NATIONAL UNIVERSITY OF SINGAPORE

School of Computing

PH.D DEFENCE - PUBLIC SEMINAR

Title: **Specification and Verification of Shared-memory Concurrent Programs**

Speaker: Mr Le Duy Khanh

Date/Time: 26 November 2014, Wednesday, 02:00 PM to 03:30 PM

Venue: Executive Classroom, COM2-04-02

Supervisor : Dr Teo Yong Meng, Associate Professor, School of Computing
Dr Chin Wei Ngan, Associate Professor, School of Computing

Abstract:
The recent adoption of multi-core processors has accelerated the importance of formal verification for shared-memory concurrent programs. Understanding and reasoning about concurrent programs are more challenging than sequential programs because of the notoriously non-deterministic interleavings of concurrent threads. These interleavings may lead to violations of functional correctness, data-race freedom, and synchronization properties such as deadlock freedom. This results in low confidence in the reliability of software systems. Although recent advances in specification and verification have shown promise in increasing the reliability of shared-memory concurrent programs, they mainly focus on partial correctness and data-race freedom, and often ignore the verification of synchronization properties.

In shared-memory concurrent programs, threads, locks, and barriers are among the most commonly-used constructs and the most well-known sources of software bugs. The aim of this thesis is to develop methodologies for advancing verification of shared-memory concurrent programs, in particular to ensure partial correctness, data-race freedom, and synchronization properties of programs with these constructs.

First, we propose ?threads as resource? to enable verification of first-class threads. Threads are first-class in existing programming languages, but current verification approaches do not fully consider threads as first-class. Reasoning about first-class threads is challenging because threads are dynamic and non-lexically-scoped in nature. Our approach considers threads as first-class citizens and allows the ownership of a thread (and its resource) to be flexibly split, combined, and (partially) transferred across procedure and thread boundaries. The approach also allows thread liveness to be precisely tracked. This enables verification of partial correctness and data-race freedom of intricate fork/join behaviors, including the multi-join pattern and threadpool idiom. The notion of ?threads as resource? has recently inspired us to propose ?flow-aware resource predicate? for more expressive verification of various concurrency mechanisms.

Second, threads and locks are widely-used, and their interactions could potentially lead to deadlocks that are not easy to verify. Therefore, we develop a framework for ensuring deadlock freedom of shared-memory programs using fork/join concurrency and non-recursive locks. Our framework advocates the use of precise locksets, introduces delayed lockset checking technique, and integrates with the well-known concept of locklevel to form a unified formalism for verifying deadlock freedom of various scenarios, some of which are not fully studied in the literature. Experimental evaluation shows that, compared to the state-of-the-art deadlock verification system, our approach ensures deadlock freedom of programs with intricate interactions between thread and lock operations.

Lastly, we propose the use of bounded permissions for verifying correct synchronization of static and dynamic barriers in fork/join programs. Barriers are commonly used in practice; hence, verifying correct synchronization of barriers is desirable because it can help improve the precision of compilers and analysers for their analyses and optimizations. However, static verification of barrier synchronization in fork/join programs is a hard problem and has mostly been neglected in the literature. This is because programmers must not only keep track of (possibly dynamic) number of participating threads, but also ensure that all participants proceed in correctly synchronized phases. To the best of our knowledge, ours is the first approach for verifying both static and dynamic barrier synchronization in fork/join programs. The approach has been applied to verify barrier synchronization in the SPLASH-2 benchmark suite.