Automatic Verification of Data Race Freedom in Device Drivers∗ (Extended Abstract)

Pantazis Deligiannis1 and Alastair F. Donaldson2

1 Imperial College London p.deligiannis@imperial.ac.uk
2 Imperial College London alastair.donaldson@imperial.ac.uk

Abstract

Device drivers are notoriously hard to develop and even harder to debug. They are typically prone to many serious issues such as data races. In this paper, we present static pair-wise lock set analysis, a novel sound verification technique for proving data race freedom in device drivers. Our approach not only avoids reasoning about thread interleavings, but also allows the reuse of existing successful sequential verification techniques.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Device Drivers, Verification, Concurrency, Data Races

Digital Object Identifier 10.4230/OASIcs.ICCSW.2014.1

1 Introduction

Device drivers are complex pieces of system-level software responsible for the interaction between an operating system and any hardware devices that are attached to a computer [10]. Drivers are notoriously hard to develop and even harder to debug. Even after a device driver has shipped, it is typically prone to many serious errors [6, 31]. Regarding concurrency bugs, a recent study [28] found that they account for 19% of the total bugs in Linux drivers, showcasing their significance. The majority of these concurrency bugs were found to be data races or deadlocks in various configuration functions and hot-plugging handlers.

The main focus of this paper is on data races, which can lead to nondeterministically occurring bugs that can be challenging to reproduce, isolate and debug. Well-known Linux kernel analysers, such as sparse [8], coccinelle [22] and lockdep [9], have successfully found deadlocks in kernel code, but are typically unable to detect data races. Techniques such as [1, 7, 25, 12, 14, 16, 27, 17] have been used to analyse Linux and Windows device drivers, but primarily focus on sequential program properties. Furthermore, most previous techniques that attempt to reason about thread interleavings face significant scalability issues, because of the exponentially large state-space of realistic concurrent programs [19].

This work-in-progress paper presents static pair-wise lock set analysis, a novel technique for automatically verifying data race freedom in device drivers. The key idea behind our approach is that a driver can be proven free from data races by (i) deriving a sound sequential model that over-approximates the originally concurrent driver, (ii) instrumenting it for lock set analysis and race checking, and (iii) asserting that all accesses to the same shared resource are protected by at least one common lock. The immediate benefit is that our approach not only avoids reasoning about thread interleavings, and thus has the potential to scale well, but also allows the reuse of existing successful sequential verification techniques.

∗ This work is part of the research project “Automatic Synthesis of High-Assurance Device Drivers” and is generously funded by a gift from Intel Corporation.

© Pantazis Deligiannis and Alastair F. Donaldson; licensed under Creative Commons License CC-BY
Imperial College Computing Student Workshop (ICCSW’14).
Editors: Rumyana Neykova and Nicholas Ng; pp. 1–4
OpenAccess Series in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
2 Static Pair-Wise Lock Set Analysis

Our technique involves reasoning about the lock sets of a driver. Lock set analysis has its roots in Eraser [29], a dynamic data race detector that tracks the set of locks that consistently protect a memory location during program execution. If that set ever becomes empty, the tool reports a potential data race. This is because an inconsistent lock set suggests that a memory location can be accessed simultaneously by two or more threads.

Eraser avoids reasoning about arbitrary thread interleavings, and thus can scale well in realistic concurrent programs, but suffers from imprecision (i.e. can report false bugs), because a violation of the locking discipline does not always correspond to a real data race [29, 23, 20, 11, 13]. Furthermore, as a dynamic analyser, Eraser’s bug finding ability is limited to the execution paths that the tool explores. To counter the second limitation, we apply the idea of Eraser’s lock set analysis in a static verification context.

Static pair-wise lock set analysis begins by performing two-thread reduction, an abstraction that removes all but two arbitrary threads, each running an entry point of the originally concurrent driver. This reduction was inspired by a reduction to two threads employed in the GPUVerify tool for verification of GPU kernels [5, 2]. The technique then proceeds with pair-wise sequentialisation, which combines the two arbitrary threads in a single sequential pair. The pair is finally instrumented for lock set analysis with assertions that check if each memory location is consistently protected by at least one common between the two threads lock. This process repeats until all pairs of entry points have been sequentialised. To achieve soundness, each time an entry point performs a read access to a shared resource, we return a nondeterministic value. This over-approximates any effects from all the unmodeled threads on the driver shared state.

We have prototyped this technique in Whoop, a practical tool for automatic concurrency verification of Linux drivers written in C [15]. Whoop initially compiles the driver source code, together with an environmental model, to LLVM-IR using Clang/LLVM [18]. The program is then compiled to the Boogie [3] verification language using SMACK [26], an LLVM to Boogie translator which can efficiently model heap manipulating programs. Next, the Boogie program is sequentialised and instrumented, using static pair-wise lock set analysis. The abstract program is finally send to the Boogie verifier, which generates verification conditions [4] and discharges them to a theorem prover. Successful verification implies that the original driver is free of data races, while an error denotes a potential data race.

The main limitation of our approach is that it can potentially report many false positives, as we over-approximate the shared state. To tackle this problem, we plan to investigate (i) invariant generation for taming our coarse abstraction and (ii) counterexample feasibility checking to evaluate if a reported bug is real or spurious.

3 Related Work

Notable previous works on static analysis for race detection include the static analysers Warlock [30] and LockLint [21], which, however, heavily rely on user annotations. Whoop does not require any source code modifications, and thus can be applied with zero effort.

Most related to our work are the static lock set analysers RELAY [32] and Locksmith [24]. Both tools, though, have significant limitations. RELAY uses unsound post-analysis filters to limit the false positives, but these can filter out true races. Although Locksmith successfully detected data races in 7 medium-sized Linux device drivers, it reported a significant number of false positives. The authors also reported that Locksmith was unable to run on several large programs, showcasing its limited scalability.


4 Automatic Verification of Data Race Freedom in Device Drivers


