Lessons between Computer Algebra and Verification/Satisfiability Checking

James Davenport[1]
University of Bath
J.H.Davenport@bath.ac.uk

20 June 2018

[1] Thanks to EU H2020-FETOPEN-2016-2017-CSA project $\mathcal{SC}^2$ (712689)

But first, a word from our sponsors

EU Coordinating and Support Action 712689
Satisfiability Checking and Symbolic Computation
http://www.sc-square.org/CSA/welcome.html

University of Bath	James Davenport; Russell Bradford
RWTH Aachen	Erika Ábrahám
Fondazione Bruno Kessler	Alberto Griggio; Alessandro Cimatt
Università degli Studi di Genova	Anna Bigatti
Maplesoft Europe Ltd	Jürgen Gerhard; Stephen Forrest
Université de Lorraine (LORIA)	Pascal Fontaine
Coventry University	Matthew England
University of Oxford	Daniel Kroening; Martin Brain
Universität Kassel	Werner Seiler; John Abbott
Max Planck Institut für Informatik	Thomas Sturm
Universität Linz	Tudur Jebelean; Bruno Buchberger
	Wolfgang Windsteiger; Roxana-Maria Holom

A Personal Reflection

Q Are you a happy computer science professor?

JHD Yes: several times a week I put my life in the hands of my ex-students, and I am happy with this!

Q *How* does this happen?

JHD Several work for a local software house, writing railway signalling and air traffic control software

Q When mine write code, it has bugs!

JHD Same, but they don't *deliver* bugs

Q How come?

JHD Program *verification*, based on *satisfiability*.

JHD For example National Air Traffic System has 1,000,000 crash-free hours

\+ Métro ligne 14 (driverless) software delivered in 1999: no bug reports

History: Computer Algebra

- **1894** Bachmann [Bac94] invents O-notation.
- **1953** First MSc theses in Computer Algebra
- **1961** Slagle's AI thesis [Sla61] "integrates better than a freshman".
- **1966** First Computer Algebra Conference (SYMSAC)
- **1967** Moses' thesis [Mos67], beating [Sla61] algorithmically, moves computer algebra out of AI
- **1974** Knuth [Knu74] popularises O etc. in computer science
- **today** Annual ISSAC conferences, dominated by complexity results
- **And** ACA and other conferences.

History: SAT Solving

- 1971 Cook [Coo71] shows that 3-SAT is NP-complete.
- 1988 Exponential lower bounds for resolution (DPLL) solvers
- 1993 Modern SMT (Satisfiability Modulo Theories) starts [AG93]
- ∼1995 CDCL (Conflict Driven Clause Learning) introduced
- 1996 First SAT conference
- 2001 "Two watched literals" invented [MMZ⁺01]

  "just" a programming hack, but powerful
- 2003 First SMT[2] workshop
- today Annual SAT conferences and SMT workshops, with contests a major feature

[2] Then known as PDPAR.

Reminder of SAT

Problem (SAT)

Given a Boolean formula (in CNF) ($l_{ij} \in \{x_k, \overline{x_k}\} : 1 \leq k \leq m$)

$$(l_{11} \vee l_{12} \vee \cdots) \wedge (l_{21} \vee l_2 \vee \cdots) \wedge \cdots \wedge (l_{n,1} \vee l_{n,2} \vee \cdots) \qquad (1)$$

find values of $x_k \in \{T, F\}$ *to make (1) true, or return UNSAT*

These days, contests ask for an "UNSAT core", i.e. a (locally) minimal unsatisfiable equivalent.
NB: the global minimum is impracticable [CGS11, §3.1].
SAT examples with $m, n > 10^6$ occur routinely in hardware verification, and are routinely solved.

Statement of SMT

Problem (SMT)

Given a Boolean formula (possibly in CNF) ($l_{ij} \in T$ a theory)

$$(l_{11} \vee l_{12} \vee \cdots) \wedge (l_{21} \vee l_2 \vee \cdots) \wedge \cdots \wedge (l_{n,1} \vee l_{n,2} \vee \cdots) \quad (2)$$

find values in the theory to make (2) true, or return UNSAT (possibly also an UNSAT core)

There are many possible theories: SMT-LIB `http://smtlib.cs.uiowa.edu/theories.shtml` lists seven, such as `Reals`, But there are over 50 "benchmark categories", such as `QF_NRA` (quantifier-free nonlinear real arithmetic[3]).

[3] SMT-speak uses "arithmetic" where this community would use "algebra".

Statement of SMT QF_NRA

Problem (SMT–QF_NRA)

Given a Boolean formula (possibly in CNF) ($l_{ij} := f_{ij}\sigma_{ij}0$, $f_{ij} \in \mathbf{Q}[x_1, \ldots, x_k]$, $\sigma_{ij} \in \{=, \neq, <, >, \leq, \geq\}$)

$$(l_{11} \vee l_{12} \vee \cdots) \wedge (l_{21} \vee l_2 \vee \cdots) \wedge \cdots \wedge (l_{n,1} \vee l_{n,2} \vee \cdots) \tag{3}$$

find values for $x_1, \ldots, x_k \in \overline{\mathbf{Q}}$ to make (3) true, or return UNSAT.

Fortunately $\overline{\mathbf{Q}}$ is sufficient, but $\mathbf{Q}$ is not ($x^2 - 2 = 0$).
We could ask for an UNSAT core here as well, but one tends to need an "UNSAT core+proof", a concept that's still not well-defined.

Translation into Computer Algebra

Problem

Solve (or prove insoluble)

$$\exists x_1 \cdots \exists x_k \Phi(f_i \sigma_i 0) : \tag{4}$$

$f_i \in \mathbf{Q}[x_1, \ldots, x_k]$, $\sigma_i \in \{=, \neq, <, >, \leq, \geq\}$, Φ *a Boolean combination.*

This is more specific than usual quantifier elimination/Cylindrical Algebraic Decomposition, as all variables are quantified with the same quantifier, and hence the doubly-exponential bounds [BD07, DH88] don't apply.

Goals

Computer Algebra Describe the space of all solutions

Satisfiability Find one solution, or UNSAT

#SAT is the problem of counting all solutions, and this is known to be much harder in practice

MAXSAT is the problem of finding the "best" solution, also much harder in practice

worse Cylindrical Algebraic Decomposition will find all the geometry of *all* the polynomials, so will struggle with

$$(x < -1) \wedge (x > 1) \wedge \Phi(\text{big polynomials}) \tag{5}$$

whereas any decent SMT will say UNSAT immediately

Strategies

Computer Algebra:

- [Col75] Look at the f_i first
- [McC99] If Φ is $f_1 = 0 \wedge \Phi'$ process f_1 and x_k specially, then look at the $\mathrm{res}_{x_k}(f_1, f_i)$ first
- [BDE+16] Handle $(f_1 = 0 \wedge \Phi') \vee \Phi''$ as well
- [McC01] handle multiple $f_i = 0$
- [ED16] improve on this, *provided* f_i etc. are primitive

Satisfiability Modulo Theories:

- all look at the logic first
- [JdM12] Use CAD-inspired techniques to construct a refutation

[Bro13, Bro15] Feed these ideas back into Computer Algebra

- Idea An imprimitive polynomial $f(x) = 0$ is a disjunction $\mathrm{cont}(f) = 0 \vee \mathrm{pp}(f) = 0$

Other theories

- **QF_LRA** Can make use of linear programming, another field where practice is far better than theory
- **Linearise** Work in [CGI+17, Irf18] linearises multiplication, and even transcendental functions
- **Used** to verify aircraft wheel systems: note the **R** refers to the real world
- **QF_FLOAT** to verify programs *manipulating* floating-point numbers

  Tends to be done by converting into bit-vectors: very expensive.

Meta-Strategies

Computer Algebra:

Complexity	What's the worst case?
Also	Can we prove lower bounds (e.g. [BD07])
Algorithms	probably want four examples to show we're faster than some other guy
But	we run these tests, so probably comparing my experimental with his old production

Satisfiability Modulo Theories:

Benchmarks	are everything, the more examples (preferably thousands) the better
Contests	with independent jury/setting implement these.
Therefore	doing well on easy cases also matters
And	heuristics matter

But how do you present results of benchmarks on thousands of examples? [BDG17]

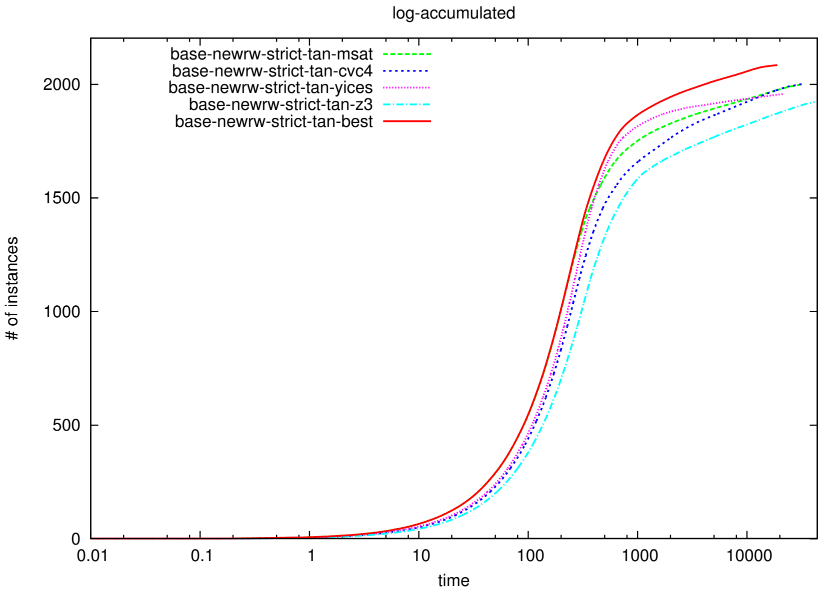
“Cactus” or “Survival” plots

The methodology for producing these, given a large benchmark set of problems, is as follows.

1. For each method separately
   1. Solve each problem p_i, noting the time t_i (up to some threshold T).
   2. Sort the t_i into increasing order (discarding the time-out ones).
   3. Plot the points $(t_1, 1)$, $(t_1 + t_2, 2)$ etc., and in general $(\sum_{i=1}^{k} t_i, k)$.
2. Place all the plots on the same axes, optionally using a logarithmic scale for time.
3. Optionally add “virtual best solver”

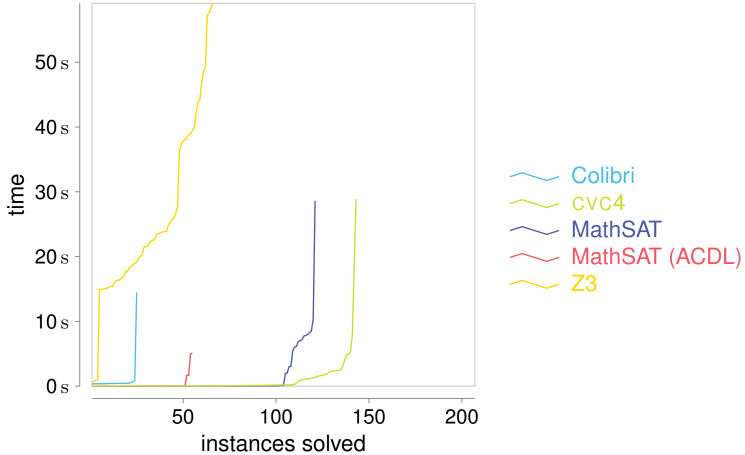
N.B. There is therefore no guarantee that the same problems were used to produce time results from different solvers.

A plot


log-accumulated
base-newrw-strict-tan-msat
base-newrw-strict-tan-cvc4
base-newrw-strict-tan-yices
base-newrw-strict-tan-z3
base-newrw-strict-tan-best
# of instances
2000
1500
1000
500
0
0.01
0.1
1
10
100
1000
10000
time

Cactus plot

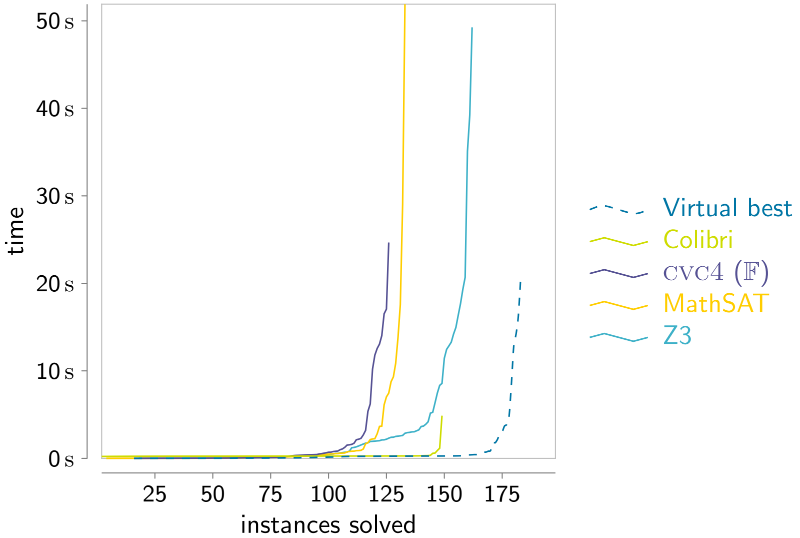
Heizmann



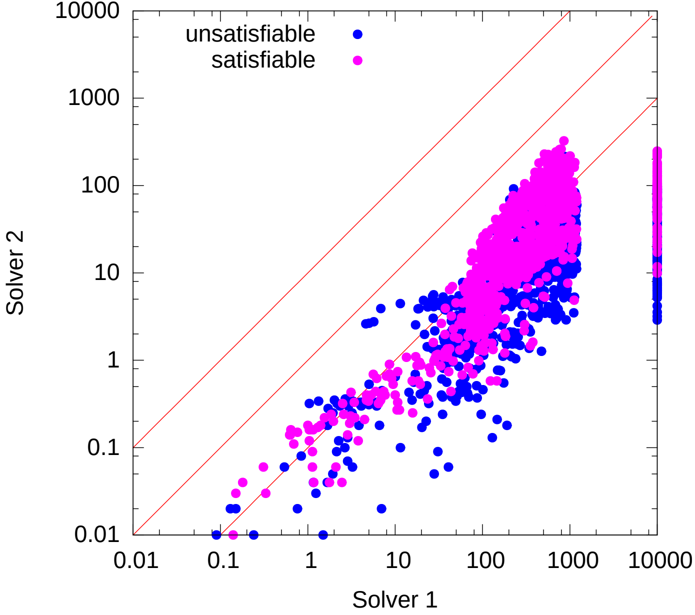
23 CVC4 IEEE-754 implementation

Plot with Virtual Best Solver

Figure: Schanda Cactus Plot



How do two solvers compare?



Conclusions

"People like theorems because they are neat, but people use software because it solves problems"

1. Computer Algebra does not make enough (?any) use of SAT solvers
2. The same is probably true of linear programming
3. Can computer algebra help with `QF_FLOAT`?
4. "If we could use reals rather than booleans for signalling, we could get 30% more trains on our tracks" SC^2 colleague

The challenge isn't writing the software — it's proving it correct, and I want to stay a happy CS professor!

??? Is ACA the right place to host benchmarking for computer algebra?

But Where do we get thousands of problems from?

By being industrially relevant, which requires demonstrating on benchmarks of thousands of problems, which . . .

SMT-Lib has similar challenges: Let's start! [WBD12]

Bibliography

A. Armando and E. Giunchiglia.
Embedding Complex Decision Procedures inside an Interactive Theorem Prover.
Annals of Mathematics and Artificial Intelligence, 8:475–502, 1993.

P. Bachmann.
Die analytische Zahlentheorie.
Teubner, 1894.

C.W. Brown and J.H. Davenport.
The Complexity of Quantifier Elimination and Cylindrical Algebraic Decomposition.
In C.W. Brown, editor, *Proceedings ISSAC 2007*, pages 54–60, 2007.

R.J. Bradford, J.H. Davenport, M. England, S. McCallum, and D.J. Wilson.
Truth table invariant cylindrical algebraic decomposition.
J. Symbolic Computation, 76:1–35, 2016.

M.N. Brain, J.H. Davenport, and A. Griggio.
Benchmarking Solvers, SAT-style.
SC-Square 2017 Satisfiability Checking and Symbolic Computation CEUR Workshop 1974, 2017.

C.W. Brown.
Constructing a single open cell in a cylindrical algebraic decomposition.
In *Proceedings ISSAC 2013*, pages 133–140, 2013.

C.W. Brown.
Open Non-uniform Cylindrical Algebraic Decompositions.
In *Proceedings ISSAC 2015*, pages 85–92, 2015.

A. Cimatti, A. Griggio, A. Irfan, M. Roveri, and R. Sebastiani.
Satisfiability Modulo transcendental functions via incremental linearization.
Proc. SMT 2017 CEUR Workshop Proceedings, 1889, 2017.

A. Cimatti, A. Griggio, and R. Sebastiani.
Computing small unsatisfiable cores in satisfiability modulo theories.
Journal of Artificial Intelligence Research, 40:701–728, 2011.

Bibliography

G.E. Collins.
Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition.
In *Proceedings 2nd. GI Conference Automata Theory & Formal Languages*, pages 134–183, 1975.

S.A. Cook.
The Complexity of Theorem-Proving Procedures.
In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

J.H. Davenport and J. Heintz.
Real Quantifier Elimination is Doubly Exponential.
J. Symbolic Comp., 5:29–35, 1988.

Bibliography

M. England and J.H. Davenport.
The Complexity of Cylindrical Algebraic Decomposition with Respect to Polynomial Degree.
In V.P. Gerdt, W. Koepf, W.M. Seiler, and E.V. Vorozhtsov, editors, *Proceedings CASC 2016*, Springer Lecture Notes in Computer Science 9890, pages 172–192. Springer, 2016.

A. Irfan.
Incremental Linearization for Satisfiability and Verification Modulo Nonlinear Arithmetic and Transcendental Functions.
PhD thesis, Università degli Studi di Trento, 2018.

D. Jovanović and L. de Moura.
Solving Non-Linear Arithmetic.
In *Proceedings IJCAR 2012*, pages 339–354, 2012.

D.E. Knuth.
Big Omicron and big Omega and big Theta.
ACM SIGACT News 2, 8:18–24, 1974.

S. McCallum.
On Projection in CAD-Based Quantifier Elimination with Equational Constraints.
In S. Dooley, editor, *Proceedings ISSAC '99*, pages 145–149, 1999.

S. McCallum.
On Propagation of Equational Constraints in CAD-Based Quantifier Elimination.
In B. Mourrain, editor, *Proceedings ISSAC 2001*, pages 223–230, 2001.

M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik.
Chaff: Engineering an Efficient SAT Solver.
In *Proceedings 38th Design Automation Conference*, pages 530–535, 2001.

J. Moses.
Symbolic Integration.
PhD thesis, M.I.T. & Project MAC TR-47, 1967.

J. Slagle.
A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus.
PhD thesis, Harvard U., 1961.

Bibliography VIII

D.J. Wilson, R.J. Bradford, and J.H. Davenport.
A Repository for CAD Examples.
ACM Communications in Computer Algebra 3, 46:67–69, 2012.