Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Model Checking of Complex Systems

Report

Author
Vince Molnár

Supervisor
Dr. István Majzik,

June 4, 2015

1 Introduction

In the early '80s, *model checking*, a promising new technique of formal verification was developed by E. M. Clarke and E. A. Emerson [10] and independently by J. P. Quielle and J. Sifakis [19]. They used *temporal logics* to specify the formal requirements and enumerated the possible behaviors of the system, checking them against the specification. Since then, there has been an enormous leap in the field of model checking. Various kinds of algorithms have been implemented for different temporal logics and model formalisms, and a lot of optimizations and special methods were devised to broaden the range of systems that can be verified this way.

A great challenge of model checking even to this day is the so-called *state space explosion problem*. Especially when a system has independent components or a large number of variables, the number of states of the system can grow extremely large. To address this problem, *symbolic model checking* was introduced in the early '90s [5]. In symbolic model checking, states and state transitions are not simply enumerated, but represented symbolically by functions, encoded in efficient data structures such as *decision diagrams*. Although this encoding can usually represent state spaces larger by several orders of magnitudes, the traditional *explicit state model checkers* can sometimes outperform symbolic approaches using the many devised optimizations that could not be implemented in symbolic model checking so far. The reason behind this mostly lies in graph traversal algorithms, because symbolic methods usually have to use breadth-first search instead of depth-first search.

In the early 2000s, G. Ciardo et al. developed a new iteration strategy specially tailored to decision diagrams [6]. They called it the *saturation* iteration strategy, in which the search was driven by the structure of the decision diagram, processing nodes in it instead of the states it represented. Saturation proved to be very successful in verifying concurrent, asynchronous systems. Such systems usually focus on the interaction of individual components, and they are especially prone to the state space explosion problem.

In terms of temporal logic specifications, two formalisms became widespread. *Linear temporal logic* (LTL) was the first formalism introduced in formal specification due to its intuitive interpretation. *Computational tree logic* (CTL) became popular with the first model checker algorithms, because the worst-case complexity of CTL model checking is linear in the size of the state space (cf. state space explosion) and the specification as well. LTL model checkers appeared later, with exponential complexity in the size of the specification [20]. However, the semantics of LTL provided an opportunity to design the algorithms to work in an on-the-fly manner, terminating immediately when a proof is found. CTL model checkers rarely possess such a feature, because many of them explore the state space first, then compute fixed points according to the subexpressions of the specification formula.

On the other hand, the latter approach made CTL model checking suitable for a symbolic implementation, since fixed point computations on decision diagrams were easy to perform.

Symbolic LTL model checkers also appeared, but the on-the-fly manner of the explicit algorithms turned out to be cumbersome to implement in a symbolic way, because it relied on depth-first search. This difficulty makes symbolic LTL model checking an interesting challenge, one that has been therefore tackled by many researchers in different ways.

This report aims to give the reader a deeper insight into the current state of model checking approaches, especially techniques related to symbolic LTL model checking.

2 Techniques of Model checking

This section summarizes results in the domain of model checking, especially those related to LTL model checking, SCC computation, abstraction and saturation. Section 2.1 will present some traditional explicit techniques and algorithms, including SCC detection algorithms and partial order reduction. Section 2.2 introduces methods to reduce model checking to the Boolean satisfiability problem (SAT). Section 2.3 will show how traditional BDD-based symbolic approaches can be used to perform LTL model checking. Section 2.4 points the reader to some abstraction techniques that proved to be successful in hybrid approaches. Finally, Section 2.5 will give some examples of using saturation in model checking.

2.1 Explicit Techniques in Model Checking

Model checking (both CTL and LTL) can be fully realized by graph algorithms – these solutions are called explicit model checkers. Implicit representation of transitions given by the high-level source model can be used to generate the states of the system one by one. State space generation can then use breadth-first search or depth-first search or some of their variants to explore the full state graph.

On-the-fly exploration of the product automaton in LTL model checking is also easy to implement, because transitions can be computed one by one as well. On-the-fly SCC detection, however, requires some specialized algorithms.

Explicit on-the-fly SCC computation

One such algorithm is *Tarjan's algorithm* [23], which works by exploiting the exploration strategy of DFS. The algorithm uses an additional stack. Every time a node is explored, it is put on the stack. Each node is assigned a number *index* in the order they are traversed, and also another one *lowlink* (initially equals *index*) that keeps track of the node with the lowest *index* reachable from the current node. This number is recursively calculated by the time DFS backtracks to the node. If *lowlink* is smaller than *index*, the node is in a strongly connected component and is left on the stack. If they are equal, the node is the "root" of the SCC and every state left on the stack above it will be part of the same SCC

(they are all popped). Tarjan's algorithm aims to compute every SCC of a directed graph. Accepting SCCs can then be filtered by checking if the SCCs contain an accepting state or not.

Another similar algorithm is *nested depth-first search* [15]. While Tarjan's algorithm computes every SCC, nested DFS will look for a single accepting loop in the graph (i. e., a counterexample in LTL model checking). It employs two depth-first searches to do this. The first DFS explores the graph, and every time it backtracks to an accepting state, the second DFS will try to find a way back to a state in the stack of the first DFS. If it manages to do so, the two stacks together give a counterexample.

Partial order reduction

Partial order reduction (POR) is a technique used to exploit the symmetries of a state space usually caused by interleaving semantics in concurrent systems [24, 18, 12]. When several processes can advance independently until a point of synchronization, the order in which parallel local steps are interleaved may be irrelevant. These steps are only partially ordered, but with interleaving semantics, every possible total orders will be explored by the model checking algorithm. In such cases, it may be sufficient to choose a single total order and aggregate the independent steps, greatly reducing the size of the state space.

There have been some attempts to combine partial order reduction and symbolic model checking [1], but it is still an open problem that is not trivial to solve.

2.2 SAT-based Model Checking

SAT-based bounded model checking encodes an instance of the bounded model checking problem as a SAT instance [2]. A path of length k is described in term of the Kripke structure and the property, and SAT-solvers are used to find an assignment representing a violating path.

Bounded model checking works by iteratively incrementing the depth value k, and is only complete if k reaches the diameter of the state space. Nevertheless, due to the boundedness of each iteration, this approach can be applied to infinite models as a semi-decision procedure.

For safety properties, k-induction – a generalized form of mathematical induction – is able to prove unreachability without exploring the full state space by providing inductive proofs [21].

A major breakthrough in this area was the application of interpolants to over-approximate the state space [17]. This allowed the algorithm to prove unsatisfiability much earlier than traditional approaches.

A recent approach called IC3 [3] works by constructing a series of small intermediate and inductive lemmas. Its novelty and power lies in focusing on small proofs instead of constructing large monolithic interpolants. By dividing the proof into smaller subgoals, SAT-solvers have to process smaller queries. As it turned out, proving more smaller lemmas can be much easier than solving less, but large and monolithic ones.

IC3 was also applied to look for SCCs in the state space, making the approach able to check liveness properties [4]. The approach identifies one-way walls in the state space to divide it into SCC-closed sets (parts of the state space that can fully contain an SCC) or prove the absence of SCCs.

2.3 Decision Diagram-based LTL Model Checking

Decision diagram-based model checking implements symbolic model checking by encoding states and transitions in decision diagrams. Symbolic steps are then reduced to set operations, which can be performed directly on the digram representation providing an efficient way to handle large state spaces.

Different properties of the state space can be characterized using fixed points [11]. Symbolic model checking works by computing these fixed points. In case of CTL and LTL model checking, least and greatest fixed points are used to characterize the set of reachable states and strongly connected components, respectively.

Fixed point algorithms on decision diagrams

The set of reachable states is characterized by the least fixed point of $\mathcal{S}_{rch} = \mathcal{N}(\mathcal{S}_{rch})$ including the set of initial states ($\mathcal{I} \subseteq \mathcal{S}_{rch}$), where $\mathcal{N}$ is the *next-state function*.

SCCs are characterized by greatest fixed points of some functions where the fixed point is part of the reachable states or transitions. The selected function greatly affects the performance of the algorithm and the exact meaning of the fixed point.

Instead of computing SCCs, it is common to compute *SCC-hulls*, parts of the state space that are sure to contain at least one SCC. A family of algorithms implementing this approach are instances of the Generalized SCC-hull (GSH) algorithm [22]. The most famous one is the algorithm of Emerson and Lei, which have optimal complexity.

As an illustration, a simple function $f : \mathcal{S} \to \mathcal{S}$ that characterizes an SCC-hull is one that removes "dead-end" states:

$$f(S) = \{\mathbf{s} \mid \exists \mathbf{s}', (\mathbf{s}, \mathbf{s}') \in \mathcal{N}\}$$

The greatest fixed point of f contained in the set of reachable states is a reachable SCC. In LTL, such an SCC containing at least one accepting state indicates the presence of a counterexample.

2.4 Abstraction in LTL Model Checking

Approximation of a state space by abstraction is a widely used technique to prove properties without exploring the full state space (as also seen in Section 2.2). In LTL model checking, SCC detection can sometimes be realized on (small) abstractions. With this technique, it is possible to implement on-the-fly symbolic LTL model checking and efficient SCC detection.

Abstraction based on the automaton

A common way of abstracting the state space based on the automaton describing the desired property. Such techniques include symbolic observation graph (SOG) [13, 16], its extension, symbolic observation product (SOP) [8] and self-loop aggregation product (SLAP) [9].

Symbolic observation graphs are aggregated Kripke structures: each state of the SOG is a set of states of the original model. Consecutive states are grouped by observable atomic propositions. An improvement of symbolic observation product is based on the observation that that as the automaton of the property progresses, the set of relevant atomic propositions decreases, allowing a more aggressive grouping. Furthermore, SOP can substitute the product, because it is in itself a tableau representing the system in terms of the property. This approach, however, assumes a globally stuttering property (i. e., operator X cannot be used).

Self-loop aggregation product works with every LTL formula. SLAP aggregates consecutive states by observing self loops of the Büchi automaton: states are aggregated if their labels satisfy the condition of self-loops, i. e., the automaton does not change its state when the system does.

Abstraction based on components

An extensive approach to using abstraction in SCC computation has been proposed in [25]. By defining a lattice of abstractions based on one or more components of the model, the paper presents strategies of using some of the abstractions to discard uninteresting parts of the state space and search in relevant components.

Each abstraction is obtained by projecting the state space to some of the components, keeping only those transitions that does not affect other components. Searching for SCCs in the abstract graphs can prove that no SCCs exist within the selected components. If an SCC is found, the algorithm also looks for accepting states, which offers another way of discarding irrelevant components. Completeness is achieved by refining the abstraction until an SCC is found or the full state space is searched.

2.5 Saturation in Model Checking

Saturation was first applied in CTL model checking in [7]. It was used to compute labels of states according to the CTL operators. While this approach worked, the algorithm became really efficient with the introduction of constrained saturation [26].

LTL model checking based on saturation was introduced in [14].

Fixed point algorithms using saturation

Saturation-based SCC computation has also been proposed by [27]. The two implemented algorithms are that of Xie and Beerel (XB) and the Transitive Closure (TC) algorithm. Both of them differ from SCC-hull algorithms, because they aim to compute exactly those states that belong to an SCC.

The main idea of the XC algorithm is to compute the set of forward and backward reachable states from a randomly picked seed state – their intersection gives an SCC. After removing the states of this SCC, the procedure is repeated until no states are left.

The TC method works by building the relation of reachability between reachable states. This relation can then be used to identify SCCs in a fully symbolic way.

While the XB algorithm is usually much faster, it does not scale well with the number of SCCs in the state space. TC does not enumerate the strongly connected components, it encodes them symbolically instead.

3 Summary

Model checking of complex, concurrent and asynchronous systems is a computationally difficult problem. Various techniques and algorithms were developed in the history to tackle the state space explosion problem which is inherent in these systems, and to combat the complexity of LTL model checking. This is also the context of this work: in this report, different approaches of model checking have been explored.

It is easy to see that many of the presented methods are orthogonal, i. e., they can be combined to possibly yield even more efficient solutions. However, many algorithms rely on special data structures and have assumptions about the models and other algorithms that make this integration harder. While many approaches are already customized to work together in a single tool, these variants are even more specialized, further increasing the diversity of available approaches.

All in all, harmonization of different approaches would be desirable and closer cooperation of tool developers could carry a lot of potential in the field of model checking.

References

[1] Rajeev Alur, Robert K. Brayton, Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. Partial-order reduction in symbolic state-space exploration. *Formal Methods in System Design*, 18(2):97–116, 2001.

[2] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, 1999. Springer-Verlag.

[3] Aaron Bradley. Understanding IC3. In *Theory and Applications of Satisfiability Testing – SAT 2012*, number 7317 in Lecture Notes in Computer Science, pages 1–14. Springer, 2012.

[4] Aaron R. Bradley, Fabio Somenzi, Zyad Hassan, and Yan Zhang. An incremental approach to model checking progress properties. In *Proc. of the International Conference on Formal Methods in Computer-Aided Design*, pages 144–153, Austin, Texas, 2011. FMCAD Inc.

[5] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[6] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. Saturation: An efficient iteration strategy for symbolic state space generation. In Tiziana Margaria and Wang Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, number 2031 in Lecture Notes in Computer Science, pages 328–342. Springer Berlin Heidelberg, 2001.

[7] Gianfranco Ciardo and Radu Siminiceanu. Structural symbolic CTL model checking of asynchronous systems. In *Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 40–53. Springer-Verlag, 2003.

[8] Alexandre Duret-Lutz, Kaïs Klai, Denis Poitrenaud, and Yann Thierry-Mieg. Combining explicit and symbolic approaches for better on-the-fly LTL model checking. *arXiv:1106.5700 [cs]*, 2011. arXiv: 1106.5700.

[9] Alexandre Duret-Lutz, Kaïs Klai, Denis Poitrenaud, and Yann Thierry-Mieg. Self-loop aggregation product – a new hybrid approach to on-the-fly LTL model checking. In *Automated Technology for Verification and Analysis*, volume 6996 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2011.

[10] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, number 85 in Lecture Notes in Computer Science, pages 169–181. Springer Berlin Heidelberg, 1980.

[11] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 169–181, London, UK, 1980. Springer-Verlag.

[12] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*. Springer-Verlag, Secaucus, NJ, USA, 1996.

[13] Serge Haddad, Jean-Michel Ilié, and Kaïs Klai. Design and evaluation of a symbolic and abstraction-based model checker. In Farn Wang, editor, *Automated Technology for Verification and Analysis*, number 3299 in Lecture Notes in Computer Science, pages 196–210. Springer Berlin Heidelberg, 2004.

[14] Alexandre Hamez, Yann Thierry-Mieg, and Fabrice Kordon. Hierarchical set decision diagrams and automatic saturation. In Kees M. van Hee and Rüdiger Valk, editors, *Applications and Theory of Petri Nets*, number 5062 in Lecture Notes in Computer Science, pages 211–230. Springer Berlin Heidelberg, 2008.

[15] Gerard J. Holzmann, Doron Peled, and Mihalis Yannakakis. On nested depth first search. In *Proceedings of the Second SPIN Workshop*, DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science, pages 81–89. AMS, 1997.

[16] Kaïs Klai and Denis Poitrenaud. MC–SOG: An LTL model checker based on symbolic observation graphs. In *Applications and Theory of Petri Nets*, volume 5062 of *Lecture Notes in Computer Science*, pages 288–306. Springer, 2008.

[17] Kenneth L. McMillan. Interpolation and SAT-based model checking. In *Lecture Notes in Computer Science*, volume 2725, pages 1–13, 2003.

[18] Doron Peled. All from one, one for all: On model checking using representatives. In *Proceedings of the 5th International Conference on Computer Aided Verification*, CAV '93, pages 409–423, London, UK, 1993. Springer-Verlag.

[19] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *International Symposium on Programming*, number 137 in Lecture Notes in Computer Science, pages 337–351. Springer Berlin Heidelberg, 1982.

[20] Philippe Schnoebelen. The complexity of temporal logic model checking. In *Advances in Modal Logic*, volume 4, pages 1–44. King's College Publications, 2003.

[21] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a SAT-solver. In *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, FMCAD '00, pages 108–125, London, UK, 2000. Springer-Verlag.

[22] Fabio Somenzi, Kavita Ravi, and Roderick Bloem. Analysis of symbolic SCC hull algorithms. In Mark D. Aagaard and John W. O'Leary, editors, *Formal Methods in Computer-Aided Design*, number 2517 in Lecture Notes in Computer Science, pages 88–105. Springer Berlin Heidelberg, 2002.

[23] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[24] Antti Valmari. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets: Advances in Petri Nets 1990*, pages 491–515, London, UK, 1991. Springer-Verlag.

[25] Chao Wang, Roderick Bloem, Gary D. Hachtel, Kavita Ravi, and Fabio Somenzi. Compositional SCC analysis for language emptiness. *Formal Methods in System Design*, 28(1):5–36, 2006.

[26] Yang Zhao and Gianfranco Ciardo. Symbolic CTL model checking of asynchronous systems using constrained saturation. In Zhiming Liu and Anders P. Ravn, editors, *Automated Technology for Verification and Analysis*, number 5799 in Lecture Notes in Computer Science, pages 368–381. Springer Berlin Heidelberg, 2009.

[27] Yang Zhao and Gianfranco Ciardo. Symbolic computation of strongly connected components and fair cycles using saturation. *Innovations in Systems and Software Engineering*, 7(2):141–150, 2011.