Verifying Concurrent Programs

(Tutorial Talk)

Aarti Gupta
NEC Laboratories America, Inc.

ABSTRACT

The proliferation of multi-core hardware has led to widespread use of concurrent programs. However, these programs are notoriously difficult to get right and to debug for developers. Even for automated verification, it is a big challenge to reason about subtle synchronization between communicating threads or processes, combined with an exponential number of interleavings. This tutorial will focus on the main ideas that have been used to handle synchronization and interleavings in automatic verification techniques for concurrent programs. These ideas have been applied in many settings, inspired by successful verification efforts for finite state systems on one hand, and for sequential programs on the other.

We will start by describing model checking efforts on concurrent programs based on pushdown system (PDS) models. PDS-based model checking and dataflow analysis have been successfully used for sequential program verification. However, extending these techniques to a system of interacting PDSs is challenging, due to undecidability of basic reachability checking when recursive programs interact using certain kinds of synchronization. Existing methods get around this by using various abstractions or restricting the patterns of synchronization allowed.

While PDSs can naturally model programs, the large model sizes for real programs and the complexity of model checking prohibit their application in practice. Instead, practical model checkers operate over finite state abstractions, typically by inlining procedures (without precisely modeling recursion). Often, they target standard concurrency-related bugs such as dataraces, deadlocks, atomicity violations, etc. Some pioneering efforts used explicit state model checking, with partial order reduction to reduce the number of interleavings. More recently, the success of SAT/SMT solvers has been leveraged for symbolic exploration in bounded settings, where memory consistency axioms implicitly encode the allowable set of interleavings.

On the program analysis front, capturing thread interference plays a key role. A general sequentialization technique has been proposed to lift any sequential program analyzer to bounded context analysis of multi-thread programs. Here, thread interference along a bounded schedule is represented using nondeterministic values on shared data, on which consistency is checked later according to individual threads. For unbounded analysis, techniques have been proposed to utilize automatically generated invariants (via abstract interpretation) to refine an over-approximation of thread interferences, or to incrementally propagate them starting from an under-approximation, until a fixpoint. The ultimate abstraction is to view interference from other threads as the environment, motivating efforts based on thread-modular and compositional verification.

Finally, we will describe trace-methods that utilize given dynamic test executions as a starting point to systematically explore alternate interleavings. Dynamic partial order reduction and preemptive context bounding techniques are based on controlling the scheduler to dynamically explore other interleavings. Another interesting direction is predictive analysis, where a predictive model is derived from a given trace and explored to predict violations in alternate interleavings of the same events.

The tutorial will highlight the progress made along these fronts, and the many challenges that remain in practical settings.