Model Checking, SAT and Bit-Vectors

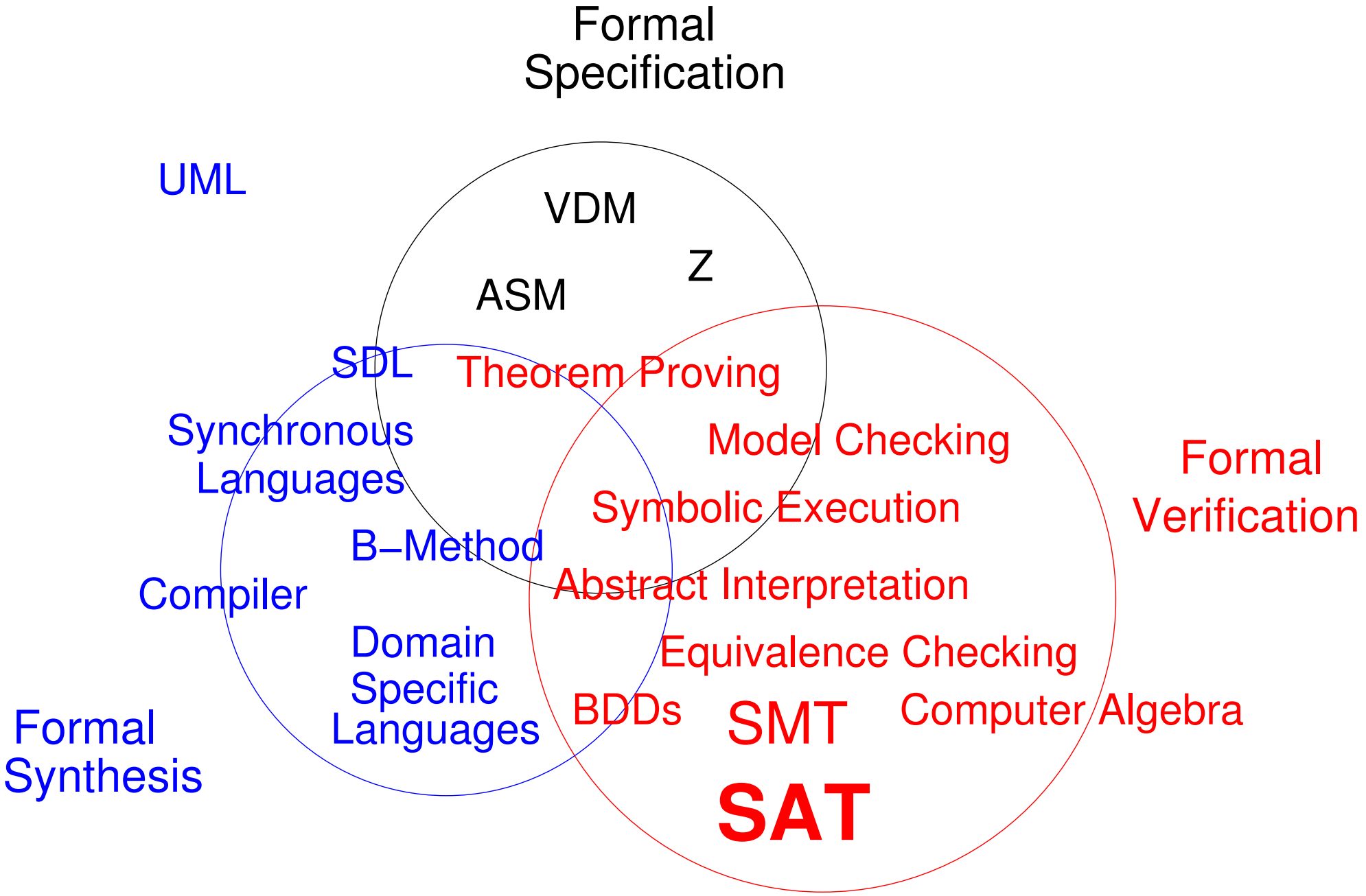
Armin Biere
Johannes Kepler University
Linz, Austria

Seminar Theoretical Computer Science

KTH Royal Institute of Technology

Stockholm, Sweden

Monday, 30rd November, 2015


Formal
Specification
UML
VDM
Z
ASM
SDL
Theorem Proving
Synchronous
Languages
Model Checking
Formal
Verification
Symbolic Execution
B–Method
Compiler
Abstract Interpretation
Domain
Specific
Languages
Equivalence Checking
BDDs
SMT
Computer Algebra
Formal
Synthesis
SAT

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

ClaessenSorensson'12: k–liveness

CoudertMadre'89: Symbolic Reachability

BiereArthoSchuppan'01: Liveness2Safety

Pnueli'77: Temporal Logic

McMillan'03: Interpolation

Bradley'10: IC3

McMillan'93: SMV

BiereCimattiClarkeZhu'99: Bounded Model Checking

ClarkeEmerson'82: Model Checking

SheeranSinghStalmarck'00: k–induction

Kurshan'93: Localization

ClarkeEmersonSifakis:
Turing Award 2007

QuielleSifakis'82: Model Checking

BallRajamani'01: SLAM

Holzmann'91: SPIN

GrafSaidi'97: Predicate Abstraction

Holzmann'81: On–The–Fly Reachability

ClarkeGrumbergJahLuVeith'03: CEGAR

Peled'94: Partial–Order–Reduction

Symbolic Model Checking without BDDs*

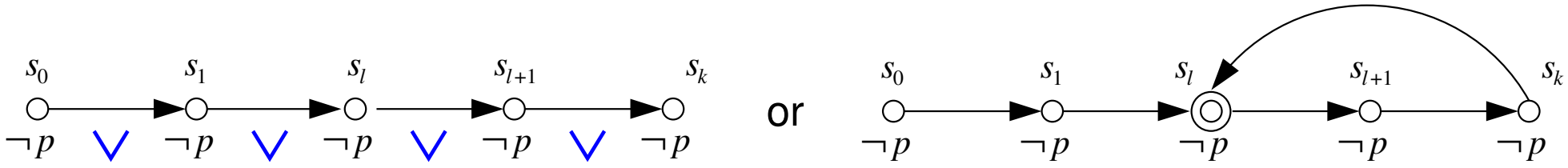
Armin Biere[1], Alessandro Cimatti[2], Edmund Clarke[1], and Yunshan Zhu[1]

[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



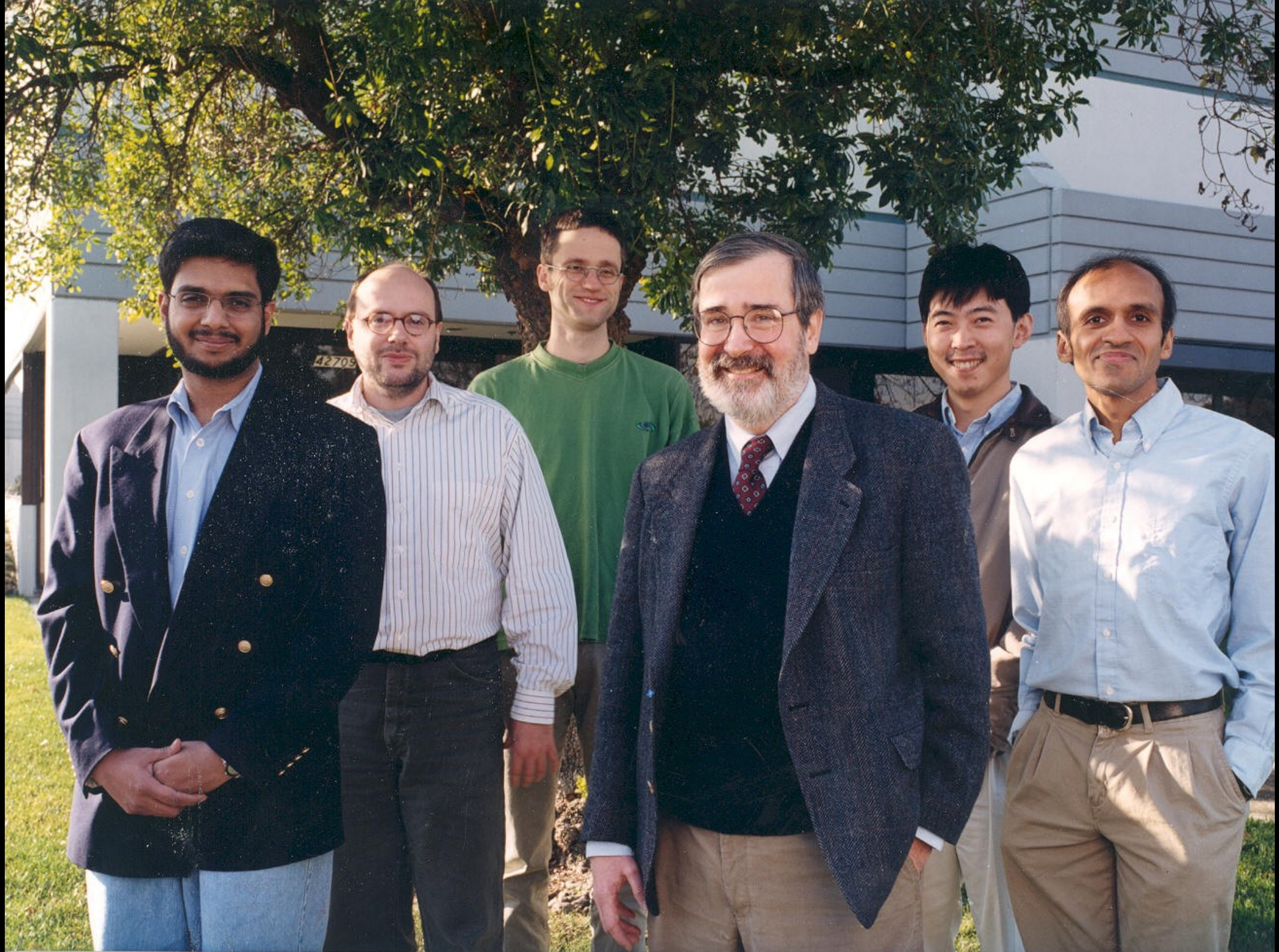
- simple for safety properties p invariantly true

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

- 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



Google

Scholar

Edit

Export

Symbolic model checking without BDDs

[PDF] from cmu.edu

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

Publication date: 1999/1/1

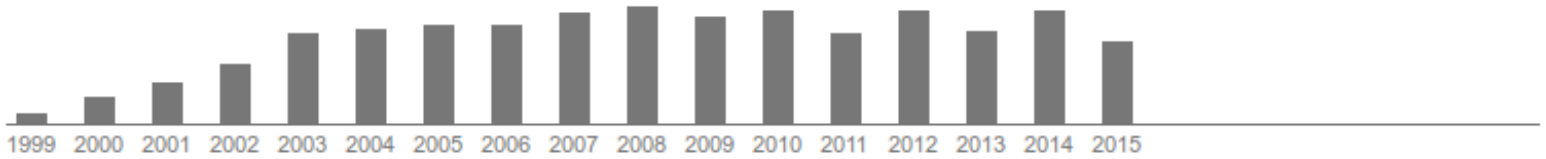
Book: Tools and Algorithms for the Construction and Analysis of Systems

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 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

A. Bouajjani and O. Maler (Eds.): CAV 2009, LNCS 5643, pp. 414–429, 2009.

6 Formal Verification Value Proposition

¬ FV may

of the design, and formal verificatio

that doing a

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

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

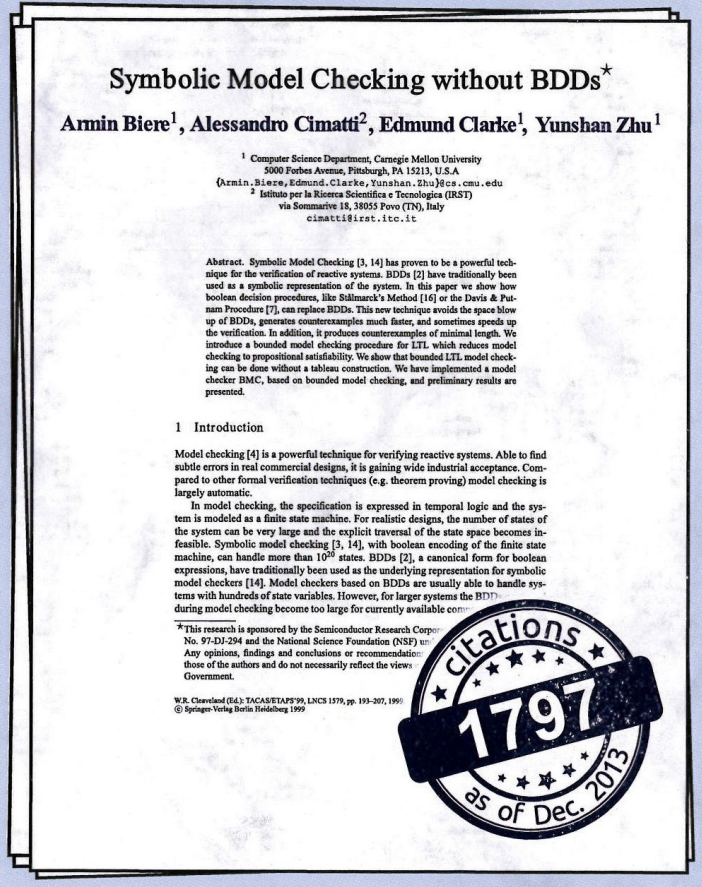
A Short Story on 15 years of

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*

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.

W.R. Cleaveland (Ed.): TACAS/ETAPS'99, LNCS 1579, pp. 193–207, 199...
© Springer-Verlag Berlin Heidelberg 1999

April 8th 2014, Grenoble

W. R. Cleaveland II

Bernhard Steffen

Dexter Kozen

Kim Larsen

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.

SAT Based Model Checking

Armin Biere, Daniel Kröning

Handbook of Model Checking

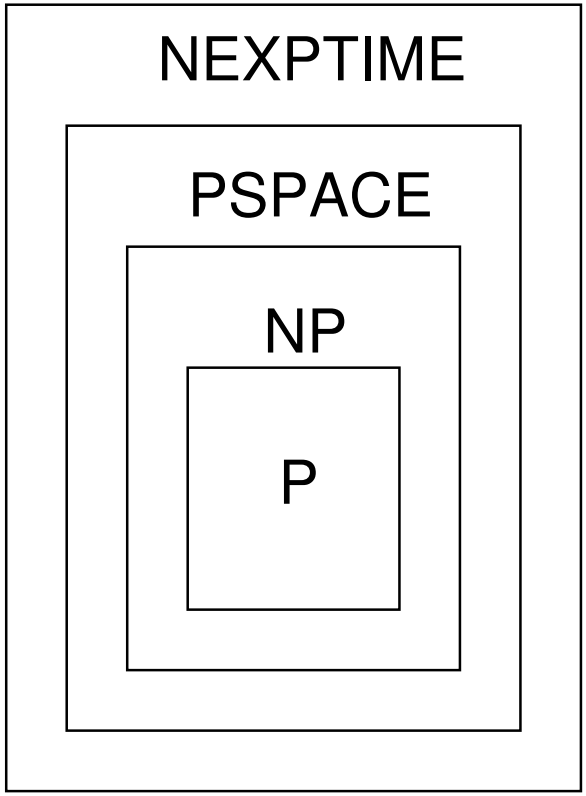
Edmund Clarke, Thomas Henzinger, Helmut Veith, *editors*

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 polynonmially **time**-bounded algorithms
  - bounds measured in terms of input (file) size
- NP
  - same as P but with non-determininistic 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

NEXPTIME Completeness of Bit-Vectors

joined work with Gergely Kovásznai and Andreas Fröhlich

- 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 quantiers)
  - my student Andreas has now an (yet unpublished) direct proof
- why are bit-vectors NEXPTIME complete?

$$x, y : \mathsf{bool}[1000000]$$

$$y \neq x \;\wedge\; x + y = x \ll 1$$

```
(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))
```

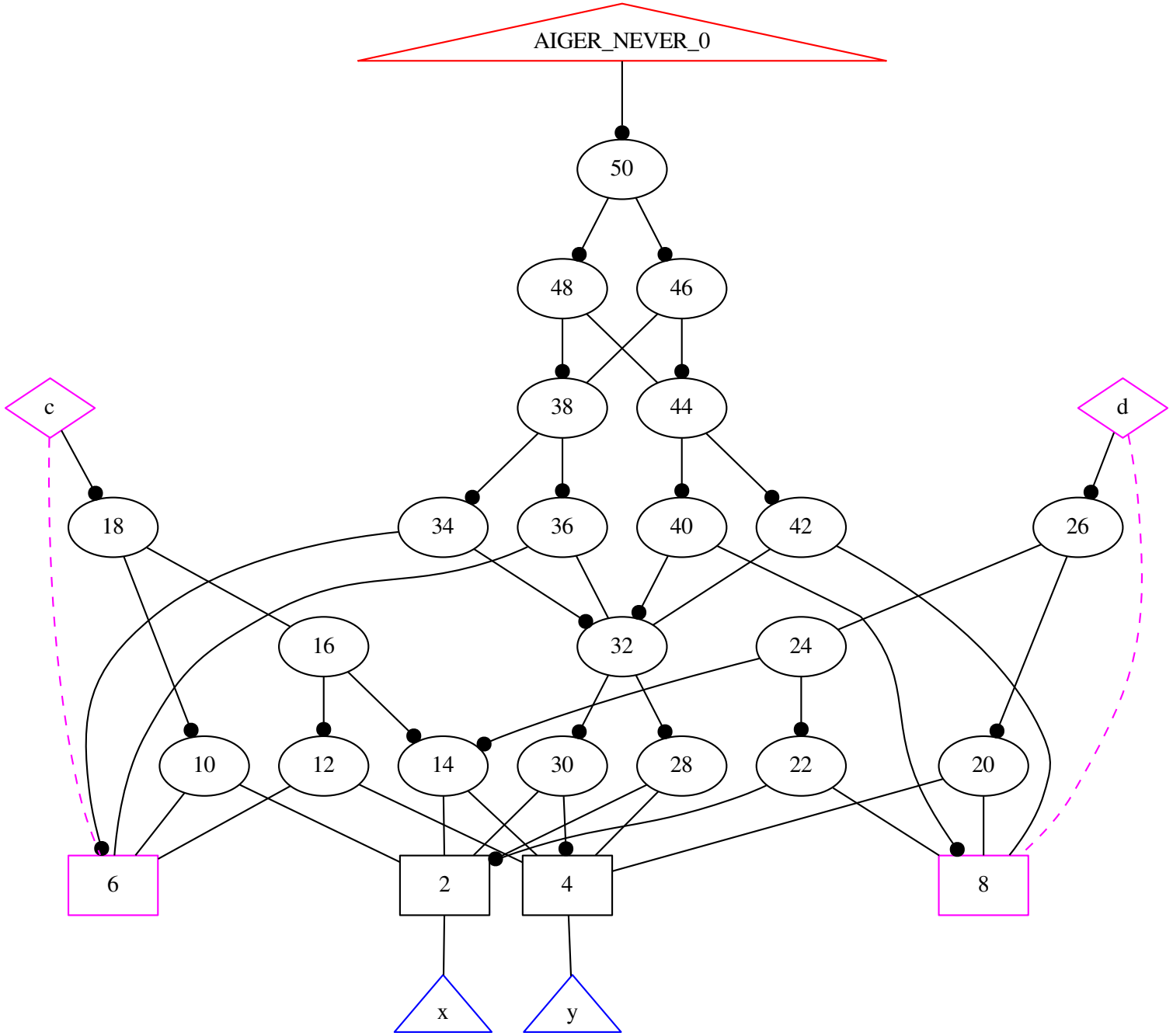
Bit-Wise Operators and Shifting Neighbouring Bits Only

- NP complete: QF_BV_{bw}
  - **relate same bits**: equality and all bit-wise operators
  - similar to well-known Ackermann reduction
- PSPACE complete: $\text{QF\_BV}_{bw,\ll 1}$
  - 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 accross otherwise unsolvable benchmarks from industry!

Commutativity of Bit-Vector Addition in SMV

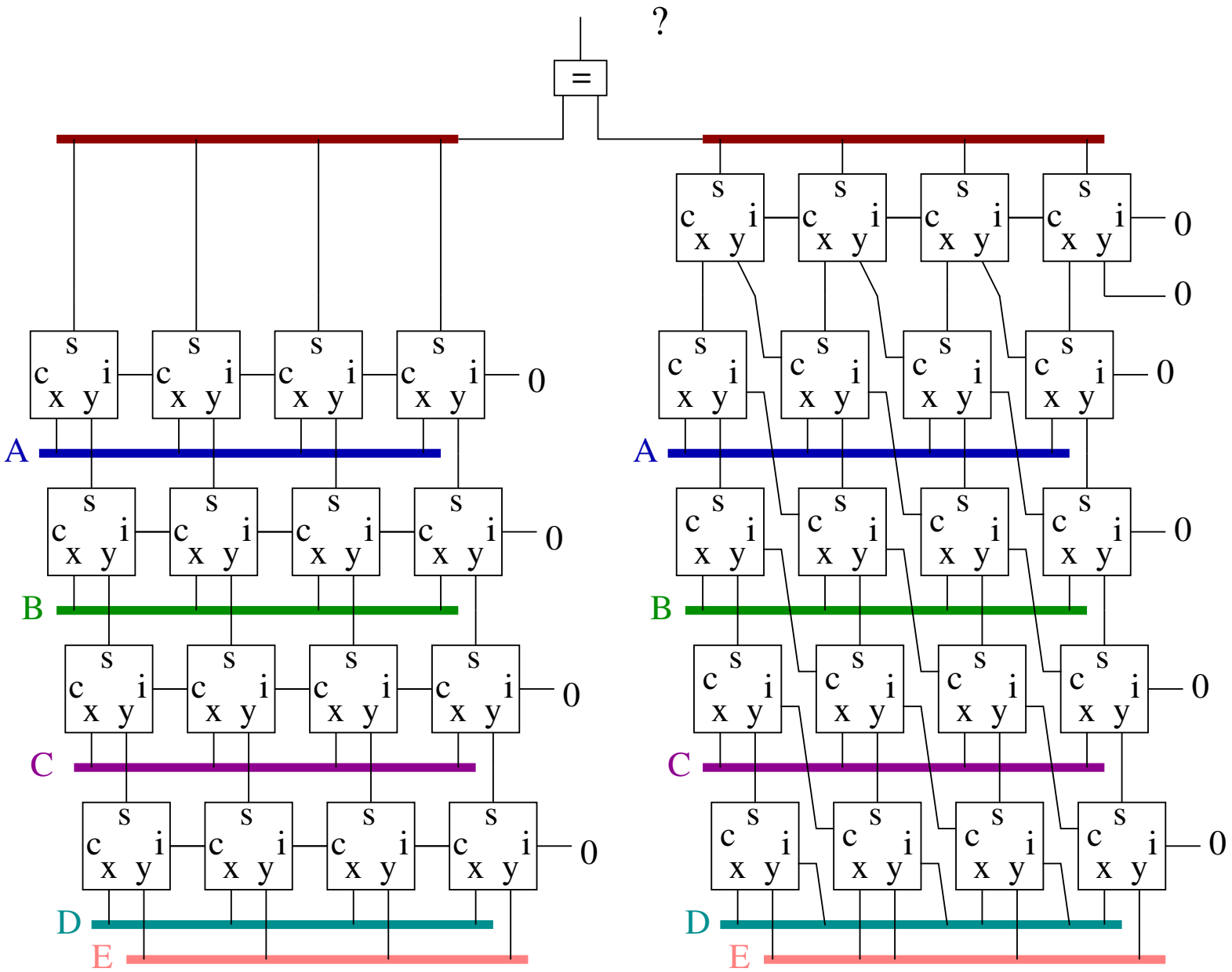
```
MODULE main
VAR
  c : boolean;                          -- carry 'bvadd x y'
  d : boolean;                          -- carry 'bvadd y x'
  x : boolean;                          -- x0, x1, ...
  y : boolean;                          -- y0, y1, ...
ASSIGN
  init (c) := FALSE;
  init (d) := FALSE;
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



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)
```

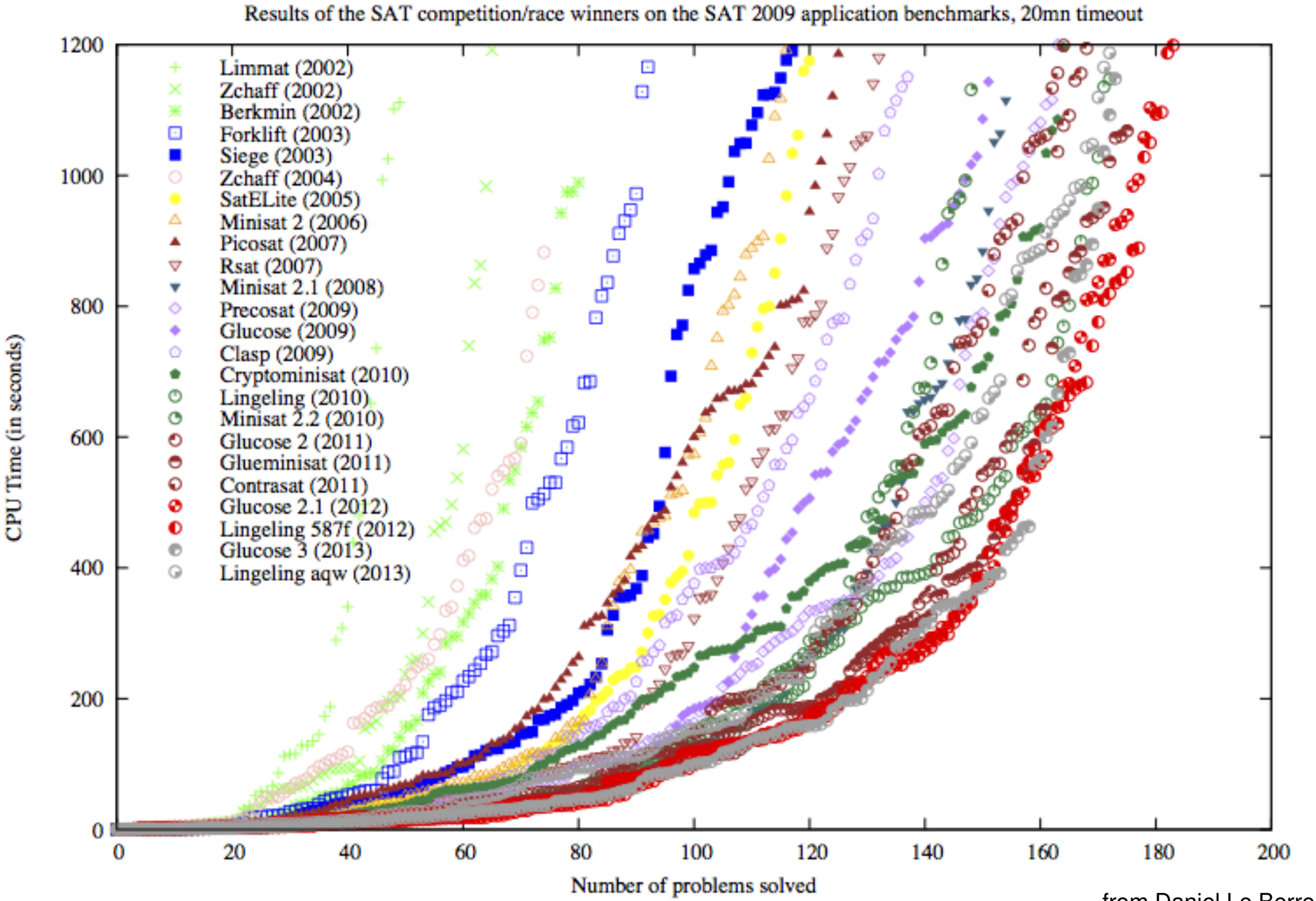
bits	1 core Glucose	1 core Lingeling	12 core cube-and-conquer March|iLingeling	12 core Treengeling
01	0.00	0.00	0.00	0.01
02	0.00	0.00	0.00	0.01
03	0.00	0.00	0.00	0.01
04	0.00	0.00	0.02	0.03
05	0.00	0.01	0.05	0.13
06	0.02	0.03	0.36	0.31
07	0.14	0.27	0.63	0.72
08	1.18	1.98	1.38	2.47
09	7.85	10.98	2.63	4.65
10	37.16	41.49	5.02	10.86
11	147.62	214.98	15.72	21.96
12	833.62	649.49	56.57	61.48
13	--	--	238.10	263.44

limit of 900 seconds wall clock time

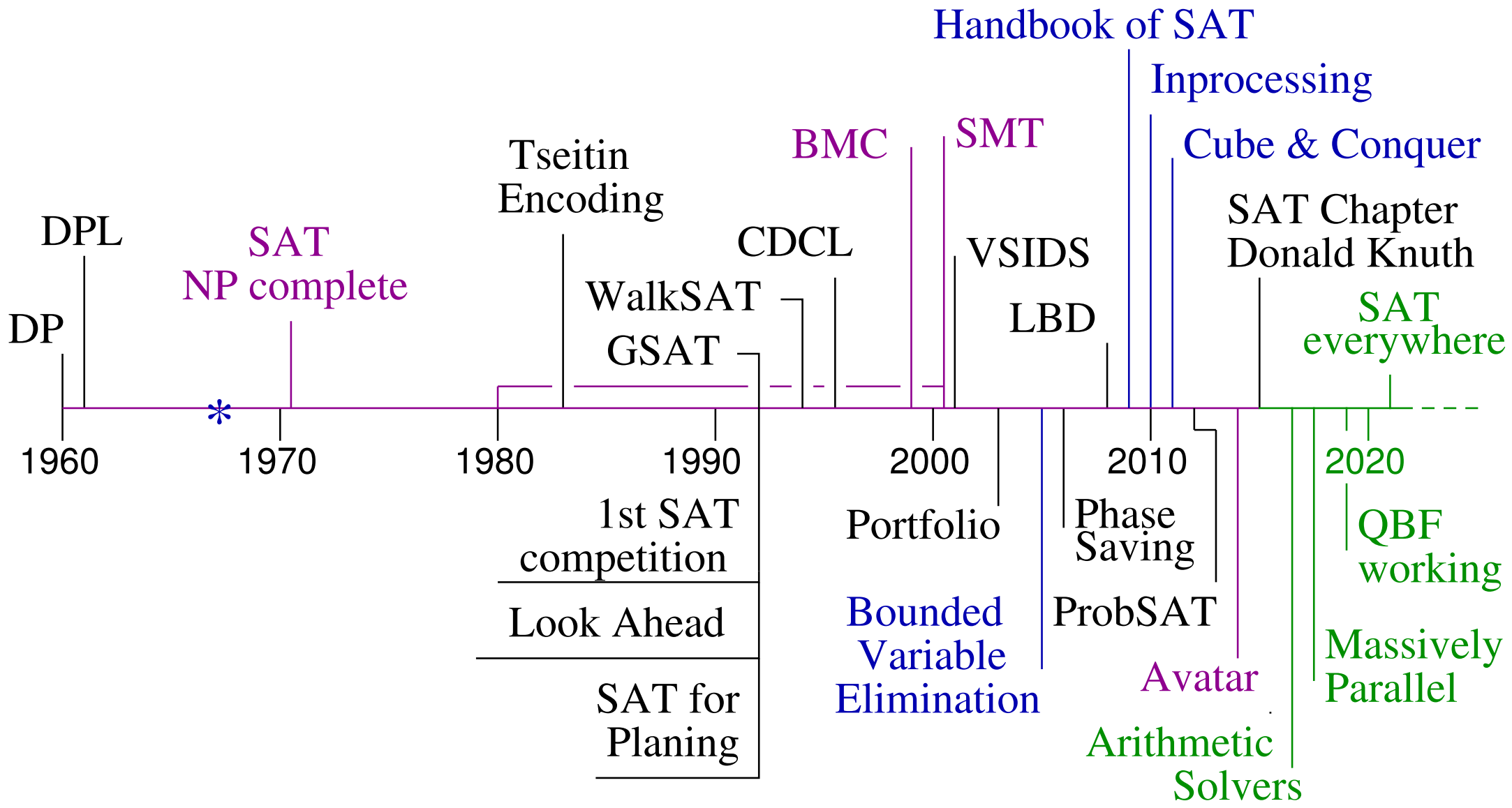
Challenge Arithmetic Circuit Equivalence Checking

- 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


SAT-Race 2010 Award
"Plingeling"
by Armin Biere
is awarded the title of
Best Parallel Solver
SAT-Race 2010 Award
"Lingeling"
by Armin Biere
is awarded the title of
Second Prize Winn
Tenth International Conference
On Theory and Applications of Satisfiability Testing
Competition 2007
Gold medal
Twelfth International Conference
On Theory and Applications of Satisfiability Testing
Competition 2009
Silver medal
Eighteenth International Conference
on Theory and Applications of Satisfiability Testing
SAT-Race 2015
Main Track
Special Prize:
Most Innovative Solver
Written by:
Armin Biere
3rd Prize
Parallel Track
Best Parallel Solver
2010
SAT-Race
2010
SAT
COMPE
TITION
2013
Winner



from Daniel Le Berre


Handbook of SAT
Inprocessing
Cube & Conquer
BMC
SMT
Tseitin
Encoding
SAT Chapter
Donald Knuth
DPL
SAT
NP complete
CDCL
VSIDS
WalkSAT
LBD
SAT
everywhere
DP
GSAT
1960
1970
1980
1990
2000
2010
2020
1st SAT
competition
Portfolio
Phase
Saving
QBF
working
Look Ahead
Bounded
Variable
Elimination
ProbSAT
Massively
Parallel
SAT for
Planing
Avatar
Arithmetic
Solvers

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 unwinding 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

Edmund M. Clarke, FORE 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.

ISBN 978-1-58603-929-5
ISSN 0922-6389

185

HANDBOOK
of satisfiability

Editors:
Armin Biere
Marijn Heule
Hans van Maaren
Toby Walsh

IOS Press

Frontiers in Artificial Intelligence and Applications
HANDBOOK
of satisfiability
Editors:
Armin Biere
Marijn Heule
Hans van Maaren
Toby Walsh
IOS Press

Part I. Theory and Algorithms

Part II. Applications and Extensions

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.

Biere
Bryant
Buss
Eén
Gent
Heule
Hoos
Janson
Jeavons
Kroening
Kullmann
Lauria
Pegden
Shortz
Sinz
Sörensson
Wermuth
Williams
Internet
MPR
Internet

* * *

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