td>177–14,303; 939–939</td>
</tr>
<tr>
<td>jython</td>
<td>157,736–157,739</td>
</tr>
<tr>
<td>xalan</td>
<td>4–16; 15–32; 15–48; 18–52; 134–149</td>
</tr>
</tbody>
</table>
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, and completely. An evaluation of our implementation of Raptor shows that it can analyze full program executions, and completely. A few billion lines of code later: Using static analysis to find bugs in the real world. CACM, 53(2):66–75, 2010.

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


A. Examples Executions Analyzed by Raptor

Example involving CP established before accesses execute.

In Figure 5, rel(m)$^1 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{acq}(n)^2$, following through the logic: rel(n)$^1 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{acq}(n)^2$ by Rule (a), which implies acq(m)$^1 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{rel}(m)^2$ by Rule (c) (since acq(m)$^1 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{rel}(n)^3$ and acq(n)$^2 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{rel}(m)^2$), which implies rel(m)$^1 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{acq}(m)^2$ by Rule (b). Thus wr(x)$^1 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{acq}(m)^2$ by applying Rule (c) to rel(m)$^1 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{acq}(m)^2$ (since wr(x)$^1 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{rel}(m)^1$ and acq(m)$^2 \xrightarrow{2} \text{CP} \xrightarrow{0} \text{wr}(x)^2$).

Since the critical section on n$^2$ finishes before 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 acq(q)$^2$, T3 : q$^1 \in \text{CP}(m)$. The CCP ordering composes with HB, so that T4 : n$^1 \in \text{CP}(m)$ before rel(n)$^2$, and thus T4 \in \text{CP}(m)$ after rel(n)$^3$. In a similar vein, Raptor adds T4 : m$^2 \to \text{CP}(x)$ at acq(m)$^2$, which allows the analysis to add T4 to CP(x)$^1$ at rel(m)$^2$.

Example involving CP ordering established only after CP-ordered variable accesses have executed. In Figure 6,
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 rel(n) → acq(m), which implies rel(n) → acq(m) → rel(m). When the execution finally executes rel(m), Raptor concludes from ξ: m1 ∈ CCP(z1) and T3 ∈ CP(m1) that it should add ξ to CP(z1).

Example that is subtly 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
<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>Lockset After</th>
</tr>
</thead>
<tbody>
<tr>
<td>wr(z)</td>
<td>acq(m)</td>
<td>rel(m)</td>
<td>acq(q)</td>
<td>wr(z)</td>
</tr>
<tr>
<td>HB(z^1) = {T1}</td>
<td>HB(m^1) = {T1}</td>
<td>HB(z^1) = HB(m^1) = {m^1, T1}</td>
<td>HB(q^1) = {T1}</td>
<td>HB(q^1) = {q^1, T1}</td>
</tr>
<tr>
<td>acq(q)</td>
<td>rel(q)</td>
<td>acq(q)</td>
<td>wr(z)</td>
<td>CCP(m) = CCP(q^1) = {T1, \xi^1}</td>
</tr>
<tr>
<td>HB(z^1) = HB(m^1) = {m^1, q^1, T1}</td>
<td>HB(z^1) = HB(m^1) = {m^1, q^1, T1, T2}</td>
<td>CCP(z^1) = CCP(m^1) = {T3: m^1}</td>
<td>CCP(z^1) = CCP(m^1) = {T3: m^1}</td>
<td></td>
</tr>
<tr>
<td>CCP(z^1) = CCP(m^1) = {T3: m^1}</td>
<td>CCP(z^1) = CCP(m^1) = {T3: m^1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(z^1) = CCP(m^1) = {T3: m^1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(z^1) = CCP(m^1) = {T3: m^1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(z^1) = CCP(m^1) = {T3: m^1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. In this execution, \( wr(z)^1 \xrightarrow{CP} wr(z)^2 \) because of a CP ordering that has not yet been established when \( wr(z)^2 \) executes.
This example is subtly different from the example in Figure 6: it flips the order of T1’s critical sections on m and q. As a result, \( \text{acq}(m)^1 \xrightarrow{CP} \text{rel}(m)^2 \).

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>Lockset After</th>
</tr>
</thead>
<tbody>
<tr>
<td>wr(z)</td>
<td>acq(q)</td>
<td>rel(q)</td>
<td>acq(m)</td>
<td>rel(m)</td>
</tr>
<tr>
<td>HB(z) = {T1}</td>
<td>HB(q) = {T1}</td>
<td>HB(z) = HB(q) = {q_1, T1}</td>
<td>HB(m) = {T1}</td>
<td></td>
</tr>
<tr>
<td>HB(z) = {m_1, T1}</td>
<td>HB(z) = HB(q) = {q_1, m_1, T1}</td>
<td>HB(z) = HB(q) = {q_1, m_1, T1, T2}</td>
<td>CCP(z) = CCP(q) = {T2: q_1}, HB(q) = {T2}</td>
<td></td>
</tr>
<tr>
<td>rel(m)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB(z) = {m_1, T1}</td>
<td>HB(z) = HB(q) = {q_1, m_1, T1}</td>
<td>HB(z) = HB(q) = {q_1, m_1, T1, T2}</td>
<td>HB(m) = {m_1, T1, T3}, HB(m) = {T3}</td>
<td></td>
</tr>
<tr>
<td>CCP(z) = CCP(q) = CCP(m) = {T3: m_1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acq(o)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB(o) = {T2}</td>
<td>HB(o) = {m_1, o_1, T1, T3}</td>
<td>HB(o) = {o_1, T3, T4}, HB(o) = {T4}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(z) = CCP(q) = CCP(m) = {o: m_1, T3: m_1, T4: m_1, T4: o_1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rel(o)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB(o) = {o_2, T4}</td>
<td>HB(o) = {o_2, T4}</td>
<td>CCP(z) = CCP(q) = CCP(m) = {o: m_1, T3: m_1, T4: m_1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>acq(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB(z) = {q_1, m_1, n_1, o_1, T1, T2, T3, T4, \xi}</td>
<td>HB(z) = {q_1, m_1, n_1, o_1, T1, T2, T3, T4, \xi}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(z) = CCP(q) = CCP(m) = {o: m_1, T3: m_1, T4: m_1, \xi: m_1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wr(y)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB(y) = {n_1, T2, T3, \xi}</td>
<td>CCP(z) = CCP(q) = CCP(m) = {T3: n_1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(y) = {T3: n_1, \xi: n_1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rel(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(q) = CCP(y) = {n_1, T2, T3, \xi}</td>
<td>CCP(y) = {n_1, T2, T3, \xi}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(y) = {n_1, T2, T3, \xi}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acq(m)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(q) = CCP(y) = {n_1, T2, T3, \xi}</td>
<td>CCP(q) = CCP(y) = {n_1, T2, T3, \xi}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(y) = {n_1, T2, T3, \xi}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCP(y) = {n_1, T2, T3, \xi}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7. This example is subtly different from the example in Figure 6: it flips the order of T1’s critical sections on m and q. As a result, \( \text{acq}(m)^1 \xrightarrow{CP} \text{rel}(m)^2 \).
implies \( \operatorname{wr}(x)^1 \xrightarrow{CP} \operatorname{wr}(x)^2 \) by Rule (c). The example shows that even when the critical section on \( m^2 \) ends, i.e., after \( \operatorname{rel}(m)^2 \) executes, it is not yet possible to determine that \( \operatorname{acq}(m)^1 \xrightarrow{CP} \operatorname{rel}(m)^2 \). This fact is not knowable until after \( \operatorname{acq}(a)^3 \) executes. In particular, at \( \operatorname{rel}(m)^2, \) \( \operatorname{acq}(m)^1 \xrightarrow{CP} \operatorname{rel}(m)^2 \) is dependent on whether \( \operatorname{rel}(a)^1 \xrightarrow{CP} \operatorname{acq}(a)^2 \), as represented by \( T3: o^1 \xrightarrow{} \operatorname{CP}(m^1) \) after \( \operatorname{rel}(m)^2 \).

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., \( e = \operatorname{wr}(x)^i \) by thread \( T \). Let \( e^+ \) be the event immediately after \( e \) in the observed total order (\( \prec_{\text{trace}} \)). 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., \( e = \operatorname{acq}(m)^i \) by thread \( T \). Let \( e^+ \) be the event immediately after \( e \) in the observed total order (\( \prec_{\text{trace}} \)). 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., \( e = \operatorname{rel}(m)^i \) 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., \( e = \operatorname{rel}(m)^i \) by thread \( T \). Let \( e^+ \) be the event immediately after \( e \) in the observed total order (\( \prec_{\text{trace}} \)). 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 \( \text{HB} \) [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 \( \sigma : m^j \) only exist during critical sections on \( m \):

1. At \( \operatorname{rel}(m)^i \), Algorithm 5 removes all elements \( \sigma : m^j \) from every \( \operatorname{CP}(\rho)^+ \).
2. When \( \nexists \sigma' : m^j \in \operatorname{CP}(\rho) \), only Algorithm 3 (for \( \operatorname{acq}(m)^{k} \)) adds elements of the form \( \sigma : m^j \) to \( \operatorname{CP}(\rho)^+ \).

**Proof of Lemma 1.**

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

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

- Let \( LHS = \operatorname{CP}(\rho) \cup \{ \sigma \mid \exists n^k : \sigma : n^k \in \operatorname{CP}(\rho) \} \).
- Let \( \text{LHS}^+ = \operatorname{CP}(\rho)^+ \cup \{ \sigma \mid \exists n^k : \sigma : n^k \in \operatorname{CP}(\rho)^+ \} \).
- Let \( \text{RHS} = \{ \sigma \mid \exists e' : \text{appl}(\sigma, e') \land e_\rho \xrightarrow{CP} e' \prec_{\text{trace}} e \} \).
- Let \( \text{RHS}^+ = \{ \sigma \mid \exists e' : \text{appl}(\sigma, e') \land e_\rho \xrightarrow{CP} e' \prec_{\text{trace}} e^+ \} \).

Suppose Figure 3’s invariants hold before \( e \), i.e., suppose \( \text{LHS} = \text{RHS} \). 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 \( \text{LHS}^+ = \text{RHS}^+ \), we first show \( \text{LHS}^+ \subseteq \text{RHS}^+ \) (subset direction) and then \( \text{LHS}^+ \supseteq \text{RHS}^+ \) (superset direction).

**Subset direction:** Let \( \sigma \in \text{LHS}^+ \).

Either \( \sigma \in \text{CP}(\rho)^+ \) or \( \exists n^k : \sigma : n^k \in \text{CP}(\rho)^+ \) and \( \exists e : \text{appl}(\sigma, e) \land e_\rho \xrightarrow{CP} e' \prec_{\text{trace}} e^+ \) (or both):

**Case 1:** \( \sigma \in \text{CP}(\rho)^+ \)

If \( \sigma \in \text{CP}(\rho) \), then by the inductive hypothesis, \( \sigma \in \text{RHS} \), i.e., \( \exists e' : \text{appl}(\sigma, e') \land e_\rho \xrightarrow{CP} e' \prec_{\text{trace}} e \).

Since \( e \prec_{\text{trace}} e^+ \), \( \exists e' : \text{appl}(\sigma, e') \land e_\rho \xrightarrow{CP} e' \prec_{\text{trace}} e^+ \), so \( \sigma \in \text{RHS}^+ \).

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

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

By the inductive hypothesis on \( m^j \in \text{HB}(x^{i-1}) \) (HB-critical-section invariant), \( \text{acq}(m)^j \xrightarrow{PO} \text{wr}(x)^{i-1} \xrightarrow{PO} \text{rel}(m)^j \xrightarrow{\text{wr}(x)^{i-1}} \prec_{\text{trace}} e \). Because \( m \in \text{heldBy}(T) \) and \( T \notin \text{PO}(x^{i-1}) \), let \( k \) be such that \( \text{acq}(m)^k \xrightarrow{PO} e \xrightarrow{PO} \text{rel}(m)^k \). By the definition of CP (Rule (a)), \( \text{rel}(m)^j \xrightarrow{CP} \text{acq}(m)^k \). By the definition of CP
<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>Lockset After</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{wr}(x)^1 )</td>
<td>( \text{acq}(m)^1 )</td>
<td>( \text{rel}(m)^1 )</td>
<td>( \text{acq}(o)^1 )</td>
<td>( \text{rel}(o)^1 )</td>
</tr>
<tr>
<td>( \text{HH}(x^1) = {T1} )</td>
<td>( \text{HH}(m^1) = {T1} )</td>
<td>( \text{HH}(x^1) = \text{HH}(m^1) = {m^1, T1} )</td>
<td>( \text{HH}(o^1) = {T1} )</td>
<td>( \text{HH}(o^1) = {o^1, T1} )</td>
</tr>
<tr>
<td>( \text{HH}(q^1) = {T1} )</td>
<td>( \text{HH}(m^1) = {T1} )</td>
<td>( \text{HH}(o^1) = {o^1, T1} )</td>
<td>( \text{HH}(q^1) = {T1} )</td>
<td>( \text{HH}(y^1) = {q^1, T1} )</td>
</tr>
<tr>
<td>( \text{HH}(o^1) = {o^1, q^1, T1} )</td>
<td>( \text{HH}(q^1) = {q^1, T1} )</td>
<td>( \text{HH}(o^1) = {o^1, q^1, T1} )</td>
<td>( \text{HH}(q^1) = {q^1, T1} )</td>
<td>( \text{HH}(y^1) = {q^1, T1} )</td>
</tr>
<tr>
<td>( \text{HH}(x^1) = \text{HH}(m^1) = {m^1, o^1, q^1, T1} )</td>
<td>( \text{HH}(o^1) = {o^1, q^1, T1} )</td>
<td>( \text{HH}(x^1) = \text{HH}(m^1) = {m^1, o^1, q^1, T1} )</td>
<td>( \text{HH}(o^1) = {o^1, q^1, T1, T2} )</td>
<td>( \text{HH}(o^1) = {o^1, q^1, T1, T2} )</td>
</tr>
<tr>
<td>( \text{HH}(x^1) = \text{HH}(m^1) = {m^1, o^1, q^1, T1} )</td>
<td>( \text{HH}(o^1) = {o^1, q^1, T1} )</td>
<td>( \text{HH}(o^1) = {o^1, q^1, T1} )</td>
<td>( \text{HH}(o^1) = {o^1, q^1, T1} )</td>
<td>( \text{HH}(o^1) = {o^1, q^1, T1} )</td>
</tr>
<tr>
<td>( \text{ACQP}(x^1) = \text{CCP}(m^1) = {T1, o^1} )</td>
<td>( \text{CCP}(o^1) = {T2, o^1} )</td>
<td>( \text{CCP}(o^1) = {T2, o^1} )</td>
<td>( \text{CCP}(o^1) = {T2, o^1} )</td>
<td>( \text{CCP}(o^1) = {T2, o^1} )</td>
</tr>
<tr>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
</tr>
<tr>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
<td>( \text{CCP}(x^1) = \text{CCP}(m^1) = {T2, o^1, T1, T2, T3} )</td>
</tr>
<tr>
<td>( \text{HH}(o^1) = {o^1, T1} )</td>
<td>( \text{HH}(y^1) = {q^1, T1} )</td>
<td>( \text{HH}(o^1) = {o^1, T1} )</td>
<td>( \text{HH}(y^1) = {q^1, T1} )</td>
<td>( \text{HH}(o^1) = {o^1, T1} )</td>
</tr>
</tbody>
</table>

Figure 8. \( \text{wr}(x)^1 \xrightarrow{\text{CP}} \text{wr}(x)^2 \) due to CCP dependencies that Raptor necessarily resolves lazily.
Case 2: Let $n^k$ and $j$ be such that $\sigma : n^k \in CCP(\rho)^+ \land rel(n^k) \xrightarrow{CP} acq(n^j) \xrightarrow{\text{trace}} e^+$. 

If $\sigma : n^k \notin CCP(\rho) \land rel(n^k) \xrightarrow{CP} acq(n^j) \xrightarrow{\text{trace}} e$ then by the inductive hypothesis (CP invariant), $\sigma \in RHS$, i.e., $\exists e' | appl(\sigma, e') \land e_\rho \xrightarrow{CP} e' \xrightarrow{\text{trace}} e$. Since $e \xrightarrow{\text{trace}} e'$ and $e \neq acq(n^j)$, note that $e = wr(x)^j$ 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 $\sigma \notin RHS$), $\exists e' | appl(\sigma, e') \land e_\rho \xrightarrow{CP} e' \xrightarrow{\text{trace}} e$ which implies $appl(\sigma, e') \land e_\rho \xrightarrow{CP} e$. By the definition of CP, since $e = wr(x)^j$ is an acquire operation, $\exists e' | thr(e') = T \land e_\rho \xrightarrow{CP} e' \xrightarrow{\text{trace}} e$, i.e., $T \in RHS$. According to the definition of $appl(\sigma, e)$, either (1) $\sigma = T$ or (2) $\sigma = \xi$ (note that $e = wr(x)^j$ is $e^+$ from Figure 3). However, $\sigma \notin RHS$, so $\sigma = \xi$. Therefore $\rho = x^{i-1}$ and $e_\rho = wr(x)^{i-1}$, and $wr(x)^{i-1} \xrightarrow{CP} e$. 

By the inductive hypothesis on $T \in RHS$ (CP invariant), $T \in CP(x^{i-1})$ or $\exists n^k | T : n^k \in CCP(x^{i-1}) \land \exists j | rel(n^k) \xrightarrow{CP} acq(n^j) \xrightarrow{\text{trace}} e$ (or both).

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

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

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

Since $T : n^k \in CCP(x^{i-1})$ matches line 18, line 19 adds $\xi : n^j$ to $CP(x^{i-1})^+$. Since $e \xrightarrow{\text{trace}} e^+$, $\exists j | rel(n^k) \xrightarrow{CP} acq(n^j) \xrightarrow{\text{trace}} e^+$. Thus $\xi \in RHS^+$. Since $LHS^+ \subseteq RHS^+ \land LHS^+ \supseteq RHS^+$, $LHS^+ = RHS^+$. 

Proof of Lemma 2.

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

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

$LHS = CP(\rho)^+ \cup \{ \sigma | \exists n^k | \sigma : n^k \in CCP(\rho)^+ \land rel(n^k) \xrightarrow{CP} acq(n^j) \xrightarrow{\text{trace}} e \}$.

$LHS^+ = CP(\rho)^+ \cup \{ \sigma | \exists n^k | n^k \in CCP(\rho)^+ \land \exists j | rel(n^k) \xrightarrow{CP} acq(n^j) \xrightarrow{\text{trace}} e \}$.

$LHS = \{ \sigma | \exists e' | appl(\sigma, e') \land e_\rho \xrightarrow{CP} e' \xrightarrow{\text{trace}} e \}$

$LHS^+ = \{ \sigma | \exists e' | appl(\sigma, e') \land e_\rho \xrightarrow{CP} e' \xrightarrow{\text{trace}} e \}$.

Since 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 $\exists n^k | \sigma : n^k \in CCP(\rho)^+ \land rel(n^k) \xrightarrow{CP} acq(n^j) \xrightarrow{\text{trace}} e^+$ (or both):

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

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

Otherwise ($\sigma \notin CP(\rho)$), Algorithm 3 must add $\sigma$ to $CP(\rho)^+$, which can happen only at line 4. Line 4 adds $T$ to $CP(\rho)^+$, so $\sigma = T$. Since line 4 executes, line 3 must evaluate to true for $\rho$, so $m \in CP(\rho)$. By the inductive hypothesis (CP invariant), $\exists j | e_\rho \xrightarrow{CP} rel(m)^j \xrightarrow{\text{trace}} e$.

By the definition of $CP$, $e_\rho \xrightarrow{CP} e$ (since $e = acq(m)^j$).

Because $thr(e) = T \land e_\rho \xrightarrow{CP} e \xrightarrow{\text{trace}} e^+$, therefore $T \in RHS^+$. 

Case 2: Let $n^k$ and $j$ be such that $\sigma : n^k \in CCP(\rho)^+ \land rel(n^k) \xrightarrow{CP} acq(n^j) \xrightarrow{\text{trace}} e^+$.
If \( \sigma : n^k \in CCP(\rho) \land \text{rel}(n)^k \overset{CP}{\rightarrow} \text{acq}(n)^j \prec_{\text{trace}} e \), then by the inductive hypothesis, \( \sigma \in RHS \), i.e., \( \exists e' \mid \text{app}(\sigma, e') \land e_p \overset{CP}{\rightarrow} e' \prec_{\text{trace}} e \). Since \( e \prec_{\text{trace}} e' \), and then \( e_p \overset{CP}{\rightarrow} e' \prec_{\text{trace}} e \), then \( \sigma \in RHS^+ \).

Otherwise, \( -\sigma : n^k \in CCP(\rho) \land \text{rel}(n)^k \overset{CP}{\rightarrow} \text{acq}(n)^j \prec_{\text{trace}} e \). In fact, we can conclude that \( \sigma : n^k \notin CCP(\rho) \) using the following reasoning. If \( -\text{rel}(n)^k \overset{CP}{\rightarrow} \text{acq}(n)^j \prec_{\text{trace}} e \), then \( \text{acq}(n)^j \notin_{\text{trace}} e \) (since \( \text{rel}(n)^k \overset{CP}{\rightarrow} \text{acq}(n)^j \)), and \( \exists e \in \text{acq}(n)^j \) (since \( \text{acq}(n)^j \notin_{\text{trace}} e \) and \( e \) immediately precedes \( e' \)), which by the inductive hypothesis (CCP-constraint invariant) gives \( \sigma : n^k \notin CCP(\rho) \). Since \( \sigma : n^k \notin CCP(\rho) \) regardless, Algorithm 3 must add \( \sigma : n^k \rightarrow CCP(\rho)^+ \), which can happen only at line 7 or 13 (or both).

- If line 7 adds \( \sigma : n^k \rightarrow CCP(\rho)^+ \), then \( \sigma = T \) and \( m : n^k \rightarrow CCP(\rho) \) (line 6). By the inductive hypothesis (CP invariant), \( \exists e : e_p \overset{CP}{\rightarrow} \text{rel}(m)^j \prec_{\text{trace}} e \). By the definition of CP, \( e_p \overset{CP}{\rightarrow} e \) since \( e = \text{acq}(m)^j \). Since \( \text{thr}(e) = T \land e_p \overset{CP}{\rightarrow} e \prec_{\text{trace}} e^+ \), therefore \( T \in RHS^+ \).

- If line 13 adds \( \sigma : n^k \rightarrow CCP(\rho)^+ \), then \( n^k \in HB(\rho) \) or \( m^k \in HB(\rho) \) according to line 10. Furthermore, \( n = m \) and \( \sigma = T \). By the inductive hypothesis (HB, HB-index, and HB-critical-section invariants), \( e_p \overset{HB}{\rightarrow} \text{rel}(m)^j \prec_{\text{trace}} e \). Since we know \( \text{rel}(n)^k \overset{CP}{\rightarrow} \text{acq}(n)^j \prec_{\text{trace}} e^+ \) (from the beginning of Case 2) and \( e = \text{acq}(m)^j \) (since \( n = m \)), therefore \( e_p \overset{CP}{\rightarrow} e \) by the definition of CP (regardless of whether \( j = i \) or \( j < i \)). Since \( \text{thr}(e') = T \land e_p \overset{CP}{\rightarrow} e^+ \prec_{\text{trace}} e \), therefore \( T \in RHS^+ \).

**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, \( (\sigma \in RHS^+ \land \sigma \notin RHS) \), \( \exists e' \mid \text{app}(\sigma, e') \land e_p \overset{CP}{\rightarrow} e' \prec_{\text{trace}} e \) and \( \exists \bar{e}' \mid \text{app}(\sigma, \bar{e}') \land e_p \overset{CP}{\rightarrow} \bar{e}' \prec_{\text{trace}} \bar{e} \), which implies \( \text{app}(\sigma, e') \land e_p \overset{CP}{\rightarrow} e \) and thus \( \sigma = T \). Since \( e_p \overset{CP}{\rightarrow} e \) and \( \exists \bar{e}' \mid \text{thr}(\bar{e}') = T \land e_p \overset{CP}{\rightarrow} \bar{e}' \prec_{\text{trace}} \bar{e} \), by the definition of CP, \( \exists e : e_p \overset{CP}{\rightarrow} \text{rel}(m)^j \overset{HB}{\rightarrow} e \) or \( \exists e : e_p \overset{HB}{\rightarrow} \text{rel}(m)^j \overset{CP}{\rightarrow} e \) (or both).

**Case 1:** \( \exists e : e_p \overset{CP}{\rightarrow} \text{rel}(m)^j \overset{HB}{\rightarrow} e \)

By the inductive hypothesis on \( e_p \overset{CP}{\rightarrow} \text{rel}(m)^j \) (CP invariant), \( m \in CP(\rho) \) or \( \exists j : m^k \in CCP(\rho) \land \exists e : e_p \overset{CP}{\rightarrow} \text{acq}(n)^j \prec_{\text{trace}} e \) (or both).

**Case 1a:** \( m \in CP(\rho) \)

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

**Case 1b:** Let \( n^k \) be such that \( m : n^k \in CCP(\rho) \land \exists j : \text{rel}(n)^k \overset{CP}{\rightarrow} \text{acq}(n)^j \prec_{\text{trace}} e \).

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

**Case 2:** \( \exists e : e_p \overset{HB}{\rightarrow} \text{rel}(m)^j \overset{CP}{\rightarrow} e \)

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

Since \( LHS^+ \subseteq RHS^+ \land LHS^+ \supseteq RHS^+ \), \( LHS^+ = RHS^+ \).

**Proof of Lemma 3.**

Let \( e = \text{rel}(m)^j \) by thread \( T \). Let \( \rho \) be any lockset owner. Let \( e_p \) be the event corresponding to \( \rho \), i.e., \( e_p = \text{wr}(\rho)^k \) if \( \rho = \rho^k \), or \( e_p = \text{acq}(m)^k \) if \( \rho = m^k \).

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

Let \( LHS = \text{CP}(\rho) \cup \{ \sigma : (\exists n^k : n^k \in CCP(\rho) \land \exists l \mid \text{rel}(n)^k \overset{CP}{\rightarrow} \text{acq}(n)^j \prec_{\text{trace}} e) \} \).

Let \( LHS^+ = \text{CP}(\rho)^+ \cup \{ \sigma : (\exists n^k : n^k \in CCP(\rho)^+ \land \exists l \mid \text{rel}(n)^k \overset{CP}{\rightarrow} \text{acq}(n)^j \prec_{\text{trace}} e) \} \).

Let \( RHS = \{ \sigma : (\exists e' \mid \text{app}(\sigma, e') \land e_p \overset{CP}{\rightarrow} e' \prec_{\text{trace}} e) \} \).

Let \( RHS^+ = \text{RHS} \).

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 \( RHS^+ \supseteq RHS^+ \) (superset direction).

**Subset direction:** Let \( \sigma \in RHS^+ \).

Either \( \sigma \in CP(\rho)^+ \) or \( \exists n^k : n^k \in CCP(\rho)^+ \land \exists l \mid \text{rel}(n)^k \overset{CP}{\rightarrow} \text{acq}(n)^j \prec_{\text{trace}} e \) (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 \(\sigma\) to \(CP(\rho)^+\), which can happen only at line 5. Let \(j\) be such that \(\sigma : m^j \in CCP(\rho)\) (line 3) and \(T : m^j \in CP(m^j)\). By the inductive hypothesis on \(T \in CP(m^j)\) (CP invariant), let \(e'\) be such that \(\text{thr}(e') = T \land acq(m)^j \xrightarrow{CP} e' \prec_{\text{trace}} e\). Since \(e' \prec_{\text{trace}} e\), the definition of CB. Thus \(acq(m)^j \xrightarrow{CP} e\) and therefore \(\text{rel}(m)^j \xrightarrow{CP} acq(m)^j\) by the definition of CP. Because of those facts, together with \(\sigma : m^j \in CCP(\rho)\), 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 \(\sigma : n^k \in CCP(\rho)\) and \(\text{rel}(n^k) \xrightarrow{CP} acq(n)^k\). By the inductive hypothesis (CP invariant), \(\sigma \in LHS\). Thus \(RHS = RHS^+\), \(\sigma \in RHS^+\).

Otherwise, \(-\sigma : n^k \in CCP(\rho) \land \text{rel}(n^k) \xrightarrow{CP} acq(n)^k\). We know \(\text{rel}(n^k) \xrightarrow{CP} acq(n)^k\). Since \(n^k \notin CCP(\rho)\), we need to show that \(\sigma \notin CCP(\rho)\). Let \(j\) be such that \(\sigma : m^j \in CCP(\rho)\) (line 3) and \(T : n^k \in CCP(n^k)\). Since \(\sigma \notin CCP(\rho)\), \(\sigma \in LHS\). Thus \(RHS = RHS^+\), \(\sigma \in RHS^+\).

Superset direction: Let \(\sigma \in RHS^+\).

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

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

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

Let \(LHS = CP(\rho)^+ \cup \{e \mid \text{rel}(m)^j \xrightarrow{CP} acq(n)^j \prec_{\text{trace}} e\}\). Let \(LHS^+ = CP(\rho)^+ \cup \{e \mid \text{rel}(m)^j \xrightarrow{CP} acq(n)^j \prec_{\text{trace}} e^+\}\).

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

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

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.

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 \(\sigma \in RHS^+\).

Either \(\sigma \in CP(\rho)^+\) or \(\exists n^k \mid \sigma : n^k \in CCP(\rho)\) and \(\text{rel}(n^k) \xrightarrow{CP} acq(n)^k\).

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

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

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

Case 2: Let \(n^k\) and \(j\) be such that \(\sigma : n^k \in CCP(\rho)\) and \(\text{rel}(n^k) \xrightarrow{CP} acq(n)^k\).

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

Otherwise \(-\sigma : n^k \in CCP(\rho) \land \exists j \mid \text{rel}(n^k) \xrightarrow{CP} acq(n)^k \prec_{\text{trace}} e\). However, \(\exists j \mid \text{rel}(n^k) \xrightarrow{CP} acq(n)^k \prec_{\text{trace}} e\).
Superset direction: Let \( \sigma / \) Case 2:

Contradicting \( e \) therefore \( \exists \sigma \) or \( \exists \sigma \) and \( \exists \sigma \) or \( \exists \sigma \) because \( e = \rel(m) \) and \( e = \rel(m) \) since \( e = \rel(m)^+ \) and \( e = \rel(m)^+ \) and \( e = \rel(m)^+ \).

Case 2a: \( T \in \CP(\rho) \)

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

Case 2b: Let \( n \) and \( j \) such that \( T : n \in \CP(\rho) \land \rel(n)^k \CP \Rightarrow \acq(n)^j \Rightarrow \trace e \).

If \( n \neq m \), then because \( T : n \in \CP(\rho) \) matches line 6 of Algorithm 5, line 7 adds \( m \in \CP(\rho)^+ \).

Otherwise \( n = m \), so \( m \in \CP(\rho)^+ \) because Algorithm 5 removes \( m \) from \( \CP(\rho)^+ \) at line 14. Since \( \rel(m)^k \CP \Rightarrow \acq(m)^j \Rightarrow \trace e \), \( \acq(m)^j \CP \Rightarrow \acq(m)^j \Rightarrow \trace e \) by the definition of \( \CP \) (since \( e = \rel(m)^j \)). By the inductive hypothesis (CP invariant) and Lemma 5, \( T \in \CP(m^k) \) or \( \exists \rho \mid o \neq m \land T : o \in \CP(m^k) \land \exists h \mid \rel(o)^i \CP \Rightarrow \acq(o)^h \Rightarrow \trace e \).

Case 2bi: \( T \in \CP(m^k) \)

In the pre-release algorithm (Algorithm 4), line 4 evaluated to true for \( T : m^k \in \CP(\rho) \) (line 3). Thus line 5 executed, so \( T \in \CP(\rho) \).

In the release algorithm (Algorithm 5), line 3 evaluates to true for \( \rho \). Therefore Algorithm 5’s line 4 executes, so \( m \in \CP(\rho)^+ \) and thus \( m \in \LHS^+ \).

Case 2bii: Let \( o \) and \( h \) be such that \( T : o \in \CP(m^k) \land \rel(o)^i \CP \Rightarrow \acq(o)^h \Rightarrow \trace e \).

In the pre-release algorithm (Algorithm 4), line 8 executed for \( T : m^k \in \CP(\rho) \) (line 3) and \( T : o \in \CP(m^k) \) (line 7). Line 8 added \( o \) to \( \CP(\rho) \). The release algorithm (Algorithm 5) does not remove \( o \) because \( o \neq m \) (line 14).

Therefore \( \sigma : o \in \CP(\rho)^+ \) and thus \( \sigma \in \LHS^+ \).

Case 2: \( \sigma \notin \RHS \)

Thus \( \exists \rho \mid \app(\sigma, e^\prime) \land e_{\rho} \CP \Rightarrow e^\prime \Rightarrow \trace e^+ \) but \( \exists \rho \mid \app(\sigma, e^\prime) \land e_{\rho} \CP \Rightarrow e^\prime \Rightarrow \trace e^+ \). Therefore \( e_{\rho} \CP \Rightarrow e \). Since \( e = \rel(m)^j \), \( \sigma = m \) or \( \sigma = T \). But \( \sigma = T \) would imply \( \exists \rho \mid thr(e^\prime) = T \land e_{\rho} \CP \Rightarrow e^\prime \Rightarrow \trace e \), contradicting \( \sigma \notin \RHS \). Therefore \( \sigma = m \).

By the definition of \( \CP \) and the fact that \( e = \rel(m)^j \), let \( e^\prime \) be such that \( thr(e^\prime) = T \land e_{\rho} \CP \Rightarrow e^\prime \Rightarrow \trace e \). By the inductive hypothesis (CP invariant), \( T \in \CP(\rho) \) or \( \exists \rho \mid T : n^\rho \in \CP(\rho) \land \exists j \mid \rel(n)^k \CP \Rightarrow \acq(n)^j \Rightarrow \trace e \).

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 \( m \) and \( j \), \( d(m^j) \), as the following:

- \( 0 \) if \( \exists x, g, h, j, j^\prime \mid j \leq j^\prime < i \land \acq(m)^{j^\prime} \Rightarrow \PO \Rightarrow \wr(x)^g \Rightarrow \rel(m)^{i} \Rightarrow \PO \Rightarrow \acq(m)^{i} \Rightarrow \PO \Rightarrow \wr(x)^h \Rightarrow \PO \Rightarrow \rel(m)^{j^\prime} \).
- \( +1 \) if \( d(m^j) \Rightarrow \acq(m)^{j^\prime} \Rightarrow \PO \Rightarrow \rel(n)^k \CP \Rightarrow \acq(n)^{j^\prime} \Rightarrow \PO \Rightarrow \rel(m)^{j^\prime} \).

Since \( \LHS^+ \subseteq \RHS^+ \land \LHS^+ \supseteq \RHS^+ \land \LHS^+ = \RHS^+ \).

The proof of Lemma 4 relied on an additional lemma, which we state and prove below. First we state a new definition.
Lemma 5. Let $e$ be any release event, i.e., $e = \text{rel}(m)^j$ executed by thread $T$. If there exists $j$ such that $\text{acq}(m)^j \xrightarrow{CP} e$, 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:

- $\exists j' \geq j \mid e \in \text{CP}(m')$
- $\exists j' \geq j \mid \exists n^k \mid n \neq m \land T : n^k \in \text{CCP}(m') \land \exists n \mid \text{rel}(n)^k \xrightarrow{CP} n \prec j' \prec j' < d(n^k \sim n) < d(m^j \sim m^i)$

Proof of Lemma 5.

Let $e$ be any release event, i.e., $e = \text{rel}(m)^j$ executed by thread $T$. Let $j$ be such that $\text{acq}(m)^j \xrightarrow{CP} e$. 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 $d(m^j \sim m^i)$.

**Base case.** If $d(m^j \sim m^i) = 0$, then by the definition of CP-distance, $\exists x,y,z \mid x \in \text{CP}(m')$ where $T' = \text{thr}(\text{rel}(m)^j)$. Because $\text{rel}(m)^j \xrightarrow{HB} \text{rel}(m)^i$, and the algorithms propagate HB-ordered events through CP locksets, $T' \in \text{CP}(m^i)$ at $e$.

**Inductive step.** $d(m^j \sim m^i) > 0$

Inductive hypothesis: Suppose the lemma holds true for all $n^k$ and $s$ such that $d(n^k \sim n^s) < d(m^j \sim m^i)$.

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

**Case 1:** $\text{rel}(m)^j \prec \text{trace} \text{rel}(n)^i$

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

**Case 2:** $\text{rel}(n)^i \prec \text{trace} \text{rel}(m)^j$

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

- $d(o^j \sim o^g)$ is as minimal as possible;
- $\text{rel}(o)^g < \text{trace} \text{rel}(m)^j$; and
- there exists $f' \geq f$ and $s$ such that, at $\text{rel}(o)^g$,
  - $\exists j' \geq j \mid \sigma \in \text{CCP}(m')$
  - $\exists e^j \mid \text{acq}(m)^j \xrightarrow{CP} e^j \prec \text{trace} \text{rel}(m)^j \land e$.

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

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

- $T' \in \text{CP}(o^j)$ or
- $\exists q^c \mid q \neq o \land \exists T : q^c \in \text{CCP}(o^j) \land \exists d \mid \text{rel}(q)^c \xrightarrow{CP} \text{acq}(q)^i \xrightarrow{HB} \text{rel}(o)^g \land d(q^c \sim q^d) < d(o^j \sim o^g)$

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

**Case 2a:** $T' \in \text{CP}(o^j)$

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

**Case 2b:** Let $q^c$ and $d$ be such that, at $\text{rel}(o)^g$, $q \neq o \land \exists T : q^c \in \text{CCP}(o^j) \land \exists d \mid \text{rel}(q)^c \xrightarrow{CP} \text{acq}(q)^i \xrightarrow{HB} \text{rel}(o)^g \land d(q^c \sim q^d) < d(o^j \sim o^g)$.

Thus $\text{acq}(q)^i \prec \text{trace} \text{rel}(o)^g \prec \text{trace} \text{rel}(q)^d$.

If $\text{rel}(q)^d \prec \text{trace} \text{rel}(m)^j$, then that would violate the stipulation above that $d(o^j \sim o^g)$ is minimal. Thus either $\text{rel}(m)^j = \text{rel}(q)^g$ or $\text{rel}(m)^j \prec \text{trace} \text{rel}(q)^d$.

**Case 2b(i):** $\text{rel}(m)^j = \text{rel}(q)^g$

Thus $m = q$ and $i = d$, but $c < j$ since $d(m^c \sim m^i) = d(q^c \sim q^d) < d(m^j \sim m^i)$.

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

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

- $T' \in \text{CP}(m^c)$

The pre-release algorithm (Algorithm 4) adds $T$ to $\text{CP}(m^i)$ at line 5 because $T : m^c \in \text{CCP}(m^j)$ (matching line 3) and $T \in \text{CP}(m^c)$ (matching line 4).

- $\exists r^a \mid r \neq m \land T : r^a \in \text{CCP}(m^c) \land \exists b \mid \text{rel}(r)^a \xrightarrow{CP} \text{acq}(r)^b \prec \text{trace} \text{rel}(m)^j \land d(r^a \sim r^b) < d(m^i \sim m^i)$

The pre-release algorithm (Algorithm 4) adds $T : r^b$ to $\text{CCP}(m^j)$ at line 8 because
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 \( e' \) \( \triangleright_{\text{trace}} e \) (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(\( e \)) for “fake” initial accesses is non-empty, i.e., \( PO(x^0) = \{ \xi \} \) 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., \( wr(x)^0 \xrightarrow{PO} wr(x)^1 \).

**Inductive step.** Suppose the invariants in Figure 3 hold immediately before an event \( e \). If \( e = wr(x)^i \), the invariants hold after \( e \), by Lemma 1. If \( e = acq(m)^i \), the invariants hold after \( e \), by Lemma 2. If \( e = rel(m)^i \), 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 (\( \Rightarrow \)) and backward direction (\( \Leftarrow \)) 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 \( wr(x)^i \xrightarrow{CP} wr(x)^i+1 \) \( \wedge \) \( wr(x)^i \xrightarrow{PO} wr(x)^i+1 \), but \( \xi \notin CP(x^i) \cup PO(x^i) \) at program termination. According to Theorem 1, by the CP and PO invariants, \( \xi \notin CP(x^i) \cup PO(x^i) \) 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 \( \xi \notin CP(x^i) \cup PO(x^i) \) at program termination, but \( wr(x)^i \xrightarrow{CP} wr(x)^i+1 \) \( \vee \) \( wr(x)^i \xrightarrow{PO} wr(x)^i+1 \).

Since \( \xi \notin PO(x^i) \) at program termination, according to Theorem 1, by the PO invariant, \( wr(x)^i \xrightarrow{PO} wr(x)^i+1 \). Thus \( wr(x)^i \xrightarrow{CP} wr(x)^i+1 \). By the CP invariant, at program termination,

\[ \xi \in CP(x^i) \vee (\exists n^k | \xi : n^k \in CCP(x^i) \wedge \exists j | rel(n)^j \xrightarrow{CP} acq(n)^j \xrightarrow{\text{trace}} e^\Omega) \]

where \( e^\Omega \) represents a final “program termination” event. Since \( \xi \notin CP(x^i) \), \( \exists n^k | \xi : n^k \in CCP(x^i) \). By the CCP-constraint invariant, \( \exists l | acq(n)^j \xrightarrow{\text{trace}} e^\Omega \wedge rel(n)^j \xrightarrow{\text{trace}} e^\Omega \). But a thread releases all of its held locks before program termination (Section 8), so \( rel(n)^j \xrightarrow{\text{trace}} e^\Omega \) 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

\[ wd(x_i^\tau) \text{ by } T \]

1: \( \triangleright \) Apply Rule (a)
2: \( \triangleright \) Write-Read Race
3: for all \( m \in \text{heldBy}(T) \) do
4: \( \text{if } \exists j \exists h \mid m'_h \in HB(x^h) \land T \notin PO(x^h) \text{ then} \)
5: \( CP(m^h)^+ \leftarrow CP(m^h)^+ \cup \{ T \} \)
6: end if
7: end for
8: \( \triangleright \) Add termination CCP elements
9: if \( T \in PO(x^i) \) then
10: \( PO(x^i)^+ \leftarrow PO(x^i)^+ \cup \{ \xi_T \} \)
11: end if
12: if \( T \in HB(x^i) \) then
13: \( HB(x^i)^+ \leftarrow HB(x^i)^+ \cup \{ \xi_T \} \)
14: end if
15: if \( T \in CP(x^i) \) then
16: \( CP(x^i)^+ \leftarrow CP(x^i)^+ \cup \{ \xi_T \} \)
17: end if
18: for all \( m \mid T : m \in CCP(x^i) \) do
19: \( CCP(x^i)^+ \leftarrow CCP(x^i)^+ \cup \{ \xi_T : m \} \)
20: end for
21: \( \triangleright \) Initialize locksets for \( x^i_t. \)
22: \( \triangleright \) Reset lockets if \( x^i_t \) already exists.
23: if \( T \in PO(x^i_t) \) then
24: \( PO(x^i_t) \leftarrow \emptyset, HB(x^i_t) \leftarrow \emptyset \)
25: \( CP(x^i_t) \leftarrow \emptyset, CCP(x^i_t) \leftarrow \emptyset \)
26: end if
27: \( PO(x^i_t) \leftarrow \{ T \} \)
28: \( HB(x^i_t) \leftarrow \{ T \} \cup \{ m'_m \mid m'' \in \text{heldBy}(T) \} \)

Algorithm 9

\[ \text{Modified } wr(x^i) \text{ by } T \]

1: \( \triangleright \) First apply Rule (a)
2: \( \triangleright \) Write-Write and Read-Write Race
3: for all \( m \in \text{heldBy}(T) \) do
4: \( \text{if } \exists j \exists h \mid m'_h \in HB(x^h) \land T \notin PO(x^h) \text{ then} \)
5: \( CP(m^h)^+ \leftarrow CP(m^h)^+ \cup \{ T \} \)
6: end if
7: for all Threads \( t \) do
8: \( \text{if } \exists j \exists h \mid m'_h \in HB(x^h) \land T \notin PO(x^h) \text{ then} \)
9: \( CP(m^h)^+ \leftarrow CP(m^h)^+ \cup \{ T \} \)
10: end if
11: end for
12: end for
13: \( \triangleright \) Add termination CCP elements
14: if \( T \in PO(x^{i-1}) \) then
15: \( PO(x^{i-1})^+ \leftarrow PO(x^{i-1})^+ \cup \{ \xi \} \)
16: end if
17: if \( T \in HB(x^{i-1}) \) then
18: \( HB(x^{i-1})^+ \leftarrow HB(x^{i-1})^+ \cup \{ \xi \} \)
19: end if
20: if \( T \in CP(x^{i-1}) \) then
21: \( CP(x^{i-1})^+ \leftarrow CP(x^{i-1})^+ \cup \{ \xi \} \)
22: end if
23: for all \( m \mid T : m \in CCP(x^{i-1}) \) do
24: \( CCP(x^{i-1})^+ \leftarrow CCP(x^{i-1})^+ \cup \{ \xi : m \} \)
25: end for
26: for all Threads \( t \) do
27: \( \text{if } T \in PO(x^{i-1}_t) \text{ then} \)
28: \( PO(x^{i-1}_t)^+ \leftarrow PO(x^{i-1}_t)^+ \cup \{ \xi \} \)
29: end if
30: \( \text{if } T \in HB(x^{i-1}_t) \text{ then} \)
31: \( HB(x^{i-1}_t)^+ \leftarrow HB(x^{i-1}_t)^+ \cup \{ \xi \} \)
32: end if
33: \( \text{if } T \in CP(x^{i-1}_t) \text{ then} \)
34: \( CP(x^{i-1}_t)^+ \leftarrow CP(x^{i-1}_t)^+ \cup \{ \xi \} \)
35: end if
36: for all \( m \mid T : m \in CCP(x^{i-1}_t) \) do
37: \( CCP(x^{i-1}_t)^+ \leftarrow CCP(x^{i-1}_t)^+ \cup \{ \xi : m \} \)
38: end for
39: end for
40: \( \triangleright \) Initialize locksets for \( x^i \)
41: \( PO(x^i) \leftarrow \{ T \} \)
42: \( HB(x^i) \leftarrow \{ T \} \cup \{ m'_m \mid m'' \in \text{heldBy}(T) \} \)