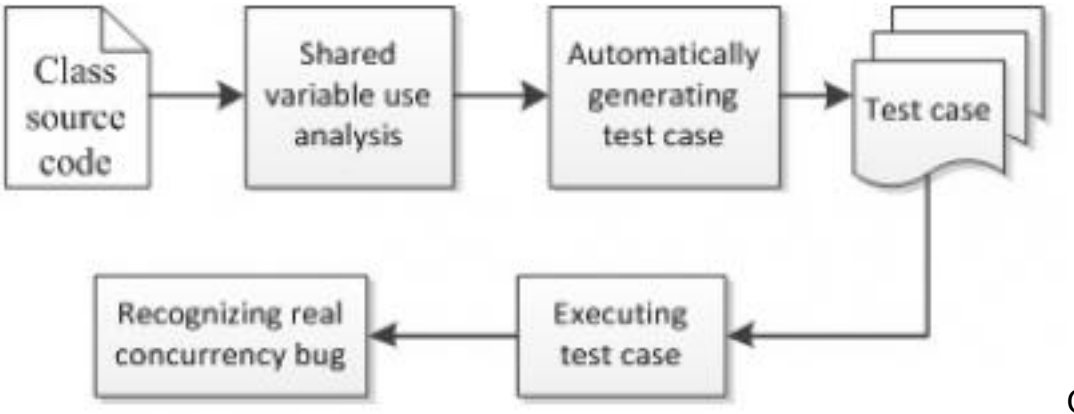
An efficient method for detecting concurrency errors in object-oriented programs

EurekAlert!



Owing to the prevalence of multicore processors, more and more programs are written in a multi-threaded style to improve performance. However, associated concurrency errors have become an inconvenient cause of system faults. The research group from State Key Laboratory of Software Engineering, School of Computers, Wuhan University, focused on finding methods to improve the trustworthiness of concurrent programs. By analyzing shortcomings of existing methods, they developed a more efficient method for detecting concurrency errors in object-oriented programs. Exploiting a static code analysis technique to guide test case generation, they proposed a method to recognize real concurrency errors quickly. Their work, entitled "An efficient method for detecting concurrency errors in object-oriented programs", was published in SCIENCE CHINA Information Sciences 2012, Vol 55(12).

Improving CPU performance solely through manufacturing technology has reached its limits in recent years, owing to the physical limitations of semiconductor-based microelectronics, which have resulted in significant heat dissipation and data synchronization problems. Commercial incentives and technical factors have led to the development of multicore CPUs, which in turn, have had a great impact on software development. Multi-threaded and multi-processor programs on shared memory machines have become increasingly important and prevalent in the past few years. While concurrency can take full advantage of the performance of multicore processors, it makes the programs more likely to be untrusted. A study performed at Microsoft reports that over 60% of developers face concurrency issues and most of the respondents handle concurrency errors on a monthly basis. Not only do concurrency errors reduce the productivity of developers, but they can cause serious problems and disasters, such as the Northeastern blackout and the Therac-25 accident, which were caused by race conditions. So how to simplify the development process and improve the quality of concurrent programs is a highly relevant and important issue urgently in need of solving.

In concurrent programs, threads can interact with each other while they are executing. Thus, the number of possible execution paths in the program can be

extremely large resulting in an indeterminate outcome. Concurrent use of shared resources is a major source of indeterminacy leading to concurrency errors. Concurrency errors are more difficult to detect compared to those in sequential programs because they occur in a specific interleaving of memory-access sequences. The non-deterministic behavior of concurrent programs exacerbates reproducing the specific interleaving. Monitoring and investigating all memory accesses is practically impossible because the number of possible interleavings increases exponentially with the number of threads.

There are several works dealing with detecting concurrency errors. These studies can be divided into two types, namely, static and dynamic testing methods. Model checking, theorem proving and code analysis are three typical static methods. Model checking detects errors by traversing the abstract state space. Although research in space reducing techniques has made great progress, it is still difficult to cope with the state space explosion caused by concurrency. Theorem proving is a sound and complete method to guarantee a program error-free, but it is difficult and time-consuming. Although code analysis is widely used for different purposes, it can only detect shallow concurrency errors. Above all, static methods do not scale well when analyzing concurrent programs owing to the large interleaved space they need to explore.

Dynamic methods detect errors mainly by executing the code. In practice, dynamic testing plays an important role in detecting concurrency errors. There are many dynamic data race detectors, which search for unsynchronized conflicting accesses to shared data by analyzing happens-before relations, checking whether the program follows a locking discipline, or through a combination of these techniques. Most of the existing works can only detect one type of concurrency error, such as a data race, atomicity violation, or deadlock. The method presented in this work is a general approach suitable for detecting many common types of concurrency errors. Integrating static analysis to generate test cases dramatically improves the efficiency and precision. By using an efficient method to decide whether a runtime exception is a real error, this method offers a further advantage in that it does not report false positives.

This work focuses on detecting thread-unsafe classes in object-oriented programs. In general, the proposed method uses three steps to detect concurrency errors in classes using dynamic analysis. First, a generation mechanism has been designed to generate concurrency test cases using a tested class that may run into a state exposing an error. Next, the test case is executed with different code interleavings to expose the error. Finally, the method analyzes the execution result and decides whether the error in execution is a real concurrency error. The key parts and operations in this method for deciding whether a class is thread-safe are illustrated in the following figure.

The innovation of this work relates mainly to two aspects. The researchers propose a heuristic method for automatically generating test cases for concurrency errors. By statically analyzing the class code, the method gains useful information to generate a test case set with high probability of triggering concurrency errors. In addition, the researchers propose an efficient technique for detecting concurrency

errors by analyzing dynamic execution results. They use the concept of "linearization" to distinguish concurrency errors and serial errors.

Implementation and promotion of this work will increase development efficiency of concurrency software, and improve the dependability of concurrent systems. It will contribute greatly towards reducing the complexity of concurrency error detection, thereby reducing manual overhead and economic cost.

The implementation and realization of this work was a collaborative effort involving many researchers and doctoral students. The research project was partially supported by a grant from the National Natural Science Foundation of China. This work greatly advances research in the area of trusted software. The researchers suggest that their work can be improved in many different ways. For example, it can be extended to deal with multi-variable errors and use more powerful heuristic strategies to generate test cases.

Source: http://www.eurekalert.org/pub_releases/2013-02/scp-aem020513.php [1]

Links:
[1] http://www.eurekalert.org/pub_releases/2013-02/scp-aem020513.php