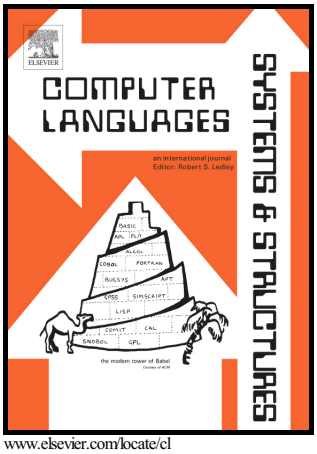

Refinement of structural heuristics for model checking of concurrent programs through data mining

Reed Milewicz, Peter Pirkelbauer

PII: S1477-8424(16)30079-3
DOI: http://dx.doi.org/10.1016/j.cl.2016.06.001
Reference: COMLAN219

To appear in: *Computer Language*

Received date: 4 December 2015
Revised date: 19 May 2016
Accepted date: 10 June 2016

Cite this article as: Reed Milewicz and Peter Pirkelbauer, Refinement o structural heuristics for model checking of concurrent programs through dat mining, *Computer Language,* http://dx.doi.org/10.1016/j.cl.2016.06.001

Refinement of Structural Heuristics for Model Checking of Concurrent Programs through Data Mining

Reed Milewicz[1] and Peter Pirkelbauer[1]

University of Alabama at Birmingham
Birmingham, AL 35223
rmmilewi@cis.uab.edu pirkelbauer@uab.edu

Abstract. Detecting concurrency bugs in multi-threaded programs through model-checking is complicated by the combinatorial explosion in the number of ways that different threads can be interleaved to produce different combinations of behaviors. At the same time, concurrency bugs tend to be limited in their scope and scale due to the way in which concurrent programs are designed, and making visible the rules that govern the relationships between threads can help us to better identify which interleavings are worth investigating. In this work, patterns of read-write sequences are mined from a single execution of the target program to produce a quantitative, categorical model of thread behaviors. This model is exploited by a novel structural heuristic. Experiments with a proof-of-concept implementation, built using Java Pathfinder and WEKA, demonstrate that this heuristic locates bugs faster and more reliably than a conventional counterpart.

1 Introduction

For the purposes of this work, we define a *thread* as a sequence of instruction blocks that are linked to one another by a chain of continuations. A thread is the smallest unit of execution that can be handled independently by a scheduler, and threads are presented here as a general-purpose solution for managing concurrency. A multi-threaded program executing on a multi-core architecture consists of the birth, life, and dissolution of arbitrarily many threads operating in parallel across a fixed number of cores. In the analyses of multi-threaded, concurrent programs, our task is to characterize interactions between threads and other threads, and between threads and the execution environment. Meanwhile, *concurrency bug* is an umbrella term for classes of bugs which arise as a consequence of improper synchronization between threads over the use of shared resources, such as deadlocks, race conditions, and atomicity violations. Our goal is to detect concurrency bugs that could interfere with the proper execution of a concurrent program through careful analysis.

Recent history is replete with high profile news stories of concurrency bugs causing dramatic and spectacular failures. One especially powerful example, the