# Tutorial: Software Model Checking

Edmund Clarke[1] and Daniel Kroening[2,⋆]

[1] Department of Computer Science, Carnegie Mellon University,
Pittsburgh, PA, 15213

[2] Computer Systems Institute, ETH Zürich, Switzerland

**Abstract.** Model Checking is an automated technique for the systematic exploration ofu the state space of a state transition system. The first part of the tutorial provides an introduction to the basic concepts of model checking, including BDD- and SAT-based symbolic model checking, partial order reduction, abstraction, and compositional verification. Model Checking has been applied sucessfully to hardware in the past. However, software has become the most complex part of safety ciritcal systems. The second part of the tutorial covers tools that use Model Checking to formally verify computer software.

## 1 Introduction

Software has become the most complex part of today's safety critical embedded systems. Testing methods can only provide very limited coverage due to the enormous state space that needs to be searched. Formal verification tools, on the other hand, promise full coverage of the state space. Introduced in 1981, *Model Checking* [1, 2] is one of the most commonly used formal verification techniques in a commercial setting. The first part of the tutorial reviews classical explicit state and symbolic model checking algorithms with a focus on software.

The capacity of Model Checking algorithms is constrained by the state-space explosion problem. In case of BDD-based symbolic model checking algorithms, this problem manifests itself in the form of unmanagbly large BDDs. Thus, techniques to reduce the size of the state space, such as the partial order reduction, are discussed.

Abstraction and compositional verification techniques will also be covered briefly.

The second part of the tutorial discusses tools and algorithms for the model checking computer software. We first cover explicit state methods and implementations such as Spin, JPF [3], Bogor [4], and CMC [5]. We describe the area of application of each of

[⋆] This research was sponsored by the Gigascale Systems Research Center (GSRC), the National Science Foundation (NSF) under grant no. CCR-9803774, the Office of Naval Research (ONR), the Naval Research Laboratory (NRL) under contract no. N00014-01-1-0796, and by the Defense Advanced Research Projects Agency, and the Army Research Office (ARO) under contract no. DAAD19-01-1-0485, and the General Motors Collaborative Research Lab at CMU. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of GSRC, NSF, ONR, NRL, DOD, ARO, or the U.S. government.

J. Davies et al. (Eds.): ICFEM 2004, LNCS 3308, pp. 9–10, 2004.

these tools using a concrete example. However, the size of the software system is usually severely constrained when using explicit state model checker.

Software model checking has, in recent years, been applied successfully to large, real software programs, but within certain restricted domains. Many of the tools that have been instrumental in this success have been based on the Counterexample Guided Abstraction Refinement (CEGAR) paradigm [6, 7], first used to model check software programs by Ball and Rajamani [8]. Their SLAM tool [9] has demonstrated the effectiveness of software verification for device drivers. BLAST [10] and MAGIC [11] have been applied to security protocols and real-time operating system kernels.

A common feature of the success of these tools is that the programs and properties examined did not depend on complex data structures. The properties that have been successfully checked or refuted have relied on control flow and relatively simple integer variable relationships. SLAM, BLAST, and MAGIC rely on theorem provers to perform the critical refinement step. The tutorial covers some of the details of these implementations.

## References

1. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.
2. E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop*, volume 131 of *LNCS*. Springer-Verlag, 1981.
3. W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model checking programs. *Automated Software Engineering*, 10(2):203–232, 2003.
4. Robby, E. Rodriguez, M. Dwyer, and J. Hatcliff. Checking strong specifications using an extensible software model checking framework. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 404–420, 2004.
5. M. Musuvathi, D. Park, A. Chou, D. Engler, and D. Dill. CMC: a pragmatic approach to model checking real code. In *Symposium on Operating System Design and Implementation*, 2002.
6. R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata- Theoretic Approach*. Princeton University Press, 1995.
7. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, 2000.
8. T. Ball and S.K. Rajamani. Boolean programs: A model and process for software analysis. Technical Report 2000-14, Microsoft Research, February 2000.
9. T. Ball and S. Rajamani. Automatically validating temporal safety properties of interfaces. In *SPIN Workshop on Model Checking of Software*, pages 103–122, 2001.
10. T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *Principles of Programming Languages*, pages 58–70, 2002.
11. S. Chaki, E. Clarke, A. Groce, S. Jha, and H. Veith. Modular verification of software components in C. *IEEE Transactions on Software Engineering*, 30(6):388–402, June 2004.