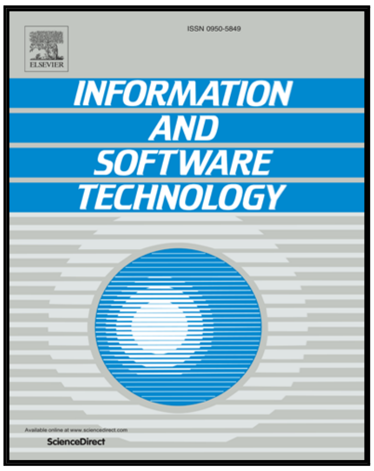

Diversity Driven Adaptive Test Generation for Concurrent Data Structures

Linhai Ma, Peng Wu, Tsong Yueh Chen

PII:	S0950-5849(18)30135-6
DOI:	10.1016/j.infsof.2018.07.001
Reference:	INFSOF 6014

To appear in:	*Information and Software Technology*

Received date:	7 August 2017
Revised date:	23 April 2018
Accepted date:	2 July 2018

Please cite this article as: Linhai Ma, Peng Wu, Tsong Yueh Chen, Diversity Driven Adaptive Test Generation for Concurrent Data Structures, *Information and Software Technology* (2018), doi: 10.1016/j.infsof.2018.07.001

Diversity Driven Adaptive Test Generation for Concurrent Data Structures

Linhai Ma[a,b], Peng Wu[a,b,*], Tsong Yueh Chen[c]

[a] *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences*
[b] *University of Chinese Academy of Sciences*
[c] *Department of Computer Science and Software Engineering, Swinburne University of Technology*

Abstract

Context: Testing concurrent data structures remains a notoriously challenging task, due to the nondeterminism of multi-threaded tests and the exponential explosion on the number of thread schedules.

Objective: We propose an automated approach to generate a series of concurrent test cases in an adaptive manner, i.e., the next test cases are generated with the guarantee to discover the thread schedules that have not yet been activated by the previous test cases.

Method: Two diversity metrics are presented to induce such adaptive test cases from a static and a dynamic perspective, respectively. The static metric enforces the diversity in the program structures of the test cases; while the dynamic one enforces the diversity in their capabilities of exposing untested thread schedules. We implement three adaptive test generation approaches for C/C++ concurrent data structures, based on the state-of-the-art active testing engine Maple.

Results: We then report an empirical study with 9 real-world C/C++ concurrent data structures, which demonstrates the efficiency of our test generation approaches in terms of the number of thread schedules discovered, as well as the time and the number of tests required for testing a concurrent data structure.

Conclusion: Hence, by using diverse test cases derived from the static and dynamic perspectives, our adaptive test generation approaches can deliver a more efficient coverage of the thread schedules of the concurrent data structure under test.

Keywords: Concurrent Data Structures, Test Case Diversity, Test Case Generation, Active Testing, Adaptive Random Testing

1. Introduction

Concurrent data structures are key components for the development of concurrent software, because shared objects are often implemented with concurrent data structures. A concurrent data structure encapsulates self-synchronization mechanisms to coordinate the simultaneous accesses of multiple threads to a shared object. Presumably concurrent operations (e.g., method calls) of the shared object can be equivalently serialized to access the shared object sequentially. This assumption eases the development of concurrent software to a great extent, because developers just need to write sequential programs separately for individual threads without any reference to inter-thread synchronization. To be precise, the term *concurrent data structure* throughout the paper refers to a concurrent implementation of a data structure (e.g., a concurrent implementation of queue), instead of its sequential specification (e.g., a first-in-first-out sequential specification). Object-oriented programming languages have already provided typical concurrent data structures for multi-threaded programming, such as the *java.util.concurrent* package in Java. Plenty of open-source or proprietary concurrent data structures are also available to support the development of concurrent applications in practice.

Thus, the reliability of a concurrent data structure is vital to the correctness of a concurrent program that uses it. However, a concurrent data structure is no less error-prone than a concurrent program. Furthermore, testing a concurrent data structure may raise more challenges than testing a concurrent program because a concurrent data structure cannot run on its own. A test case of a concurrent data structure is a multi-threaded program that makes simultaneous accesses to the concurrent data structure. Obviously, the test case space is infinite in general. Therefore, it gives rise to a fundamental problem on the automated generation of effective multi-threaded test cases (i.e., concurrent programs) for concurrent data structures, let alone the challenge due to the nondeterministic execution of a test case and the exponential number of possible thread interleavings.

Active testing has established a coverage-guided paradigm for efficiently exploring the possible thread interleavings of a concurrent program under test. Typically, the state-of-the-art active testing tool Maple [1] works in two stages as follows: *profiling* and *active testing*. At the profiling stage, the concurrent program is profiled by running on its own, and the resulting concurrent executions are collected to discover the interleaving instances that encompass the shared-memory accesses by the multiple threads of the concurrent program. These include the interleaving instances that have taken place during the profiling executions, and the untested interleaving instances predicted

*Corresponding author
Email addresses: mlh@ios.ac.cn (Linhai Ma), wp@ios.ac.cn (Peng Wu), tychen@swin.edu.au (Tsong Yueh Chen)