Model Checking: Historical Perspective and Example (Extended Abstract) ⋆

Edmund M. Clarke and Sergey Berezin

Carnegie Mellon University — USA

Abstract. Model checking is an automatic verification technique for finite state concurrent systems such as sequential circuit designs and communication protocols. Specifications are expressed in propositional temporal logic. An exhaustive search of the global state transition graph or system model is used to determine if the specification is true or not. If the specification is not satisfied, a counterexample execution trace is generated if possible. By encoding the model using Binary Decision Diagrams (BDDs) it is possible to search extremely large state spaces with as many as 10^{120} reachable states. In this paper we describe the theory underlying this technique and outline its historical development. We demonstrate the power of model checking to find subtle errors by verifying the Space Shuttle *Three-Engines-Out Contingency Guidance Protocol*.

1 Introduction

Logical errors found late in the design phase are an extremely important problem for both circuit designers and programmers. During the past few years, researchers at Carnegie Mellon University have developed an alternative approach to verification called *temporal logic model checking* [10,11]. In this approach specifications are expressed in a propositional temporal logic, and circuit designs and protocols are modeled as state-transition systems. An efficient search procedure is used to determine automatically if the specifications are satisfied by the transition systems.

Model checking has several important advantages over mechanical theorem provers or proof checkers for verification of circuits and protocols. The most important is that the procedure is completely automatic. Typically, the user provides a high level representation of the model and the specification to be checked. The model checking algorithm will either terminate with the answer *true*, indicating that the model satisfies the specification, or give a counterexample execution that shows why the formula is not satisfied. The counterexamples

⋆ This research is sponsored by the the Semiconductor Research Corporation (SRC) under Contract No. 97-DJ-294, the National Science Foundation (NSF) under Grant No. CCR-9505472, and the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-96-C-0071. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of SRC, NSF, DARPA, or the United States Government.

© Springer-Verlag Berlin Heidelberg 1998

are particularly important in finding subtle errors in complex transition systems. The procedure is also quite fast, and usually produces an answer in a matter of minutes or even seconds. Partial specifications can be checked, so it is unnecessary to specify the circuit completely before useful information can be obtained regarding its correctness. Finally, the logic used for specifications can directly express many of the properties that are needed for reasoning about concurrent systems.

The main disadvantage of this technique is the state explosion which can occur if the system being verified has many components that can make transitions in parallel. Recently, however, the size of the transition systems that can be verified by model checking techniques has increased dramatically. The initial breakthrough was made in the fall of 1987 by McMillan, who was then a graduate student at Carnegie Mellon. He realized that using an explicit representation for transition relations severely limited the size of the circuits and protocols that could be verified. He argued that larger systems could be handled if transition relations were represented implicitly with *ordered binary decision diagrams* (OBDDs) [6]. By using the original model checking algorithm with the new representation for transition relations, he was able to verify some examples that had more than 10^{20} states [9,21]. He made this observation independently of the work by Coudert, et. al. [12] and Pixley [23,24,25] on using OBDDs to check equivalence of deterministic finite-state machines. Since then, various refinements of the OBDD-based techniques by other researchers at Carnegie Mellon have pushed the state count up to more than 10^{120} [7].

2 Temporal Logic Model Checking

Pnueli [26] was the first to use temporal logic for reasoning about the concurrent programs. His approach involved proving properties of the program under consideration from a set of axioms that described the behavior of the individual statements in the program. The introduction of temporal logic model checking algorithms in the early 1980's allowed this type of reasoning to be automated. Since checking that a single model satisfies a formula is much easier than proving the validity of a formula for all models, it was possible to implement this technique very efficiently. The first algorithm was developed by Clarke and Emerson in [10]. Their algorithm was polynomial in both the size of the model determined by the program under consideration and in the length of its specification in Computational Tree Logic (CTL). They also showed how *fairness* could be handled without changing the complexity of the algorithm. This was an important step since the correctness of many concurrent programs depends on some type of fairness assumption; for example, absence of starvation in a mutual exclusion algorithm may depend on the assumption that each process makes progress infinitely often.

At roughly the same time Quielle and Sifakis [27] gave a model checking algorithm for a similar branching-time logic, but they did not analyze its complexity or show how to handle an interesting notion of fairness. Later Clarke, Emerson, and Sistla [11] devised an improved algorithm that was linear in the product of the length of the formula and in the size of the global state graph. Sistla and Clarke [28] also analyzed the model checking problem for a variety of other temporal logics and showed, in particular, that for linear temporal logic the problem was PSPACE complete.

A number of papers demonstrated how the temporal logic model checking procedure could be used for verifying network protocols and sequential circuits ([2], [3], [4], [5], [11], [15], [22]). Early model checking systems were able to check state-transition graphs with between 10^4 and 10^5 states at a rate of about 100 states per second. In spite of these limitations, model checking systems were used successfully to find previously unknown errors in several published circuit designs.

Alternative techniques for verifying concurrent systems were proposed by a number of other researchers. The approach developed by Kurshan [17,18] was based on checking *inclusion* between two automata. The first machine represented the system that was being verified; the second represented its specification. Automata on infinite tapes (ω-automata) were used in order to handle fairness. Pnueli and Lichtenstein [20] reanalyzed the complexity of checking linear-time formulas and discovered that although the complexity appears exponential in the length of the formula, it is linear in the size of the global state graph. Based on this observation, they argued that the high complexity of linear-time model checking might still be acceptable for short formulas. Emerson and Lei [16] extended their result to show that formulas of the logic CTL*, which combines both branching-time and linear-time operators, could be checked with essentially the same complexity as formulas of linear temporal logic. Vardi and Wolper [29] showed how the model checking problem could be formulated in terms of automata, thus relating the model checking approach to the work of Kurshan.

3 New Implementations

In the original implementation of the model checking algorithm, transition relations were represented explicitly by adjacency lists. For concurrent systems with small numbers of processes, the number of states was usually fairly small, and the approach was often quite practical. Recent implementations [9,21] use the same basic algorithm; however, transition relations are represented implicitly by *ordered binary decision diagrams* (OBDDs) [6]. OBDDs provide a canonical form for boolean formulas that is often substantially more compact than conjunctive or disjunctive normal form, and very efficient algorithms have been developed for manipulating them. Because this representation captures some of the regularity in the state space determined by circuits and protocols, it is possible to verify systems with an extremely large number of states—many orders of magnitude larger than could be handled by the original algorithm.

The implicit representation is quite natural for modeling sequential circuits and protocols. Each state is encoded by an assignment of boolean values to the set of state variables associated with the circuit or protocol. The transition relation can, therefore, be expressed as a boolean formula in terms of two sets of variables, one set encoding the old state and the other encoding the new. This formula is then represented by a binary decision diagram. The model checking algorithm is based on computing fixed points of *predicate transformers* that are obtained from the transition relation. The fixed points are sets of states that represent various temporal properties of the concurrent system. In the new implementations, both the predicate transformers and the fixed points are represented with OBDDs. Thus, it is possible to avoid explicitly constructing the state graph of the concurrent system.

The model checking system that McMillan developed as part of his Ph.D. thesis is called SMV [21]. It is based on a language for describing hierarchical finite-state concurrent systems. Programs in the language can be annotated by specifications expressed in temporal logic. The model checker extracts a transition system from a program in the SMV language and uses an OBDD-based search algorithm to determine whether the system satisfies its specifications. If the transition system does not satisfy some specification, the verifier will produce an execution trace that shows why the specification is false. The SMV system has been distributed widely, and a large number of examples have now been verified with it. These examples provide convincing evidence that SMV can be used to debug real industrial designs.

4 Related Verification Techniques

A number of other researchers have independently discovered that OBDDs can be used to represent large state-transition systems. Coudert, Berthet, and Madre [12] have developed an algorithm for showing equivalence between two deterministic finite-state automata by performing a breadth first search of the state space of the product automata. They use OBDDs to represent the transition functions of the two automata in their algorithm. Similar algorithms have been developed by Pixley [23,24,25]. In addition, several groups including Bose and Fisher [1], Pixley [23], and Coudert, et. al. [13] have experimented with model checking algorithms that use OBDDs. Although the results of McMillan's experiments [8,9] were not published until the summer of 1990, his work is referenced by Bose and Fisher in their 1989 paper [1].

5 Example: Space Shuttle Digital Autopilot

We illustrate the power of model checking to find subtle errors by considering a protocol used by the Space Shuttle. We discuss the verification of the *Three-Engines-Out Contingency Guidance Requirements* using the SMV model checker. The example describes what should be done in a situation where all of the three main engines of the Space Shuttle fail during the ascent. The main task of the *Space Shuttle Digital Autopilot* is to separate the shuttle from the external tank. This task has many different input parameters, and it is important to make sure that all possible cases and input values are taken into account.

The Digital Autopilot chooses one of the six contingency regions depending on the current flight conditions. Each region uses different maneuvers for separating from the external tank. This involves computing a guidance *quaternion.* Usually, the region is chosen once at the beginning of the contingency and is maintained until separation occurs. However, under certain conditions a change of region is allowed. In this case, it is necessary to recompute the quaternion and certain other output values. Using SMV we were able to find a counterexample in the program for this task. We discovered that when a transition between regions occurs, the autopilot system may fail to recompute the quaternion and cause the wrong maneuver to be made. The guidance program consists of about 1200 lines of SMV code. The number of reachable states is $2 \cdot 10^{14}$, and it takes 60 seconds to verify 40 CTL formulas.

Specifically, the error occurs when a change is made from region 2 to region 1. Region 2 is selected initially if the Shuttle is descending and the dynamic pressure is not safe for attitude independent separation. In this region it is necessary to consider the position of the craft relative to its velocity vector, and the quaternion computed in this region is supposed to minimize the *angle of attack* and the *side slip.* However, if the side slip is too big and the dynamic pressure builds up too quickly, meaning that we do not have enough time to perform the maneuver, then the program performs the transition to region 1 — an attitude independent emergency separation.

In this mode, in contrast to region 2, the current values of the angle of attack and the side slip must be frozen, and the tank will separate as soon as the *angle rates* become relatively small. A special flag called `Freeze_flag` is set to indicate this maneuver. However, the quaternion from region 2 is not recomputed and causes the space shuttle to rotate. This violates the condition that the angle of attack and sideslip should be frozen. Since the part of the specifications we possessed does not indicate whether the `Freeze_flag` has a precedence over the quaternion or not, this situation may lead to an incorrect behavior of the Space Shuttle in a critical situation.

The same example was also verified by Judith Crow at SRI [14] using an explicit state model checker called Murϕ. She had to abstract away many variables to avoid the state explosion problem, and her model was not as complete as ours. She found a similar error in the transition from region 2 to region 1, but for a different variable, which turned out to be correct in our model. Instead, the error shows up in the quaternion, which she didn't consider.

References

1. S. Bose and A. L. Fisher. Automatic verification of synchronous circuits using symbolic logic simulation and temporal logic. In L. Claesen, editor, *Proceedings of the IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, November 1989.
2. M. C. Browne and E. M. Clarke. Sml: A high level language for the design and verification of finite state machines. In *IFIP WG 10.2 International Working Conference from HDL Descriptions to Guaranteed Correct Circuit Designs, Grenoble, France.* IFIP, September 1986.

3. M. C. Browne, E. M. Clarke, and D. Dill. Checking the correctness of sequential circuits. In *Proceedings of the 1985 International Conference on Computer Design*, Port Chester, New York, October 1985. IEEE.
4. M. C. Browne, E. M. Clarke, and D. Dill. Automatic circuit verification using temporal logic: Two new examples. In *Formal Aspects of VLSI Design.* Elsevier Science Publishers (North Holland), 1986.
5. M. C. Browne, E. M. Clarke, D. L. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, C-35(12):1035–1044, December 1986.
6. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
7. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In A. Halaas and P. B. Denyer, editors, *Proceedings of the 1991 International Conference on Very Large Scale Integration*, August 1991. Winner of the Sidney Michaelson Best Paper Award.
8. J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential circuit verification using symbolic model checking. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 46–51. IEEE Computer Society Press, June 1990.
9. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.
10. E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In D. Kozen, editor, *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*, volume 131 of *Lecture Notes in Computer Science.* Springer-Verlag, 1981.
11. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.
12. O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *Proceedings of the 1989 International Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France*, volume 407 of *Lecture Notes in Computer Science.* Springer-Verlag, June 1989.
13. O. Coudert, J. C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In Kurshan and Clarke [19].
14. Judith Crow. Finite-state analysis of space shuttle contingency guidance requirements. Technical Report NASA Contractor Report 4741, SRI International, Menlo Park, CA, May 1996.
15. D. L. Dill and E. M. Clarke. Automatic verification of asynchronous circuits using temporal logic. *IEE Proceedings*, Part E 133(5), 1986.
16. E.A. Emerson and Chin Laung Lei. Modalities for model checking: Branching time strikes back. *Twelfth Symposium on Principles of Programming Languages, New Orleans, La.*, January 1985.
17. Z. Har'El and R. P. Kurshan. Software for analytical development of communications protocols. *AT&T Technical Journal*, 69(1):45–59, Jan.–Feb. 1990.
18. R. P. Kurshan. Analysis of discrete event coordination. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness*,

volume 430 of *Lecture Notes in Computer Science*, pages 414–453. Springer-Verlag, May 1989.

19. R. P. Kurshan and E. M. Clarke, editors. *Proceedings of the 1990 Workshop on Computer-Aided Verification*. Springer-Verlag, June 1990.
20. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 97–107. Association for Computing Machinery, January 1985.
21. K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, 1992.
22. B. Mishra and E.M. Clarke. Hierarchical verification of asynchronous circuits using temporal logic. *Theoretical Computer Science*, 38:269–291, 1985.
23. C. Pixley. A computational theory and implementation of sequential hardware equivalence. In R. Kurshan and E. Clarke, editors, *Proc. CAV Workshop (also DIMACS Tech. Report 90-31)*, Rutgers University, NJ, June 1990.
24. C. Pixley, G. Beihl, and E. Pacas-Skewes. Automatic derivation of FSM specification to implementation encoding. In *Proceedings of the International Conference on Computer Desgin*, pages 245–249, Cambridge, MA, October 1991.
25. C. Pixley, S.-W. Jeong, and G. D. Hachtel. Exact calculation of synchronization sequences based on binary decision diagrams. In *Proceedings of the 29th Design Automation Conference*, pages 620–623, June 1992.
26. A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
27. J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the Fifth International Symposium in Programming*, 1982.
28. A. P. Sistla and E.M. Clarke. Complexity of propositional temporal logics. *Journal of the ACM*, 32(3):733–749, July 1986.
29. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, June 1986.