2016 IEEE International Conference on Software Testing, Verification and Validation

Testing Concurrent Software Systems

Francesco A. Bianchi
Università della Svizzera italiana (USI) - Faculty of Informatics
Lugano, Switzerland
Email: francesco.bianchi@usi.ch
Research Advisor: Mauro Pezz è

***Abstract*—Many modern software systems are intrinsically concurrent, consisting of multiple execution flows that progress simultaneously. Concurrency introduces new challenges to the testing process, since some faults can manifest only under specific interleavings of the execution flows. Thus, testing concurrent systems requires not only to explore the space of possible inputs, but also the space of possible interleavings, which can be intractable also for small programs. Most of the research on testing concurrent systems has focused exclusively on selecting interleavings that expose potentially dangerous patterns, such as data races and atomicity violations. However, by ignoring the high-level semantics of the system, these patterns may miss relevant faults and generate many false positive alarms. In this PhD work we aim to define a novel testing technique that (i) offers a holistic approach to select both test inputs and interleavings, and (ii) exploits high-level semantic information on the program to guide the selection and to improve the accuracy of the state-of-the-art techniques.**

I. Problem Statement

Concurrent systems are becoming more and more relevant in the everyday life. Common examples are mobile applications, high-performance systems and peer-to-peer applications. Concurrent systems include multiple execution flows that progress simultaneously interacting with each other [1].

The behavior of a concurrent system depends not only on the program inputs, but also on the *interleaving* of the execution flows occurring at runtime. This key feature has a relevant impact on the testing activity of such systems: concurrent systems may expose *concurrency faults*, that are faults that depend on the interleaving of otherwise correct code. Concurrency faults are hard to find and reproduce, since they can manifest only under specific interleavings. This means that the testers should not only identify a representative sample of the input space, but also select a significant subset of the interleaving space. This is problematic, since the interleaving space grows exponentially with the number of execution flows and the number of statements of each of them and can thus be huge and impossible to explore exhaustively.

In the last decades, several techniques have been proposed for testing concurrent systems. The key activities when dealing with testing concurrent systems are: (i) generating test cases to stimulate the system and (ii) selecting interleavings to be executed for such test cases.

The problem of generating test cases for concurrent systems has received little attention so far, and the few techniques that have been proposed rely on random approaches to generate inputs to stimulate the target system [2].

Conversely, most of the current techniques for testing concurrent systems focus exclusively on selecting interleavings. In this context, two approaches have emerged: *systematic exploration* and *property-driven* approaches.

Systematic exploration aims to explore the space of the interleavings as broadly as possible. Some approaches adopt a brute force approach that attempts to explore *all* possible interleavings [3], and suffer from scalability problems which make them unpractical for testing real-world systems. Other approaches exploit partial order reduction techniques or heuristics to bound the exploration [4], to mitigate the space explosion problem.

Property-driven approaches build on the observation that concurrency faults are more likely to occur under interleavings that expose specific properties, such as suspicious memory access patterns. Property-driven approaches try to overcome the scalability limitations of systematic exploration by guiding the search towards interleavings that expose such properties. Different property-driven approaches target different faults that depend on one or few specific properties. The most widely investigated faults in the literature are *data races* [5], *deadlocks* [6], and *atomicity violations* [7].

The main limitation of property-driven approaches is that the properties they rely on do not necessarily represent the intended behavior of the system [8]. As a consequence, property-driven approaches miss many real faults and typically generate a large number of false positive results, that is, interleavings that expose the target property but that do not manifest a real concurrency fault.

To the best of our knowledge, no existing technique seamlessly integrates the exploration of both the input and interleaving spaces. We believe that a holistic approach that combines the exploration of inputs and interleavings could benefit from the synergy of these two aspects. We claim that a paradigm shift in testing concurrent systems is required to overcome the limitations of current approaches: instead of focusing on detecting properties that might cause concurrency faults, we propose to exploit program specific information and semantic aspects to identify possible effects of concurrency on the expected behavior of the system under test, and use this insight to address the problems of sampling the input and interleaving spaces for test case generation and interleaving selection respectively.

978-1-5090-1827-7/16 $31.00 © 2016 IEEE
DOI 10.1109/ICST.2016.45

II. A New Approach to Test Concurrent Systems

In this PhD work, we plan to define a new approach for testing concurrent systems that integrates the exploration of both the input and the interleaving spaces, and exploits semantic information about the target system to better exercise critical corner cases that can hide concurrency faults otherwise difficult to discover.

Our intuition is that semantic and program specific aspects of the system under test can guide the generation of test cases and the selection of interleavings. Specifically, we rely on the information that the developers encode into assertions, both in the code and in test cases. We prioritize the execution of inputs and interleavings that have a higher probability of changing the values upon which the assertions predicate. This constitutes a paradigm shift from the current approaches for testing concurrent systems, which are guided by patterns that represent potential causes of concurrency faults, to a new approach driven by the possible effects of concurrency on the expected behavior of the system under test. Our goal is to improve the accuracy of the exploration, which results in reducing the amount of false positives and false negatives.

In summary, this PhD work contributes to the research on testing concurrent systems by:

- improving the effectiveness and fault revealing capability of automatically generated test cases for concurrent systems by exploiting semantic information of the target system,
- improving the accuracy and the scalability of the process of selecting interleavings by moving from an approach guided by potentially dangerous patterns of interleavings, to a new approach guided by semantic information to capture possible effects of concurrency on the expected behaviour of the target system,
- reducing the number of false positive and false negative results with respect to the state-of-the-art approaches by seamlessly integrating the processes of test case generation and interleaving selection.

III. Research Plan

We plan to organize the research as follows: we will first validate the benefits of exploiting semantic information extracted from test oracles and program assertions on the selection of interleavings; this includes an empirical assessment on the availability and relevance of assertions for testing concurrent behaviors in real software systems; we will then extend the approach to include the automatic generation of test cases; finally, we will investigate the suitability of additional sources of semantic information of the system under test.

Currently, we are focusing on the problem of selecting interleavings by exploiting test oracles and program assertions. We select the set of interleavings that explore different values of the variables that occur in a program assertion by focusing on the statements that change the values of the *control and data dependencies* [9] of the assertion. These statements may have an impact on the result of the assertions and test oracles of the target program. In particular, their order of execution determines the value of the control and data dependencies of the assertions during the execution, and exploring different order of execution of these statements results in exploring different state values that impact on the assertions and test oracles of the program.

In practice, our approach is articulated in the following activities: (i) dynamically tracing a correct execution of the system under test; (ii) computing the control and data dependencies of a program assertion; (iii) analyzing the execution trace to select different feasible interleavings that can change the value of the control and data dependencies of the assertion; (iv) executing the selected interleavings to check if they lead to a system failure.

We will experimentally validate our approach by comparing it with the state-of-the-art approaches in terms of (i) number of detected faults, (ii) number of selected interleavings, (iii) number of false positives and false negatives, and (iv) types of detected concurrency fault.

We plan to extend the approach to simultaneously consider test inputs and interleavings, by investigating new techniques for test case generation that take advantage from program specific aspects of the system under test.

We will enrich the technique with the usage of other semantic information, such as specifications and models of the system under test. Finally, given the importance of semantic information for our technique, we will investigate the automatic inference of program semantics from execution traces to further reduce the human effort in the overall testing process.

References

[1] G. R. Andrews and F. B. Schneider, "Concepts and notations for concurrent programming," *ACM Computing Surveys*, vol. 15, no. 1, pp. 3–43, 1983.

[2] A. Nistor, Q. Luo, M. Pradel, T. R. Gross, and D. Marinov, "Ballerina: Automatic generation and clustering of efficient random unit tests for multithreaded code," in *Proceedings of the International Conference on Software Engineering*, ser. ICSE '12. IEEE Computer Society, 2012, pp. 727–737.

[3] K. Sen and G. Agha, "Automated systematic testing of open distributed programs," in *Proceedings of the International Conference on Fundamental Approaches to Software Engineering*, ser. FASE '06. Springer, 2006, pp. 339–356.

[4] P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*. Springer, 1996.

[5] D. Marino, M. Musuvathi, and S. Narayanasamy, "Literace: Effective sampling for lightweight data-race detection," in *Proceedings of the Conference on Programming Language Design and Implementation*, ser. PLDI '09. ACM, 2009, pp. 134–143.

[6] M. Eslamimehr and J. Palsberg, "Sherlock: scalable deadlock detection for concurrent programs," in *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '14. ACM, 2014, pp. 353–365.

[7] C. Flanagan, S. N. Freund, and J. Yi, "Velodrome: A sound and complete dynamic atomicity checker for multithreaded programs," in *Proceedings of the Conference on Programming Language Design and Implementation*, ser. PLDI '08. ACM, 2008, pp. 293–303.

[8] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: A comprehensive study on real world concurrency bug characteristics," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '08. ACM, 2008, pp. 329–339.

[9] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems*, vol. 9, no. 3, pp. 319–349, 1987.