SAT Solving for Model Checking and Beyond

Armin Biere
Johannes Kepler University
Linz, Austria

HVC’15
11th Haifa Verification Conference
IBM Research, Haifa, Israel

Thursday, 19th November, 2015
HVC Award

The HVC award is given to the most influential work in the last five years in formal verification, simulation, and testing. The award is not limited to influential articles; it can also be a system or a collection of activities that promote the area.

The HVC award committee has decided to give the award this year to Prof. Armin Biere from Johannes Kepler University in Austria.

Since Biere's PhD graduation in 1997, he made pivotal contributions in formal verification, one of which — bounded model checking — was recently selected as the most influential article in the 20 years of TACAS. This technique is still being used in numerous EDA companies, and also led to similar ideas in verification of software. In addition, he is the developer of numerous award-winning SAT, bitvector, arrays and QBF solvers. Such solvers developed by him or under his guidance rank at the top of many international competitions and were awarded 42 medals including 24 gold medals.

Biere is one of the editors of the 980-pages "Handbook of Satisfiability", the chair of the SAT association, the founder and organizer of the Hardware Model Checking Competition (HWMCC), the inventor of the now ubiquitous AIG format for model-checking, and also has served in the last two years as an informal advisor of D. Knuth for SAT-related issues.

By awarding Prof. Biere, the committee recognizes his major contributions to the formal verification and computational logic communities.
μcke – Efficient μ-Calculus Model Checking

Armin Biere¹
armin@ira.uka.de, Institut für Logik, Komplexität und Deduktionssysteme,
Universität Karlsruhe, Am Fasanengarten 5, D-76128 Karlsruhe, Germany

Abstract. In this paper we present an overview of the verification tool μcke. It is an implementation of a BDD-based μ-calculus model checker and uses several optimization techniques that are lifted from special purpose model checkers to the μ-calculus. This gives the user more expressibility without loosing efficiency.

Introduction

In [5] μ-calculus model checking with BDDs has been proposed as a general framework for various verification problems like model checking of LTL and CTL or testing for bisimulation equivalence and language containment. With a μ-calculus model checker all these verification tasks could be handled with one tool. Also some applications of
The Industrial Success of Verification Tools Based on Stålmarck’s Method

Arne Borälv
arne@lk.se
Logikkonsult NP AB,
Swedenborgsgatan 2,
S-118 48 Stockholm, Sweden,
http://www.lk.se

Abstract. Stålmarck’s Method is a patented natural deduction proof method with a novel proof-theoretic notion of \textit{proof depth}, defined as the largest number of nested assumptions in the proof. An implementation of the method, called Prover, has been used as proof engine in various commercial tools since 1990, and is now integrated in a formal verification framework called NP-Tools. Prover searches for shallow subformula proofs, which has proven to be an efficient strategy for solving many industrial problems, the largest of which today consists of several 100,000’s of sub-formulas. Stålmarck’s method is in industrial use, for instance in the areas of telecom service specification analysis, analysis of
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  
  \[ \text{“} k \text{”} = \text{bound} \]

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

Roope Kaivola, Rajnish Ghugal, Naren Narasimhan, Amber Telfer, Jesse Whitemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladmir Frolov, Erik Reeb, 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 data-path, 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 concerns easy to spell out: it’s expensive and it takes too much time. Nevertheless, in the past few years, formal verification has proven to be a critical part of the digital design process in several projects. It is a good investment for design teams to verify critical parts of the design, and formal verification is acknowledged as a key to success in many projects. However, doing a formal verification project takes a lot of effort and time, and even though effort only as few as ten employees, 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 unexpected 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, has been used to verify critical parts of the design and interactors. Interestingly,
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 Biere¹, Alessandro Cimatti², Edmund Clarke¹, Yunshan Zhu¹

¹ Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
² Politecnico di Torino, Torino, 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, like BDDs, can be used for SMC, and how they provide a more efficient alternative to the traditional approach based on binary decision diagrams. We call this approach Symbolic Model Checking without BDDs (SMCwBDD), or simply SMCwBDD. The main advantage of SMCwBDD is that it can handle systems that are too large for the traditional approach. In addition, a probabilistic measure of reachability is provided. We implement a tool that implements the SMCwBDD approach and compare its performance to a tool that uses SMC based on binary decision diagrams. Our results show that SMCwBDD is significantly faster than SMC based on binary decision diagrams.

1 Introduction

Symbolic model checking [1-4] is a powerful technique for the verification of reactive systems. While it is used in a variety of domains, it is often used in industrial applications. Compared to other formal verification techniques (e.g. bounded model checking), symbolic model checking is more efficient.

In symbolic model checking, the specification is expressed in temporal logic and the system is represented as a finite state machine. For reactive systems, the number of states of the system can be very large and the explicit representation of the state space becomes infeasible. Symbolic model checking [1, 4], with boolean encoding of the finite state machines, can handle more than 10⁶ states. BDDs [5], a concrete form for boolean expressions, have traditionally been used as the underlying representation for symbolic model checking [1-4]. Symbolic model checking based on BDDs are usually faster to handle even if the state space size is large. However, BDDs are not always faster as the BDD representation of the state space can be large. In general, using BDDs for symbolic model checking is too large for many currently available approaches.

This work was supported by the National Science Foundation (NSF) under grant No. 0721589 and the National Security Foundation (NSF) under grant No. 0721589. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Government.

April 8th 2014, Grenoble

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 **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 **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
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 (single exponentially)
  - 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 has now an (yet unpublished) direct proof

- why are bit-vectors NEXPTIME complete?

```lisp
(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))
```

\[ x, y : \text{bool}[1000000] \]

\[ y \neq x \land x + y = x \ll 1 \]
Bit-Wise Operators and Shifting Neighbouring Bits Only

- **NP complete:** \( QF\_BV_{bw} \)
  - **relate same bits:** equality and all bit-wise operators
  - similar to well-known Ackermann reduction

- **PSPACE complete:** \( QF\_BV_{bw,\ll 1} \)
  - 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;
  next (d) := d & y | d & x | y & x;
DEFINE
  o := c != (x != y);
  p := d != (y != x);
SPEC
  AG (o = p)
Hardware Model Checking Competition (HWMCC)

- founding lunch at CAV’06, first competition at CAV’07
- HWMCC lunch at FMCAD’08 ⇒ need multiple properties !!!
- affilliated with either CAV (7,8,10,14) or FMCAD (11,12,13,15)
- HWMCC’11: old SINGLE, new LIVEness and new MULTI property track
- HWMCC’12 as HWMCC’11, new DEEP bounds track
- in essence no change in HWMCC’12 - HWMCC’15
- HWMCC’15: DEEP, SINGLE, and LIVE, MULTI, 1h time limit, before 15min
- **abcsimple, abcsimplive, abcsuprove** from Berkeley
  - Brayton, Sterin, Mishchenko, ...
- **aigbmc, blimc** from JKU Linz
  - Biere
- **avy** from Technion+SEI+Princeton
  - Vizel, Gurfinkel, Malik
- **iimc** from Boulder
  - Somenzi, Bradley, Hassan
- **iprover(hc), iproverdeep(hc)** from Manchester
  - Tsarkov, Korovin
- **nuxmv, nuxmvbmc** from Trento
  - Griggio, Roveri, ...
- **pdtravdeep, pdtravthrd** from Torino
  - Cabodi, Quer, ...
- **ricecnu** from Rice
  - Li, Vardi
- **shifbmc** from Dresden
  - Manthey
- **tip2014, tip2014bmc** from Chalmers
  - Sörensson, Claessen
- **v3s** from Taipei
  - Yang, Wu, Huang
<table>
<thead>
<tr>
<th>rank</th>
<th>name</th>
<th>cnt</th>
<th>ok</th>
<th>sat</th>
<th>uns</th>
<th>fld</th>
<th>to</th>
<th>mo</th>
<th>s11</th>
<th>s6</th>
<th>unk</th>
<th>real</th>
<th>time</th>
<th>max</th>
<th>best</th>
<th>uniq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abcsimple</td>
<td>548</td>
<td>363</td>
<td>122</td>
<td>241</td>
<td>185</td>
<td>154</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>45438</td>
<td>161407</td>
<td>6785</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>abcsuprove</td>
<td>548</td>
<td>358</td>
<td>120</td>
<td>238</td>
<td>190</td>
<td>156</td>
<td>34</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>41596</td>
<td>115597</td>
<td>6905</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>nuxmv</td>
<td>548</td>
<td>357</td>
<td>127</td>
<td>230</td>
<td>191</td>
<td>165</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47600</td>
<td>186902</td>
<td>6900</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>pdtravthrd</td>
<td>548</td>
<td>353</td>
<td>115</td>
<td>238</td>
<td>195</td>
<td>62</td>
<td>124</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>52819</td>
<td>158951</td>
<td>6824</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>avy</td>
<td>548</td>
<td>345</td>
<td>115</td>
<td>230</td>
<td>203</td>
<td>171</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>49724</td>
<td>166878</td>
<td>5924</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>iimc</td>
<td>548</td>
<td>335</td>
<td>105</td>
<td>230</td>
<td>213</td>
<td>163</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>57840</td>
<td>202386</td>
<td>6969</td>
<td>76</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>tip2014</td>
<td>548</td>
<td>315</td>
<td>98</td>
<td>217</td>
<td>233</td>
<td>233</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46817</td>
<td>46596</td>
<td>1062</td>
<td>144</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>v3s</td>
<td>548</td>
<td>296</td>
<td>62</td>
<td>234</td>
<td>252</td>
<td>141</td>
<td>7</td>
<td>102</td>
<td>2</td>
<td>0</td>
<td>54273</td>
<td>152125</td>
<td>6037</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>blimc</td>
<td>548</td>
<td>141</td>
<td>128</td>
<td>13</td>
<td>407</td>
<td>287</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>31687</td>
<td>31584</td>
<td>2241</td>
<td>61</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>aigbmc</td>
<td>548</td>
<td>126</td>
<td>126</td>
<td>0</td>
<td>422</td>
<td>310</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>69</td>
<td>34210</td>
<td>34104</td>
<td>3644</td>
<td>47</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>shiftbmc</td>
<td>548</td>
<td>120</td>
<td>120</td>
<td>0</td>
<td>428</td>
<td>247</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>169</td>
<td>37819</td>
<td>37697</td>
<td>1810</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>tip2014bmc</td>
<td>548</td>
<td>117</td>
<td>117</td>
<td>0</td>
<td>431</td>
<td>225</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>156</td>
<td>11054</td>
<td>10911</td>
<td>5496</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>nuxmrvbmc</td>
<td>548</td>
<td>115</td>
<td>115</td>
<td>0</td>
<td>433</td>
<td>256</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>149</td>
<td>8885</td>
<td>8796</td>
<td>2504</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>iproverhc</td>
<td>548</td>
<td>112</td>
<td>64</td>
<td>48</td>
<td>436</td>
<td>156</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>280</td>
<td>32269</td>
<td>17292</td>
<td>6730</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>pdtravdeep</td>
<td>548</td>
<td>105</td>
<td>46</td>
<td>59</td>
<td>443</td>
<td>345</td>
<td>86</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>14725</td>
<td>14568</td>
<td>4202</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ricecnu</td>
<td>548</td>
<td>100</td>
<td>30</td>
<td>70</td>
<td>440</td>
<td>342</td>
<td>88</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>49153</td>
<td>49054</td>
<td>2204</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>iproverdeephc</td>
<td>548</td>
<td>93</td>
<td>46</td>
<td>47</td>
<td>455</td>
<td>177</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>278</td>
<td>18116</td>
<td>17482</td>
<td>6728</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>iproverdeep</td>
<td>548</td>
<td>86</td>
<td>46</td>
<td>40</td>
<td>462</td>
<td>149</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>313</td>
<td>14899</td>
<td>14818</td>
<td>6322</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>iprover</td>
<td>548</td>
<td>83</td>
<td>43</td>
<td>40</td>
<td>465</td>
<td>59</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>406</td>
<td>8768</td>
<td>8698</td>
<td>6176</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

hors concours (not ranked):

aigbmc blimc: organizer model checkers
pdtravthrd: issue catching ‘FATAL’ for ‘intel045’ (not counted)
ricecnu: reports 8 instances SAT which are UNSAT (not counted)
iprover*hc: last minute (script) fixes after deadline

each team / submitter only ranked once (one medal maximum)
Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

CPU Time (in seconds) vs. Number of problems solved for various SAT solvers in 2009.

From Daniel Le Berre
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 boolean formula obtained by expanding 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 bi-vectors instead of SAT. All decision procedures for bi-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

Edmund Clarke, FORC Systems, University Professor of Computer Science and Professor of Electrical and Computer Engineering at Carnegie Mellon University, is one of the initiators and main contributors to the field of Model Checking, for which he also received the 2007 ACM Turing Award.

In the late 90s, Professor Clarke was one of the first researchers to realize that SAT solving has the potential to become one of the most important technologies in model checking.

Editors:
Armin Biere
Marijn Heule
Hans van Maaren
Toby Walsh

IOS Press
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
- Marijn Heule, Hans van Maaren: *Look-Ahead Based SAT Solvers*. 155-184
- 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 Buning, Oliver Kullmann: *Minimal Unsatisfiability and Autarkies*. 339-401
- 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 A. G. Marson, Remi Monasson, Guilhem Sermi, Francesco Zamponi: *Connections to Statistical Physics*. 569-611
- Chu Min Li, Felix Many: *MaxSAT, Hard and Soft Constraints*. 613-631
- Rolf Drechsler, Tommi A. Junntila, 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 Marini, 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 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 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