SAT Based Model Checking

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

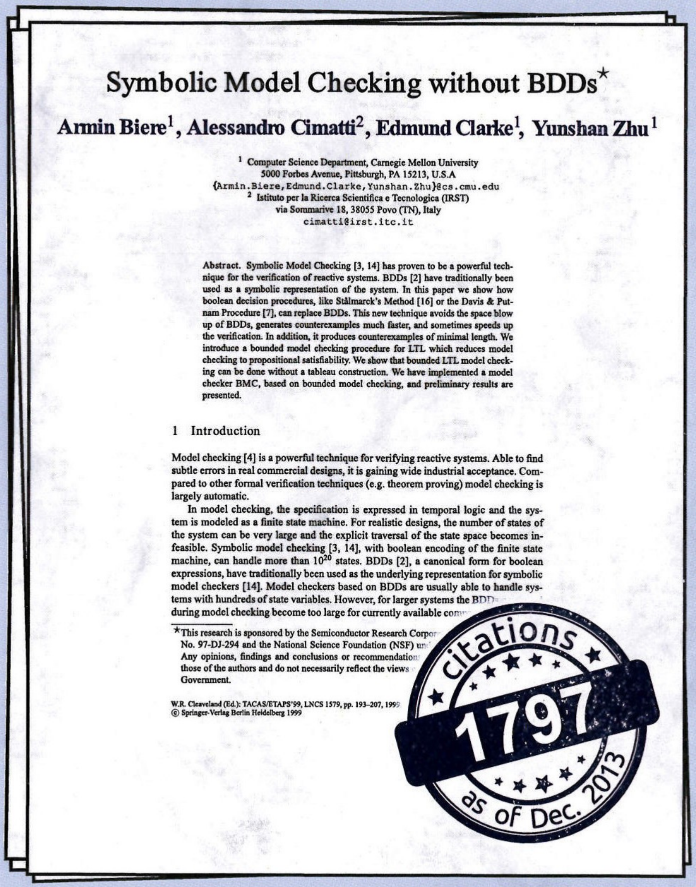
Armin Biere, Daniel Kröning

SAT Based Model Checking

Handbook of Model Checking

AWARD

Most influential paper
in the first 20 years of TACAS



Symbolic Model Checking without BDDs*

Armin Biere[1], Alessandro Cimatti[2], Edmund Clarke[1], Yunshan Zhu[1]

[1] Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A
{Armin.Biere,Edmund.Clarke,Yunshan.Zhu}@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.

1 Introduction

Model checking [4] is a powerful technique for verifying reactive systems. Able to find subtle errors in real commercial designs, it is gaining wide industrial acceptance. Compared to other formal verification techniques (e.g. theorem proving) model checking is largely automatic.

In model checking, the specification is expressed in temporal logic and the system is modeled as a finite state machine. For realistic designs, the number of states of the system can be very large and the explicit traversal of the state space becomes infeasible. Symbolic model checking [3, 14], with boolean encoding of the finite state machine, can handle more than 10^{20} states. BDDs [2], a canonical form for boolean expressions, have traditionally been used as the underlying representation for symbolic model checkers [14]. Model checkers based on BDDs are usually able to handle systems with hundreds of state variables. However, for larger systems the BDD
during model checking become too large for currently available com

*This research is sponsored by the Semiconductor Research Corpor
No. 97-DJ-294 and the National Science Foundation (NSF) un
Any opinions, findings and conclusions or recommendation
those of the authors and do not necessarily reflect the views
Government.

citations
1797
as of Dec. 2013

April 8th 2014, Grenoble

W.R. Cleaveland II [illegible] Lenore Zuck Kim Larsen

[illegible] [illegible]

The Steering Committee of TACAS

Symbolic Model Checking without BDDs*

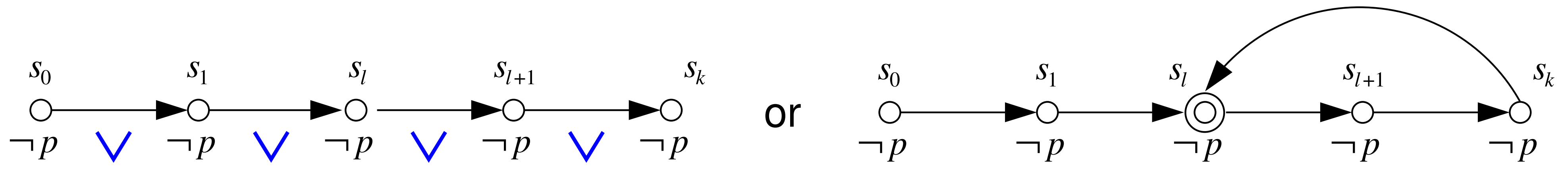
Armin Biere[1], Alessandro Cimatti[2], Edmund Clarke[1], and Yunshan Zhu[1]

[1] Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A
{Armin.Biere,Edmund.Clarke,Yunshan.Zhu}@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.

Bounded Model Checking [BiereCimattiClarkeZhu-TACAS'99]

- look only for counter example made of k states "k" = bound



- simple for *safety properties* p invariantly true

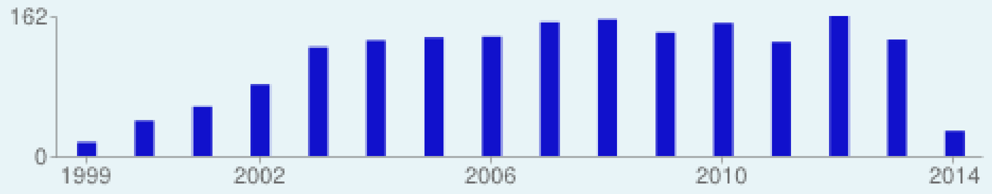
$$I(s_0) \wedge T(s_0, s_1)) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

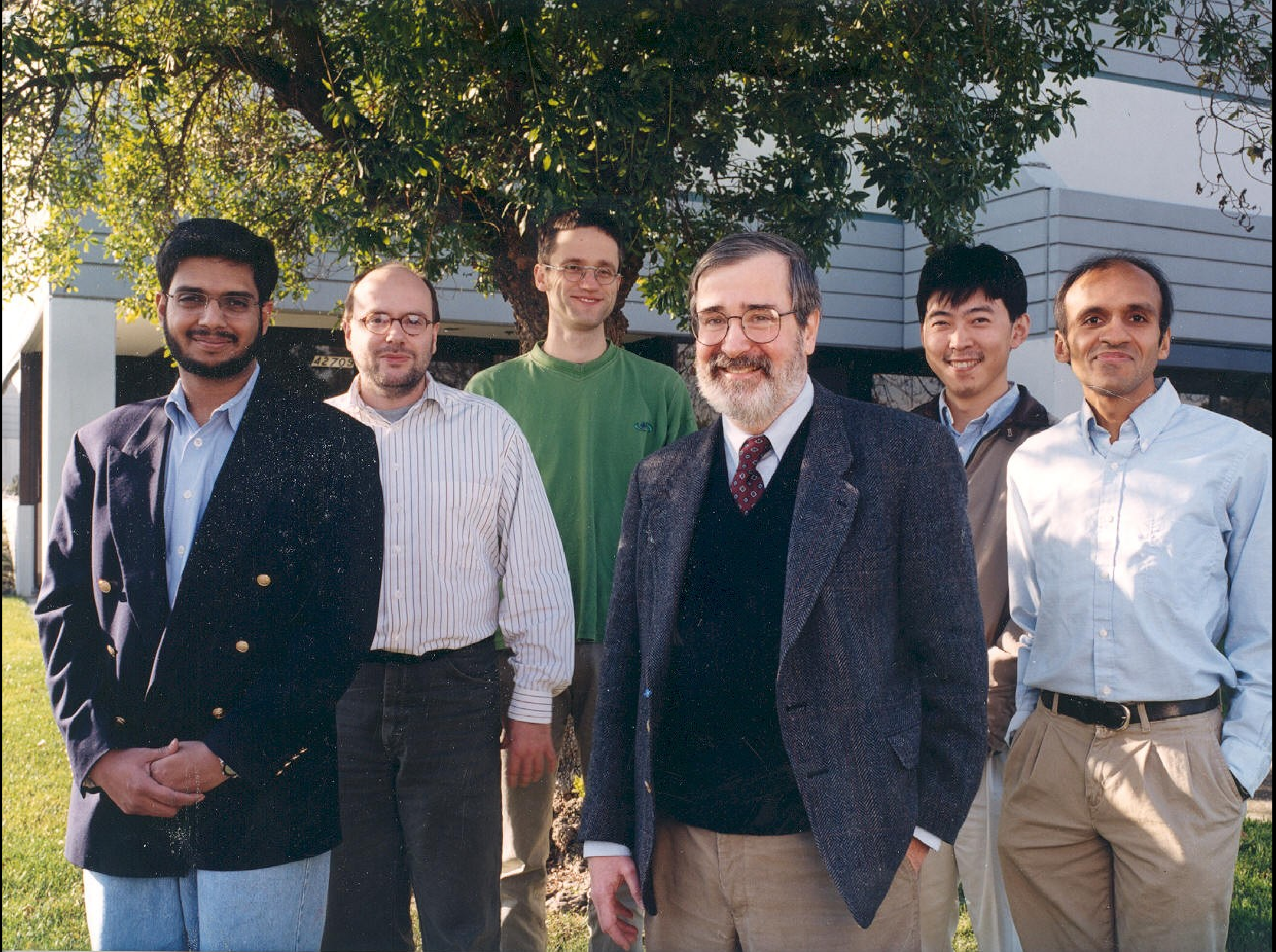
- harder for *liveness properties* p eventually true

$$I(s_0) \wedge T(s_0, s_1)) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \bigwedge_{i=0}^{k} \neg p(s_i) \wedge \bigvee_{l=0}^{k} T(s_k, s_l)$$

- compute and bound k by diameter

Titre	Symbolic model checking without BDDs	[PDF] à partir de cmu.edu
Auteurs	Armin Biere, Alessandro Cimatti, Edmund Clarke, Yunshan Zhu	
Date de publication	1999/1/1	
Titre du livre	Tools and Algorithms for the Construction and Analysis of Systems	
Pages	193-207	
Éditeur	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 ...	

Nombre total de citations	Cité 1829 fois
Nombre de citations par an	
Articles Google Scholar	Symbolic model checking without BDDs A Biere, A Cimatti, E Clarke, Y Zhu - Tools and Algorithms for the Construction and Analysis ..., 1999 Cité 1829 fois - Autres articles - Les 38 versions


42709

Replacing Testing with Formal Verification in Intel® Core™ i7 Processor Execution Engine Validation

Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer,
Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor,
Vladimir Frolov, Erik Reeber, and Armaghan 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 as the primary validation vehicle for the 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, industrially 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 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 EXE, 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 wide-spread 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. Secondly, the few remaining bugs are likely to be in extreme corners of the design, and formal verification will look at these 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-hunting 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 *non-boolean* domains
  - bounding reduces complexity / decidability
- extensions to *completeness*
  - diameter checking, *k*-induction, interpolation
  - SAT based model checking *without* unrolling: IC3

A Short Story on 15 years of Bounded Model Checking

Armin Biere, Alessandro Cimatti, Edmund Clarke, Yunshan Zhu

TACAS'99 - TACAS'14

- 1997: interest and capacity of BDDs stalled
  but there were success stories of other techniques
- *Ed Clarke* hired *Yunshan Zhu* and AB as Post-Docs:
  Use SAT for Symbolic Model Checking!
- struggled for 10 months to come up with something that could replace / improve BDDs (mainly looked at QBF then)
- *Alessandro Cimatti* went to an AI conference in Pittsburgh and at lunch (at an Indian Restaurant) we realized, that in AI Planing 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 and benchmarking later: *TACAS submission*

Lessons

- simple but useful ideas are *very* controversial
  - hard to get accepted (literally)
  - many comments of the sort: *we did this before* …
  - main points: make it work, show that it works!
- in retrospective
  - classification considerations might have been useful since we tried to use SAT for symbolic model checking without taking Savitch's theorem into account
  - but might have prevented us going along that route ...