Computer Languages, Systems & Structures 47 (2017) 62–76

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Effective abstractions for verification under relaxed memory models

Andrei Dan[a], Yuri Meshman[b], Martin Vechev[a], Eran Yahav[b]

[a] ETH Zurich, Switzerland
[b] Technion, Israel

ARTICLE INFO

Article history:
Received 3 June 2015
Received in revised form
16 February 2016
Accepted 17 February 2016
Available online 2 March 2016

Keywords:
Abstract interpretation
Relaxed memory models

ABSTRACT

We present a new abstract interpretation based approach for automatically verifying concurrent programs running on relaxed memory models. Our approach is based on three key insights: (i) Although the behaviors of relaxed memory models (e.g., TSO and PSO) are naturally captured by store buffers, directly using such encodings substantially decreases the accuracy of program analysis due to *shift operations* on buffer contents. The scalability and accuracy of program analysis can be greatly improved by eliminating the expensive shifting of store buffer contents, and we present a new abstraction of the memory model that accomplishes this goal. (ii) The precision of the analysis can be further improved by an encoding of store buffer sizes using leveraged knowledge of the abstract interpretation domain. (iii) A novel source-to-source transformation that realizes the above two techniques makes it possible to use of state-of-the-art analyzers directly under sequential consistency (SC): given a program P and a relaxed memory model M, it produces a new program P_M where the behaviors of P running on M are over-approximated by the behavior of P_M running on SC.

We implemented our approach and evaluated it on a set of finite and infinite-state concurrent algorithms under two memory models: Intel's x86 TSO and PSO. Experimental results indicate that our technique achieves better precision and efficiency than prior work: we can automatically verify algorithms with fewer fences, faster and with lower memory consumption.

1. Introduction

To improve performance, modern hardware architectures support relaxed memory models. A relaxed memory model allows the underlying architecture to reorder memory operations and execute them non-atomically. As a result, a concurrent program can have additional behaviors that would not be possible to obtain under the intuitive, sequentially consistent setting [20]. These additional relaxed behaviors complicate the task of reasoning about the correctness of the program, manually and automatically.

This necessitates the development of new, scalable and precise analysis techniques for automatic verification of (potentially infinite-state) concurrent programs running on relaxed memory models. Automatic verification in this setting is

E-mail addresses: andrei.dan@inf.ethz.ch (A. Dan), yurime@cs.technion.ac.il (Y. Meshman), martin.vechev@inf.ethz.ch (M. Vechev), yahave@cs.technion.ac.il (E. Yahav).

http://dx.doi.org/10.1016/j.cl.2016.02.003
1477-8424/© 2016 Elsevier Ltd. All rights reserved.

a challenging problem as the relaxed memory model can significantly increase the number and diversity of new behaviors, which in turn affects the overall precision and scalability of the analysis.

Our approach: We present a new analysis system for verifying concurrent programs running on relaxed memory models such as Intel's ×86 TSO and PSO buffered memory models. Our approach relies first, on a new abstraction of the memory model that eliminates some of the expensive work in managing the store buffers, thus significantly reducing the analysis effort and improving its precision. This abstraction is also directly applicable and useful for other verification frameworks, both finite and infinite-state (e.g., bounded model checking, abstract interpretation and predicate abstraction). Our approach also leverages knowledge of the particular program analysis used in this work (abstract interpretation with numerical domains) to encode the size of the store buffers in a way that reduces the loss of precision under that abstract domain.

We incorporate the above two ideas into a robust analyzer. The analyzer uses a source-to-source transformation that enables direct use of existing program analyzers under sequential consistency for verifying concurrent programs running on relaxed memory models. That is, given a program P, a specification S and a memory model M, the transformation automatically constructs a new program P_M such that if $P_M \models_{SC} S$ then $P \models_M S$. The program P_M contains *an abstraction of the relaxed behaviors induced by M*, thereby ensuring the soundness of the approach.

While prior works [12,4,22] also suggest source-to-source transformations, we show experimentally that our approach is more precise and efficient: it enables verification of (infinite-state) concurrent algorithms that prior work cannot verify, and for programs where prior work succeeds, our approach is faster and requires less memory. This work also represents one of the few studies on using abstract interpretation for verifying properties of infinite-state concurrent programs running on relaxed memory models. Moreover, our approach requires no user annotations.

Main contributions: The main contributions of this paper are:

- A new abstraction for the store buffers of the memory model that eliminates expensive shifting of buffer contents. This abstraction reduces the workload on subsequent program analyzers and improves their scalability and precision.
- A source-to-source transformation that realizes the new abstraction (and the memory model effects), producing a program that can be soundly analyzed with verifiers for sequential consistency. The translation also leverages knowledge of the underlying abstract domain in order to encode the size of the store buffers in a way which reduces the overall loss of analysis precision.
- A complete implementation of the approach integrated with ConcurInterproc [16], a tool based on abstract interpretation [10] with numerical abstract domains that can analyze infinite-state concurrent programs under sequential consistency.
- A thorough empirical evaluation on a range of challenging concurrent algorithms. Experimental results indicate that our technique is superior in both precision and efficiency to prior work and enables verification, for the first time, of several concurrent algorithms running on Intel's ×86 TSO and PSO memory models.

2. Overview

In this section we illustrate our approach on a running example. The goal of this section is to give some intuition about and informal understanding of the work. Full formal details are provided in later sections.

To understand our approach, consider the concurrent program shown in Fig. 1. It consists of two threads that share the integer variables X and Y (variables a and b are local to each thread). The figure also shows an assertion which holds once both threads have completed their execution, namely that $X+Y \geq 2$. Our objective is to verify that the program satisfies this assertion under relaxed memory models such as Intel's ×86 TSO and PSO.

2.1. Relaxed behaviors

In the example in Fig. 1, Thread 1 can execute the statements at labels 1 and 2 in the opposite order. Similarly, Thread 2 can execute the statements at labels 1 and 2 in the opposite order due to the nature of relaxed memory models such as TSO. Relaxed models such as TSO allow program statements to be executed out of order, resulting in behaviors not possible under sequential consistency. Under TSO, a store and a load (by the same thread) accessing different memory locations are allowed to be reordered. Therefore, after both threads execute the statements at labels 1 and 2, one can end up in the state X

```
  initial values: X=0   Y=0

 Thread 1:                              Thread 2:

 1: X = 1                               1: Y = 1
 2: a = Y                               2: b = X
 3: X = a + 1                           3: Y = b + 1
 4: fence                               4: fence
 5:                                     5:
```

Spec: $((pc_1 = 5) \wedge (pc_2 = 5)) \Rightarrow (X + Y \geq 2)$

Fig. 1. Example program.