Model Checking, SAT and Bit-Vectors

Armin Biere
Johannes Kepler University
Linz, Austria

Seminar Theoretical Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden

Monday, 30rd November, 2015
Industrially Successfull Formal Verification Techniques

- **Symbolic Execution**
  - particularly in combination with concolic testing
  - impressive project SAGE at Microsoft (Patrice Godefroid)

- **Equivalence Checking**
  - first widely adopted formal technique in HW verification
  - originally check combinational equivalence of RTL versus gate-level
  - first use since mid 90’ies, wide-spread adoption since 2000
  - since 10 years applied to sequential equivalence too
  - used for checking last-minute fixes: engineering change orders (ECOs)
  - checking arithmetic circuits, e.g., multipliers, still not completely automatic

- **Abstract Interpretation**
  - used to check for number / floating-point overflows etc.
  - static analyzer Astrée verified Airbus flight SW (2003)

- **Model Checking or Property Checking** increasingly used by certain companies
Personal Model Checking History

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BiereCimattiClarkeZhu’99: Bounded Model Checking

SheeranSinghStalmarck’00: k–induction

ClarkeEmersonSifakis: Turing Award 2007

McMillan’03: Interpolation

Bradley’10: IC3

Kurshan’93: Localization

BiereArthoSchuppan’01: Liveness2Safety

ClaessenSorensson’12: k–liveness

McMillan’93: SMV

GrafSaidi’97: Predicate Abstraction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

McMillan’93: SMV

ClaessenSorensson’12: k–liveness

Pnueli’77: Temporal Logic

BiereArthoSchuppan’01: Liveness2Safety

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

Pnueli’77: Temporal Logic

McMillan’03: Interpolation

Bradley’10: IC3

Kurshan’93: Localization

McMillan’93: SMV

BiereArthoSchuppan’01: Liveness2Safety

McMillan’93: SMV

SheeranSinghStalmarck’00: k–induction

ClarkeEmersonSifakis: Turing Award 2007

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

ClarkeGrumbergJahLuVeith’03: CEGAR

BiereCimattiClarkeZhu’99: Bounded Model Checking

SheeranSinghStalmarck’00: k–induction

ClarkeEmersonSifakis: Turing Award 2007

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

McMillan’93: SMV

SheeranSinghStalmarck’00: k–induction

ClarkeEmersonSifakis: Turing Award 2007

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction

BurchClarkeMcMillanDillHwang’90: Symbolic Model Checking

CoudertMadre’89: Symbolic Reachability

Pnueli’77: Temporal Logic

ClarkeEmerson’82: Model Checking

Kurshan’93: Localization

QuielleSifakis’82: Model Checking

BallRajamani’01: SLAM

Holzmann’91: SPIN

GrafSaidi’97: Predicate Abstraction

Holzmann’81: On–The–Fly Reachability

Peled’94: Partial–Order–Reduction
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.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

- look only for counter example made of \( k \) states \( "k" = \text{bound} \)

- simple for safety properties \( p \) invariantly true

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

- harder for liveness properties \( p \) eventually true

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

- compute and bound \( k \) by diameter
Symbolic model checking without BDDs

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

Publication date: 1999/1/1

Book: 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...

Total citations: Cited by 2076

Scholar articles:
Symbolic model checking without BDDs
A Biere, A Cimatti, E Clarke, Y Zhu - Tools and Algorithms for the Construction and Analysis ..., 1999
Cited by 2076 - Related articles - All 38 versions
Replacing Testing with Formal Verification in Intel® Core™ i7 Processor Execution Engine Validation

Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whitemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimír Frolov, Erik Rechter, 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 microarchitectures. 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

6 Formal Verification Value Proposition

The conventional wisdom about formal verification in industry content is now to pull out all stops and buy additional software or new people dedicated to doing a thorough formal verification. This is often ambitious, and doing a thorough effort only 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, is interesting. This approach leaves guard on all the inherent advantages of the formal verification approach and the dynamic testing approach in a single validation project.
Impact of BMC

- 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 and SW
  - 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

• 1997: interest and capacity of BDDs stalled but there were success stories of other techniques
• Ed Clarke hired Yunshan Zhu & Armin Biere 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 came 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
Symbolic Model Checking without BDDs

Armin Biere\textsuperscript{1}, Alessandro Cimatti\textsuperscript{2}, Edmund Clarke\textsuperscript{1}, Yunshan Zhu\textsuperscript{1}

\textsuperscript{1} Carnegie Mellon University, Pittsburgh, PA 15213, USA
\textsuperscript{2} University of Milano Bicocca, Milan, Italy

Abstract: Symbolic Model Checking (SMC) is a powerful technique for the verification of reactive systems. BDDs have traditionally been used as a symbolic representation of the system. In this paper we show how boolean decision diagrams, the BDD Method (BM), are superior to BDDs. This new method avoids the space blowup and the „computation on fly“ behavior that appears when working with BDDs. In addition, it produces a compact representation of the state space. We introduce a bounded model checking procedure for BM, which allows model checking and propositional satisfiability checking to be performed efficiently. We demonstrate the practical benefits of BM over BDDs on a set of hardware and software benchmarks.

1 Introduction

Model checking\cite{Clarke2004} is a powerful technique for verifying reactive systems. It is able to deal with large state spaces and unbounded temporal properties. Compared to other formal verification techniques (e.g., theorem proving), model checking is more automated.

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 representation of the state space becomes infeasible. Symbolic model checking\cite{Biere2004b,Cimatti2004}, with boolean encoding of the finite state machines, can handle more than 10\textsuperscript{9} states. BM\cite{Biere2004a}, a compact form for boolean expressions, has been widely used as an underlying representation for symbolic model checking. BM can handle larger state spaces than BDDs are usually able to handle even when working with boolean decision diagrams that are at least 4 times sparser than corresponding BDDs. During model checking BM is often twice as fast as currently available tools.

This research is sponsored by the Instrumentation Commission of the European Union and the National Science Foundation (NSF) under Grant EIA-0102307. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

April 8th, 2014, Grenoble

W. Cleveland, J. M. Shead, A. J. Zuck, T. J. C. Leemis

The Steering Committee of TACAS
SAT Based Model Checking

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

Abstract Modern satisfiability (SAT) solvers have become the enabling technology of many Model Checkers. In this chapter, we will focus on those techniques most relevant to industrial practice. In Bounded Model Checking (BMC), a transition system and a property are jointly unwound for a given number $k$ of steps to obtain a formula that is satisfiable if there is a counterexample for the property up to length $k$. The formula is then passed to an efficient SAT solver. The strength of BMC is refutation: BMC has been used to discover subtle flaws in digital systems. We cover the application of BMC to both hardware and software systems, and to hardware/software co-verification. We also discuss means to make BMC complete, including $k$-induction, Craig interpolation, abstraction refinement techniques and inductive techniques with iterative strengthening.
Lessons from BMC

● 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 ...
Some Complexity Classes

- **P**
  - problems with polynomially \textbf{time}-bounded algorithms
  - bounds measured in terms of input (file) size

- **NP**
  - same as P but with non-deterministic choice
  - needs a SAT solver

- **PSPACE**
  - as P but \textbf{space}-bounded
  - QBF and bit-level model checking fall in this class

- **NEXPTIME**
  - same as NP but with exponential time

- **P \subseteq NP \subseteq PSPACE \subseteq NEXPTIME**
  - usually it is assumed: \( P \neq NP \)
  - it is further known: \( NP \neq NEXPTIME \)
Complexity Concretely

- **NP problems**
  - anything which can be (polynomially) encoded into **SAT**
  - combinational equivalence checking, bounded model checking

- **PSPACE problems**
  - anything which can be encoded (polynomially) into **QBF**
  - or into (bit-level) **symbolic model checking**
  - sequential equivalence checking, combinational synthesis or bounded games

- **NEXPTIME problems**
  - anything which can be encoded **exponentially** into SAT
  - first-order logic Bernays-Schönfinkel class (**EPR**): no functions, $\exists^*\forall^*$ prefix
  - QBF with explicit dependencies (Henkin Quantifiers): **DQBF**
  - partial observation games, black-box bounded model checking
  - bit-vector logics: **QF_BV**
QF_BV contained in NEXPTIME
- bit-blast (exponential)
- give resulting formula to SAT solver

we showed QF_BV is NEXPTIME hard by reducing DQBF to QF_BV
\[ \forall x_0, x_1, x_2, x_3, x_4 \exists e_0(x_0, x_1, x_2, x_3), e_1(x_1, x_2, x_3, x_4) \varphi \]
- polynomially encodes dependencies (for Henkin quantifiers)
- my student Andreas has now an (yet unpublished) direct proof

why are bit-vectors NEXPTIME complete?

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 1000000))
(declare-fun y () (_ BitVec 1000000))
(declare-fun z () (_ BitVec 1000000))
(assert (= z (bvadd x y)))
(assert (= z (bvshl x (_ bv1 1000000))))
(assert (distinct x y))
```

```latex
x, y : \text{bool}[1000000] \quad y \neq x \land x+y = x \ll 1
```
NP complete: $\text{QF}\_\text{BV}_{bw}$
- **relate same bits**: equality and all bit-wise operators
- similar to well-known Ackermann reduction

PSPACE complete: $\text{QF}\_\text{BV}_{bw,<<1}$
- only allow operators which **relate neighbouring bits**:
  - base operators: equality, inequality/comparison, bit-wise ops, shift-by-one
  - extended operators: addition, multiplication by constants, single-bit-slices etc.
- encode in symbolic model checking logarithmically in bit-width

see our CSR’12, SMT’13 papers and our 2015 journal article in TOCS

came across otherwise unsolvable benchmarks from industry!
MODULE main

VAR
  c : boolean; -- carry 'bvadd x y'
  d : boolean; -- carry 'bvadd y x'
  x : boolean; -- x0, x1, ...
  y : boolean; -- y0, y1, ...

ASSIGN
  init (c) := FALSE;
  init (d) := FALSE;

ASSIGN
  next (c) := c&x | c&y | x&y; -- c + x + y >= 2
  next (d) := d&y | d&x | y&x; -- d + y + x >= 2

DEFINE
  o := c != (x != y); -- c xor y xor x
  p := d != (y != x); -- d xor x xor y

SPEC
  AG (o = p)
Commutativity of Bit-Vector Addition in AIGER

Model Checking, SAT and Bit-Vectors KTH Stockholm
Ripple-Carry-Adder vs Carry-Save-Adder

not really realistic example but shows the fundamental problem of checking arithmetic circuit equivalence
Commutativity of Bit-Vector Multiplication

(set-logic QF_BV)
(declare-fun x () (_ BitVec 12))
(declare-fun y () (_ BitVec 12))
(assert (distinct (bvmul x y) (bvmul y x)))
(check-sat)

<table>
<thead>
<tr>
<th>bits</th>
<th>Glucose</th>
<th>Lingeling</th>
<th>March</th>
<th>iLingeling</th>
<th>Treengeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>0.00</td>
<td>0.01</td>
<td>0.05</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>0.02</td>
<td>0.03</td>
<td>0.36</td>
<td>0.31</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>0.14</td>
<td>0.27</td>
<td>0.63</td>
<td>0.72</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>1.18</td>
<td>1.98</td>
<td>1.38</td>
<td>2.47</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>7.85</td>
<td>10.98</td>
<td>2.63</td>
<td>4.65</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>37.16</td>
<td>41.49</td>
<td>5.02</td>
<td>10.86</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>147.62</td>
<td>214.98</td>
<td>15.72</td>
<td>21.96</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>833.62</td>
<td>649.49</td>
<td>56.57</td>
<td>61.48</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>--</td>
<td>--</td>
<td>238.10</td>
<td>263.44</td>
<td></td>
</tr>
</tbody>
</table>

limit of 900 seconds wall clock time
secret of the success of (combinational) equivalence checking
  - assumption: many internal equivalence points
  - makes BDD and SAT sweeping effective

problems with arithmetic circuits
  - almost no equivalent internal signals (except for outputs)
  - proof complexity conjectured to be beyond resolution
  - often no "clean" implementation circuit available

challenges
  - prove conjectured complexity
  - use world-level (bit-vector) information
  - arithmetic reasoning on the bit-level
  - robust integration in SAT and/or SMT solver

started to collect a large number of such benchmarks
Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

CPU Time (in seconds)

Number of problems solved
Satisfiability (SAT) related topics have attracted researchers from various disciplines. Logic, applied areas such as planning, scheduling, operations research, and combinatorial optimization, but also theoretical issues on the theme of complexity, and much more, they all are connected through SAT.

My personal interest in SAT stems from actual solving. The increase in power of modern SAT solvers over the past 15 years has been phenomenal. It has become the key enabling technology in automated verification of both computer hardware and software. Bounded Model Checking (BMC) of computer hardware is now probably the most widely used model checking technique. The counterexamples that it finds are just satisfying instances of a first-order formula obtained by extending to some fixed depth a sequential circuit and its specification in linear temporal logic. Extending model checking to software verification is a much more difficult problem on the frontier of current research. One promising approach for languages like C with finite word-length integers is to use the same idea as in BMC, but with a decision procedure for the theory of bit-vectors instead of SAT. All decision procedures for bit-vectors that I am familiar with ultimately make use of a fast SAT solver to handle complex formulas.

Decision procedures for more complicated theories, like linear real and integer arithmetic, are also used in program verification. Most of them use powerful SAT solvers in an essential way.

Clearly, efficient SAT solving is a key technology for 21st century computer science. I expect this collection of papers on all theoretical and practical aspects of SAT solving will be extremely useful to both students and researchers and will lead to many further advances in the field.

Edmund Clarke
Part I. Theory and Algorithms

John Franco, John Martin:
A History of Satisfiability. 3-74

Steven David Prestwich:
CNF Encodings. 75-97

Adnan Darwiche, Knot Pipatsrisawat:
Complete Algorithms. 99-130

João P. Marques Silva, Inês Lynce, Sharad Malik:
Conflict-Driven Clause Learning SAT Solvers. 131-153

Marilyn Heule, Hans van Maaren:
Look-Ahead Based SAT Solvers. 155-184

Henry A. Kautz, Ashish Sabharwal, Bart Selman:
Incomplete Algorithms. 185-203

Oliver Kullmann:
Fundamentals of Branching Heuristics. 205-244

Dimitris Achlioptas:
Random Satisfiability. 245-270

Carla P. Gomes, Ashish Sabharwal:
Exploiting Runtime Variation in Complete Solvers. 271-288

Karem A. Sakallah:
Symmetry and Satisfiability. 289-338

Hans Kleine Büning, Oliver Kullmann:
Minimal Unsatisfiability and Autarkies. 339-401

Evgeny Dantsin, Edward A. Hirsch:
Worst-Case Upper Bounds. 403-424

Marko Samer, Stefan Szeider:
Fixed-Parameter Tractability. 425-454

Part II. Applications and Extensions

Armin Biere:
Bounded Model Checking. 457-481

Jussi Rintanen:
Planning and SAT. 483-504

Daniel Kroening:
Software Verification. 505-532

Hantao Zhang:
Combinatorial Designs by SAT Solvers. 533-568

Fabrizio Altarelli, Rémi Monasson, Guilhem Semerjian, Francesco Zamponi:
Connections to Statistical Physics. 569-611

Chu Min Li, Felip Manyà:
MaxSAT, Hard and Soft Constraints. 613-631

Carla P. Gomes, Ashish Sabharwal, Bart Selman:
Model Counting. 633-654

Rolf Drechsler, Tommi A. Junttila, Ilkka Niemelä:
Non-Clausal SAT and ATPG. 655-693

Olivier Roussel, Vasco M. Manquinho:
Pseudo-Boolean and Cardinality Constraints. 695-733

Hans Kleine Büning, Uwe Bubeck:
Theory of Quantified Boolean Formulas. 735-760

Enrico Giunchiglia, Paolo Marin, Massimo Narizzano:
Reasoning with Quantified Boolean Formulas. 761-780

Roberto Sebastiani, Armando Tacchella:
SAT Techniques for Modal and Description Logics. 781-824

Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, Cesare Tinelli:
Satisfiability Modulo Theories. 825-885

Stephen M. Majercik:
Stochastic Boolean Satisfiability. 887-925
Special thanks are due to Armin Biere, Randy Bryant, Sam Buss, Niklas Eén, Ian Gent, Marijn Heule, Holger Hoos, Svante Janson, Peter Jeavons, Daniel Kroening, Oliver Kullmann, Massimo Lauria, Wes Pegden, Will Shortz, Carsten Sinz, Niklas Sörensson, Udo Wermuth, Ryan Williams, and... for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections. Thanks also to Stanford’s Information Systems Laboratory for providing extra computer power when my laptop machine was inadequate.

* * *

Wow — Section 7.2.2.2.2 has turned out to be the longest section, by far, in *The Art of Computer Programming*. The SAT problem is evidently a “killer app,” because it is key to the solution of so many other problems. Consequently I can only hope that my lengthy treatment does not also kill off my faithful readers! As I wrote this material, one topic always seemed to flow naturally into another, so there was no neat way to break this section up into separate subsections. (And anyway the format of *TAOCP* doesn’t allow for a Section 7.2.2.2.1.)

I’ve tried to ameliorate the reader’s navigation problem by adding subheadings at the top of each right-hand page. Furthermore, as in other sections, the exercises appear in an order that roughly parallels the order in which corresponding topics are taken up in the text. Numerous cross-references are provided
Competitions, Benchmarks, Science

- Competitions are used to
  - compare and evaluate implementations and algorithms
  - generate benchmarks used in papers

- SAT competition is one of the largest competitions
  - many solvers, highly competitive
  - portfolio solving, over-tuning issues
  - benchmark selection scheme broken due to competing goals:
    - assess the state-of-the-art
    - high-light new ideas
    - give a fair chance to everybody

- Research in SAT solving, verification, etc. in essence empirical science
  - benchmark selection critical
  - how to select benchmarks?
    - for the competition?
    - in your papers?
Conclusion

- what I did not talk about ... (yet)
  - parallel SAT
  - QBF / quantifiers in general
  - huge improvements in local research in recent years
  - how to apply local search to bit-vectors and SMT
  - testing / debugging
  - assertion synthesis

- acknowledgements:
  Ed Clarke, all co-authors, collaborators, students and Post-Docs
  and if would list more names I would struggle with order and probably forget somebody

- if you have model checking, SMT, or SAT problems you want share let me know ...

*looking for Post-Doc's and PhD students too*