Bounded Model Checking and CBMC

Armin Biere, Alessandro Cimatti, Edmund Clarke
Daniel Kröning, Flavio Lerda, Yunshan Zhu

CAV Award 2018

Oxford, UK
June 15th, 2018
Bounded Model Checking

- Look only for counter example made of $k$ states

  \[ I(s_0) \land T(s_0, s_1) \land \cdots \land T(s_{k-1}, s_k) \land \bigvee_{i=0}^{k} \neg p(s_i) \]

  “$k$” = bound

- Simple for safety properties $p$ invariantly true

- Harder for liveness properties $p$ eventually true

- Compute and bound $k$ by diameter
How did Bounded Model Checking happen?

- 1997: interest and capacity of BDDs stalled but there were success stories of “other” techniques
  CAV’97 in Haifa had an invited talk by Arne Borälv on
  “The Industrial Success of Verification Tools Based on Stålmarck’s Method”

- Edmund Clarke hired Yunshan Zhu and Armin Biere as Post-Docs with the job-description
  *Use SAT for Symbolic Model Checking!* (YZ expert on Theorem Proving, AB on BDDs)

- struggled for 10 months to come up with something that could replace / improve on BDDs
  mainly looked at QBF back then (point was that we need to handle quantifiers to do image computation)

- Alessandro Cimatti came to an AI conference in Pittsburg and at lunch (at an Indian Restaurant)
  we realized, that for planning they **do not care about completeness**

  *What if we apply this to model checking?  How to handle temporal logic?*

- after one afternoon for the theory and 3 months of implementation, benchmarking, writing it up …
Symbolic Model Checking without BDDs*

Armin Biere¹, Alessandro Cimatti², Edmund Clarke¹, and Yunshan Zhu¹

¹ Computer Science Department, Carnegie Mellon University
   5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A
   {Armin.Biere,Edmund.Clarke,Yunshan.Zhug}@cs.cmu.edu
² Istituto per la Ricerca Scientifica e Tecnologica (IRST)
   via Sommarive 18, 38055 Povo (TN), Italy
   cimatti@irst.itc.it

Abstract. Symbolic Model Checking [3, 14] has proven to be a powerful technique for the verification of reactive systems. BDDs [2] have traditionally been used as a symbolic representation of the system. In this paper we show how boolean decision procedures, like Stålmarck’s Method [16] or the Davis & Putnam Procedure [7], can replace BDDs. This new technique avoids the space blow up of BDDs, generates counterexamples much faster, and sometimes speeds up the verification. In addition, it produces counterexamples of minimal length. We introduce a bounded model checking procedure for LTL which reduces model checking to propositional satisfiability. We show that bounded LTL model checking can be done without a tableau construction. We have implemented a model checker BMC, based on bounded model checking, and preliminary results are presented.
Symbolic model checking without BDDs

Authors: Armin Biere, Alessandro Cimatti, Edmund Clarke, Yunshan Zhu

Publication date: 1999/3/22

Conference: International conference on tools and algorithms for the construction and analysis of systems

Pages: 193-207

Publisher: Springer, Berlin, Heidelberg

Description: Abstract Symbolic Model Checking [3],[14] has proven to be a powerful technique for the verification of reactive systems. BDDs [2] have traditionally been used as a symbolic representation of the system. In this paper we show how boolean decision procedures, like Stålmarck's Method [16] or the Davis & Putnam Procedure [7], can replace BDDs. This new technique avoids the space blow up of BDDs, generates counterexamples much faster, and sometimes speeds up the verification. In addition, it produces counterexamples of minimal length. We introduce a bounded model checking procedure for LTL which reduces model checking to propositional satisfiability. We show that bounded LTL model checking can be done without a tableau construction. We have implemented a model checker BMC, based on bounded model checking, and preliminary results are presented.

Total citations: Cited by 2495

Source: Google Scholar, 28th June 2018
Symbolic model checking using SAT procedures instead of BDDs

Authors: Armin Biere, Alessandro Cimatti, Edmund M Clarke, Masahiro Fujita, Yunshan Zhu
Publication date: 1999/6/1
Pages: 317-320
Publisher: ACM

Description: In this paper, we study the application of propositional decision procedures in hardware verification. In particular, we apply bounded model checking, as introduced in [1], to equivalence and invariant checking. We present several optimizations that reduce the size of generated propositional formulas. In many instances, our SAT-based approach can significantly outperform BDD-based approaches. We observe that SAT-based techniques are particularly efficient in detecting errors in both combinational and sequential designs.

Total citations: Cited by 863

Source: Google Scholar, 28th June 2018
Verifying Safety Properties of a PowerPC- Microprocessor Using Symbolic Model Checking without BDDs

Authors: Armin Biere, Edmund Clarke, Richard Raimi, Yunshan Zhu
Publication date: 1999/7/6
Conference: International Conference on Computer Aided Verification
Pages: 60-71
Publisher: Springer, Berlin, Heidelberg

Abstract: In [1] Bounded Model Checking with the aid of satisfiability solving (SAT) was introduced as an alternative to symbolic model checking with BDDs. In this paper we show how bounded model checking can take advantage of specialized optimizations. We present a bounded version of the cone of influence reduction. We have successfully applied this idea in checking safety properties of a PowerPC microprocessor at Motorola’s Somerset PowerPC design center. Based on that experience, we propose a verification methodology that we feel can bring model checking into the mainstream of industrial chip design.

Total citations: Cited by 180

Source: Google Scholar, 28th June 2018
Bounded model checking.

Authors: Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu

Publication date: 2003/12

Journal: Advances in computers

Volume: 58

Issue: 11

Pages: 117-148

Description: Symbolic model checking with Binary Decision Diagrams (BDDs) has been successfully used in the last decade for formally verifying finite state systems such as sequential circuits and protocols. Since its introduction in the beginning of the 90's, it has been integrated in the quality assurance process of several major hardware companies. The main bottleneck of this method is that BDDs may grow exponentially, and hence the amount of available memory restricts the size of circuits that can be verified efficiently. In this article we survey a technique called Bounded Model Checking (BMC), which uses a propositional SAT solver rather than BDD manipulation techniques. Since its introduction in 1999, BMC has been well received by the industry. It can find many logical errors in complex systems that can not be handled by competing techniques, and is therefore widely perceived as a complementary ...

Total citations: Cited by 857

Source: Google Scholar, 28th June 2018
Bounded model checking using satisfiability solving

Authors: Edmund Clarke, Armin Biere, Richard Raimi, Yunshan Zhu
Publication date: 2001/7/1
Journal: Formal methods in system design
Volume: 19
Issue: 1
Pages: 7-34
Publisher: Kluwer Academic Publishers

Description: The phrase model checking refers to algorithms for exploring the state space of a transition system to determine if it obeys a specification of its intended behavior. These algorithms can perform exhaustive verification in a highly automatic manner, and, thus, have attracted much interest in industry. Model checking programs are now being commercially marketed. However, model checking has been held back by the state explosion problem, which is the problem that the number of states in a system grows exponentially in the number of system components. Much research has been devoted to ameliorating this problem. In this tutorial, we first give a brief overview of the history of model checking to date, and then focus on recent techniques that combine model checking with satisfiability solving. These techniques, known as bounded model checking, do a very fast exploration of the state space, and for ...

Total citations: Cited by 700

Source: Google Scholar, 28th June 2018
Replacing Testing with Formal Verification in Intel® Core™ i7 Processor Execution Engine Validation

Roope Karvola, Rajnimish Ghughi, Niren Narasimhan, Amber Télör, Jesse Whittemore, Shiadhinda Pandav, Anna Stobodow, Christopher Taylor, Vladimir Frolov, Erik Reberer, and Armanagh Naik

Intel Corporation, JF4-451, 2111 NE 25th Avenue, Hillsboro, OR 97124, USA

Abstract. Formal verification of arithmetic datapaths has been part of the established methodology for most Intel processor designs over the last years, usually in the role of supplementing more traditional coverage oriented testing activities. For the recent Intel® Core™ i7 design we took a step further and used formal verification to the primary validation vehicle for this core execution cluster, the component responsible for the functional behaviour of all microinstructions. We applied symbolic simulation based formal verification techniques for full datapath, control and state validation for the cluster, and dropped coverage driven testing entirely. The project, involving some twenty person years of verification work, is one of the most ambitious formal verification efforts in the hardware industry to date. Our experiences show that under the right circumstances, full formal verification of a design component is a feasible, substantially viable and competitive validation approach.

1 Introduction

Most Intel processors launched over the last ten years have contained formally verified components. This is hardly surprising, as their reliability is crucial, and the cost of correcting problems can be very high. Formal verification has been applied to a range of design components or features: low-level protocols, register renaming, arithmetic units, microarchitecture feature descriptions etc. [19, 4]. In an industrial product development setting, formal verification is a tool, one among others, and it competes with traditional testing and simulation. Usually testing can produce initial results much faster than formal verification, and in our view the value of formal verification primarily comes from its ability to cover every possible behaviour. In most of the cases where formal verification has been applied, its role has been that of a supplementary verification method on top of a full-fledged simulation based dynamic validation effort.

The single most sustained formal verification effort has been made in the area of arithmetic, in particular floating point datapaths. In this area verification methods have reached sufficient maturity that they have now been routinely applied for a series of design projects [17, 3, 13, 21, 6], and expanded to cover the full datapath functionality of the Execution Cluster EEX, a top-level component of a core responsible for the functional behaviour of all microinstructions. In the current paper we discuss further expansion of this work on Intel® Core™ i7 design [1]. For this project, we used formal verification as the primary validation vehicle for the execution cluster, including full


the projects, and the second by the fact that the control verification work that would have identified the unintended interference between the operations had not been done yet at the time. To summarize, out of the five misses, three could be attributed to an incorrect formal specification, and two to formal verification work not being completed early enough. The positive side of this is that there were no issues that would have fallen through the cracks because of failures in our methodology.

6 Formal Verification Value Proposition

The conventional wisdom about formal verification in industrial context is easy to spell out. Simulation yields partial results quickly and progresses reliably in a linear fashion, although reaching full coverage is very hard, and completeness unattainable. Formal verification, on the other hand, while in principle holding the promise of completeness, is in practice woefully capacity constrained and either slow or downright unable to produce meaningful results. Although a caricature, we feel this view is not altogether unjustified. To better understand the barriers of more widespread application of formal verification in industry, at least from an Intel perspective, let us look briefly at some possible application models for formal verification:

- FV may be applied to the fundamental algorithms,
- FV may be applied as an extra layer of protection,
- FV may be mixed with dynamic simulation on the same design, or
- FV may replace simulation as the primary validation approach.

In the first usage model, formal and dynamic validation do not directly overlap. Usually, dynamic validation cannot start until an implementation has been coded, and validation of the underlying algorithms is done only by inspection and reviews. Recent forays into such early microarchitecture validation in Intel [4] have been very encouraging. As discussed above, much of Intel’s formal verification work has historically followed the second usage model, where formal verification is done on top of a full dynamic validation effort. There are several pragmatic problems in this approach. First, if dynamic validation is done diligently, it will find most of the bugs, and thereby get most of the credit. Second, the few remaining bugs are likely to be in extreme corners of the design, and formal verification will look at those only if a very thorough and costly effort is made to cover all aspects of the design. This means that doing a little formal verification will not find any new issues, and doing a thorough effort only a few, in both cases leading to a perceived low return on investment. The areas where projects have routinely chosen to do formal verification have then been limited to those where an uncaught problem would be so visible and costly that the extra effort of doing formal verification can be justified. As a positive exception, SAT-based bounded model checking has been very successfully used as a bug-finding tool in targeted areas.

The third usage model, mixing formal and dynamic techniques on validating a single design area, sounds appealing at face value. However, the following fundamental problem makes it hard to offset the dynamic validation effort by formal verification: The coverage-based validation paradigm is based on the identification of all interesting aspects of the design and the sets of interesting cases for all these aspects, with the
Impact

- widespread use in industry (EDA)
  - industry embraced bounding part immediately
  - original *industrial* reservations: using SAT vs ATPG
  - original *academic* reservations: incompleteness?
- BMC relies on efficient SAT (SMT) solving
  - breakthroughs in SAT: CDCL '96, VSIDS '01, ...
  - encouraged investment in SAT / SMT research
- extensions to *completeness*
  - diameter checking, $k$-induction, interpolation
  - SAT based model checking *without* unrolling: IC3
- extensions to *non-boolean* domains
  - infinite state systems
  - bounding reduces complexity & increases decidability
  - software

CBMC
BMC as Enabler

- Interpolants
- IC3
- Induction
- Bug-hunting
- SAT
- BMC
- SMT
- Theorem Proving
- Deduction
- Verification completeness
- Software
- Expressivity
- Cyber physical systems
BMC for Software

- 2001: Edmund Clarke hired Daniel Kroening as Post-Doc
  
  **Use SAT for Software Model Checking!** (expertise on Theorem Proving)
  
  
- 2003: Paper on CBMC for HW/SW co-verification, first at ASP-DAC, then DAC
  
- 2004: Tool paper
  
- 2008: FShell Paper by Michael Tautschnig, while student in Helmut Veith’s group
  
- 2011: Daniel Kroening hired Michael Tautschnig as Post-Doc
  
- 2018: Release 5.9 -- diff to 5.8 has >300k lines
A Tool for Checking ANSI-C Programs

Edmund Clarke, Daniel Kroening, and Flavio Lerda
Carnegie Mellon University

Abstract. We present a tool for the formal verification of ANSI-C programs using Bounded Model Checking (BMC). The emphasis is on usability: the tool supports almost all ANSI-C language features, including pointer constructs, dynamic memory allocation, recursion, and the \texttt{float} and \texttt{double} data types. From the perspective of the user, the verification is highly automated: the only input required is the BMC bound. The tool is integrated into a graphical user interface. This is essential for presenting long counterexample traces: the tool allows stepping through the trace in the same way a debugger allows stepping through a program.

1 Introduction

We present a tool that uses Bounded Model Checking to reason about low-level ANSI-C programs. There are two applications of the tool: 1) the tool checks safety properties such as the correctness of pointer constructs, and 2) the tool can compare an ANSI-C program with another design, such as a circuit given in Verilog.
A tool for checking ANSI-C programs

Authors: Edmund Clarke, Daniel Kroening, Flavio Lerda
Publication date: 2004
Journal: Tools and Algorithms for the Construction and Analysis of Systems
Pages: 168-176
Publisher: Springer Berlin/Heidelberg

Description: We present a tool for the formal verification of ANSI-C programs using Bounded Model Checking (BMC). The emphasis is on usability; the tool supports almost all ANSI-C language features, including pointer constructs, dynamic memory allocation, recursion, and the float and double data types. From the perspective of the user, the verification is highly automated: the only input required is the BMC bound. The tool is integrated into a graphical user interface. This is essential for presenting long counterexample traces: the tool allows stepping through the trace in the same way a debugger allows stepping through a program.

Total citations: Cited by 1272

Source: Google Scholar, 28th June 2018
SATABS: SAT-based predicate abstraction for ANSI-C

Authors: Edmund Clarke, Daniel Kroening, Natasha Sharygina, Karen Yorav

Publication date: 2005/4/4

Conference: International Conference on Tools and Algorithms for the Construction and Analysis of Systems

Pages: 570-574

Publisher: Springer, Berlin, Heidelberg

Description: This paper presents a model checking tool, SatAbs, that implements a predicate abstraction refinement loop. Existing software verification tools such as Slam, Blast, or Magic use decision procedures for abstraction and simulation that are limited to integers. SatAbs overcomes these limitations by using a SAT-solver. This allows the model checker to handle the semantics of the ANSI-C standard accurately. This includes a sound treatment of bit-vector overflow, and of the ANSI-C pointer arithmetic constructs.

Total citations: Cited by 336

Source: Google Scholar, 28th June 2018

CBMC
Behavioral consistency of C and Verilog programs using bounded model checking

<table>
<thead>
<tr>
<th>Authors</th>
<th>Edmund Clarke, Daniel Kroening, Karen Yorav</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publication date</td>
<td>2003/6/2</td>
</tr>
<tr>
<td>Pages</td>
<td>368-371</td>
</tr>
<tr>
<td>Publisher</td>
<td>IEEE</td>
</tr>
<tr>
<td>Description</td>
<td>We present an algorithm that checks behavioral consistency between an ANSI-C program and a circuit given in Verilog using Bounded Model Checking. Both the circuit and the program are unwound and translated into a formula that represents behavioral consistency. The formula is then checked using a SAT solver. We are able to translate C programs that include side effects, pointers, dynamic memory allocation, and loops with conditions that cannot be evaluated statically. We describe experimental results on various reactive circuits and programs, including a small processor given in Verilog and its Instruction Set Architecture given in ANSI-C.</td>
</tr>
<tr>
<td>Total citations</td>
<td>Cited by 323</td>
</tr>
</tbody>
</table>

Source: Google Scholar, 28th June 2018
Applied Impact of CBMC

- 2011: HVC Award, recognizing the most promising academic and industrial contribution to the fields of testing and software and hardware verification from the preceding five years.
- 2014: CBMC overall winner of the Software Verification Competition (TACAS SV-COMP)
- Since 2014: BTC ships CBMC as part of their Embedded Tester product
- 2016: Diffblue Ltd spin-out founded
- 2017: CBMC best bug-finder in TACAS SV-COMP
- 2018: five of the six contestants in SV-COMP’s Concurrency category use CBMC or are forks of CBMC, eight of 21 tools participating in SV-COMP use CBMC or are forks
- Industrial users include Amazon, ARM, TATA, Toyota
Lessons

- simple but very useful ideas are highly controversial
  - hard to get accepted (literally)
  - also got many comments of the sort: *we did this before ...*
  - main points: don’t be afraid, make it work, show that it works!

- in retrospective
  - complexity classification considerations (so more theory)
    might have been useful since we tried to use SAT for symbolic model
    checking without taking Savitch's theorem into account
  - but might also have prevented us going along that route ...
SAT Based Model Checking

- BMC
- $k$-induction
- Abstractions / CEGAR
- Interpolation
- IC3

Armin Biere, Daniel Kröning
SAT Based Model Checking
Handbook of Model Checking