Special Issue on Testing, Analysis, and Debugging of Concurrent Programs

This special issue concerns a range of issues related to the development of concurrent programs. This is an important topic, since many systems are now either multi-threaded or distributed, and it is well-known that concurrency makes testing, analysis, and debugging significantly more complicated. Essentially, the alternative interleavings of events can lead to different behaviours and so any analysis, debugging, or testing technique must consider these interleavings. The interest in this topic is reflected in the larger than normal issue, which contains five papers. The papers fall into three groups: we start with a paper on debugging, then have two on static analysis techniques, and finally have two on testing. All papers were reviewed in the normal way.

In the article "UNICORN: A unified approach for localizing non-deadlock concurrency bugs," Park et al. address the topic of concurrent program correctness. The authors devise an automated approach based on a set of 17 memory access patterns. The approach detects non-deadlock concurrency bugs, including order violations and both single-variable and multi-variable atomicity violations. The authors compare UNICORN with CCI and Bugaboo, and conclude that their approach is effective and easy to use.

There are many different techniques to address the detection of race bugs in concurrent programs. The article "A survey of race bug detection techniques for multithreaded programmes," by Hong and Kim, classifies 43 race bug detection techniques and propose a formal execution model that uniformly represent those techniques and enables their comparison.

The article "Object-sensitive cost analysis for concurrent objects," by Albert et al., proposes a novel cost analysis framework to automatically approximate the resource consumption of executing a program in terms of its input parameters. The framework proposal is based on object-sensitive recurrence equations that use cost centres in order to keep the resource usage assigned to the different components separate. This work is evaluated with several classical examples of concurrent and distributed programming.

Concurrent programs have large interleaving state spaces, which are hard to cover in testing. In the article "Advances in noise-based testing of concurrent software," Fiedor et al., address this issue by using noise-based testing techniques. These influence the scheduling so that different interleavings of concurrent actions are witnessed. The authors make a thorough evaluation of the coverage metrics and new noise-injection heuristics for both the seeding (how to generate the noise) and the placement (where to inject noise).

The article "Empirical evaluation of a new composite approach to the coverage criteria and reachability testing of concurrent programs," by Souza et al., addresses the problem of the large number of infeasible synchronization sequences that occur when doing structural testing of concurrent programs. The authors propose a new composite approach that uses reachability testing to

guide the selection of the synchronization sequences according to a specific structural testing criterion and evaluate the effectiveness of their proposal.

Finally, we would like to thank all those who contributed to this special issue. The special issue could not have existed without the hard work of the authors and reviewers. We are also indebted to Wiley's editorial staff and also the STVR joint Editor-in-Chief, Professor Jeff Offutt, for his advice and patience.

Guest Editors

Eitan Farchi, Robert M. Hierons, and Joao Lourenco