# Software Model Checking

Rupak Majumdar
MPI for Software Systems, Kaiserslautern, Germany

*Software model checking* is the algorithmic analysis of programs to prove properties of their executions. It traces its roots to logic and theorem proving, both to provide the conceptual framework in which to formalize the fundamental questions and to provide algorithmic procedures for the analysis of logical questions.

Initially, the focus of program verification research was on manual reasoning, and the development of axiomatic semantics and logics for reasoning about programs provided a means to treat programs as logical objects. As the size of software systems grew, the burden of providing entire manual proofs became too cumbersome, and algorithmic techniques – at least for the more mundane parts of the proof – were sought.

The search for more automation was influenced by three parallel but somewhat distinct developments. First, development of program logics and associated decision procedures provided a framework and basic algorithmic tools to reason about infinite state spaces. Second, automatic *model checking* techniques provided basic algorithmic tools for state-space exploration. Third, compiler analysis, formalized by *abstract interpretation*, provided formal connections between the logical world of infinite state spaces and the algorithmic world of finite representations. Throughout the 1980s and 1990s, the three communities developed with only occasional interactions. However, in the last decade, there has been a convergence in the research directions and modern software model checkers are a culmination of ideas that combine and perhaps supersede each area alone. In particular, the term ßoftware model checker̈is probably a misnomer, since modern tools simultaneously perform analyses traditionally classified as theorem proving, or model checking, or dataflow analysis. We retain the term solely to reflect historical development.

One major goal of software model checking research today is to expand the scope of automated techniques for program reasoning, both in the scale of programs handled and in the richness of properties that can be checked, reducing the burden on the expert human programmer. We shall see some algorithmic advances over the past few decades that have made this possible, at least in certain domains. In particular, we shall discuss the algorithms implemented in software model checkers such as Slam, Blast, and Yogi, in test generation tools such as DART, Cute, and Klee, and in systematic exploration tools such as Verisoft and Chess. At the same time, we shall see how ideas arising out of model checking have influenced software quality research.

**Lecture schedule** The lecture series has three goals. First, to trace some of the ideas that have combined to produce automatic and precise software model checking tools. Second, to discuss particular application domains to which the tools have been successfully applied. Third, to discuss some interesting but open directions.

A very brief lecture schedule is as follows.

**Lecture 1:** Preliminaries: Symbolic analysis of systems.

**Lecture 2:** Computation of inductive invariants.

**Lecture 3:** Counterexample-guided abstraction refinement.

**Lecture 4:** Recent advances and open problems.

**Background material** The lecture series will roughly follow the survey article [1]. I shall assume basic familiarity with algorithms, logic, and formal language theory, at the level of undergraduate courses (e.g., at the level of [2] and [3]). A background in model checking, while helpful, will not be assumed for the most part.

We shall focus on model checking techniques, but will not be able to cover decision procedures (on which most modern systems are based). I suggest [4] for additional reading. I shall provide pointers to more recent material during the lectures.

## References

[1] R. Jhala, R. Majumdar. *Software Model Checking.* ACM Computing Surveys (CSUR), 41(4), 2009.

[2] T. H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[3] M. Sipser. *Introduction to the Theory of Computation.* PWS Pub. Co, 1996.

[4] A. Bradley, Z. Manna. *The Calculus of Computation.* Springer, 2007.