SAT Solvers and Computer Algebra Systems: A Powerful Combination for Mathematics

Vijay Ganesh

Assistant Professor, University of Waterloo, Canada

(jointly with Curtis Bright and Ilias Kotsireas)

Thursday June 21, 2018

ACA, Santiago De Compostela, Spain

SAT+CAS
SOME PERSPECTIVES

Brute-brute force has no hope. But clever, inspired brute force is the future. – *Dr. Doron Zeilberger, Rutgers University, 2015.*

The research areas of SMT [SAT Modulo Theories] solving and symbolic computation are quite disconnected. [...] More common projects would allow us to join forces and commonly develop improvements on both sides. – *Dr. Erika Abraham, RWTH Aachen University, 2015. (Invited talk, ISSAC 2015)*

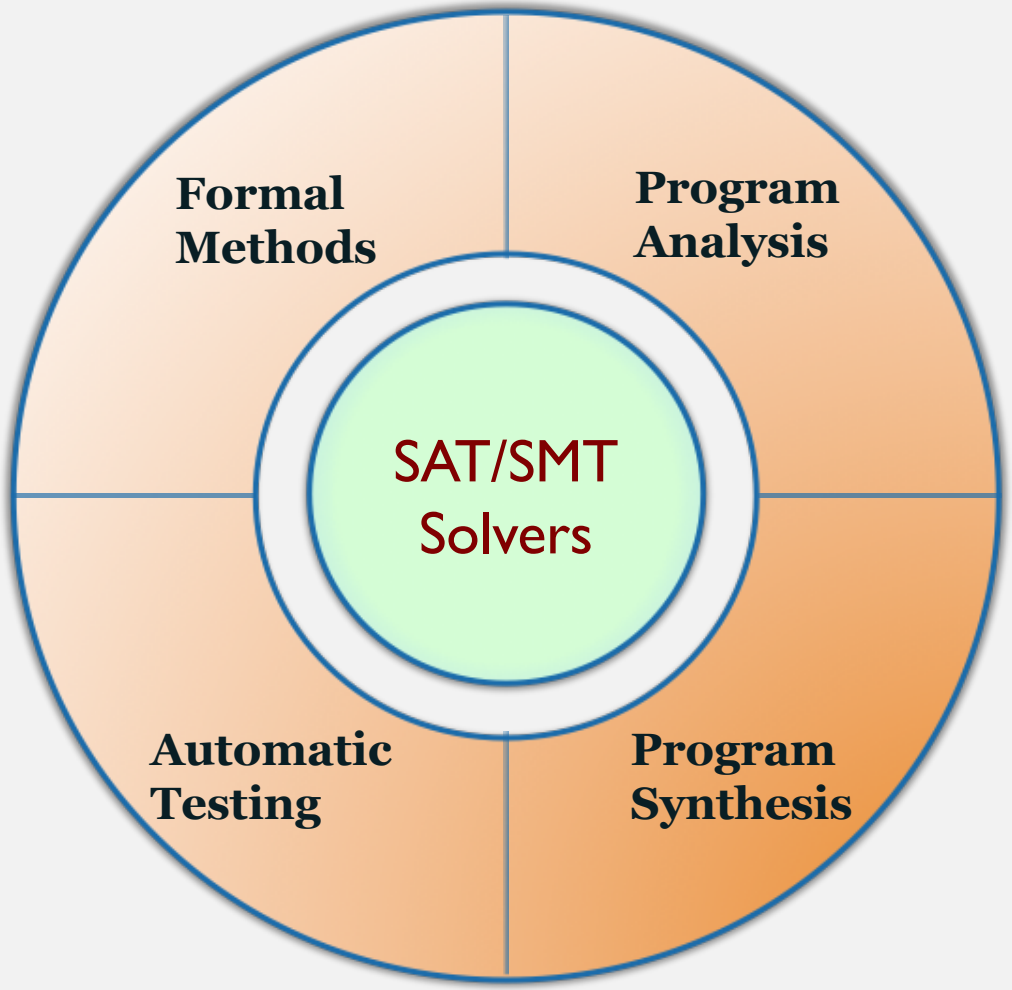
Independently, in 2014 we started looking into combinations of SAT and CAS aimed at constructing counterexamples or finite verification of math conjectures, publishing our first paper on the topic at the Conference on Automated Deduction (CADE) 2015.

PART I

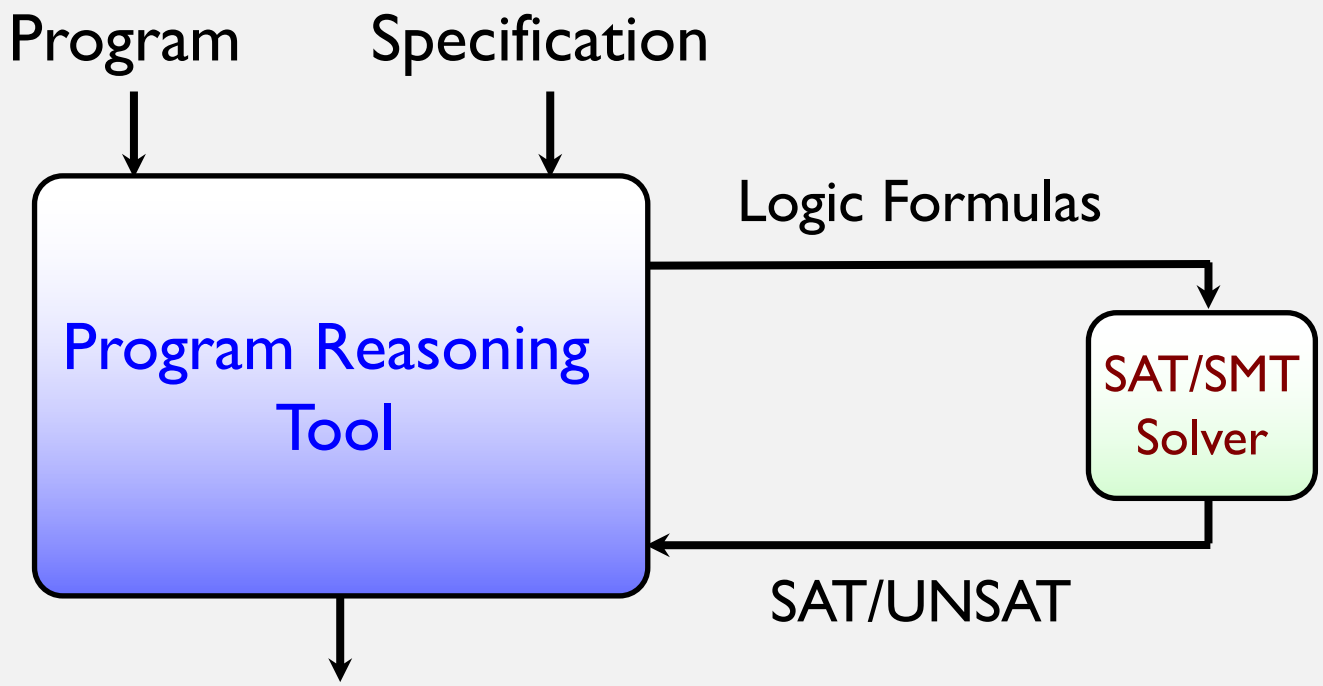
MOTIVATION

WHY SHOULD YOU CARE ABOUT SAT SOLVERS?

SOFTWARE ENGINEERING & SAT/SMT SOLVERS
AN INDISPENSABLE TACTIC FOR ANY STRATEGY

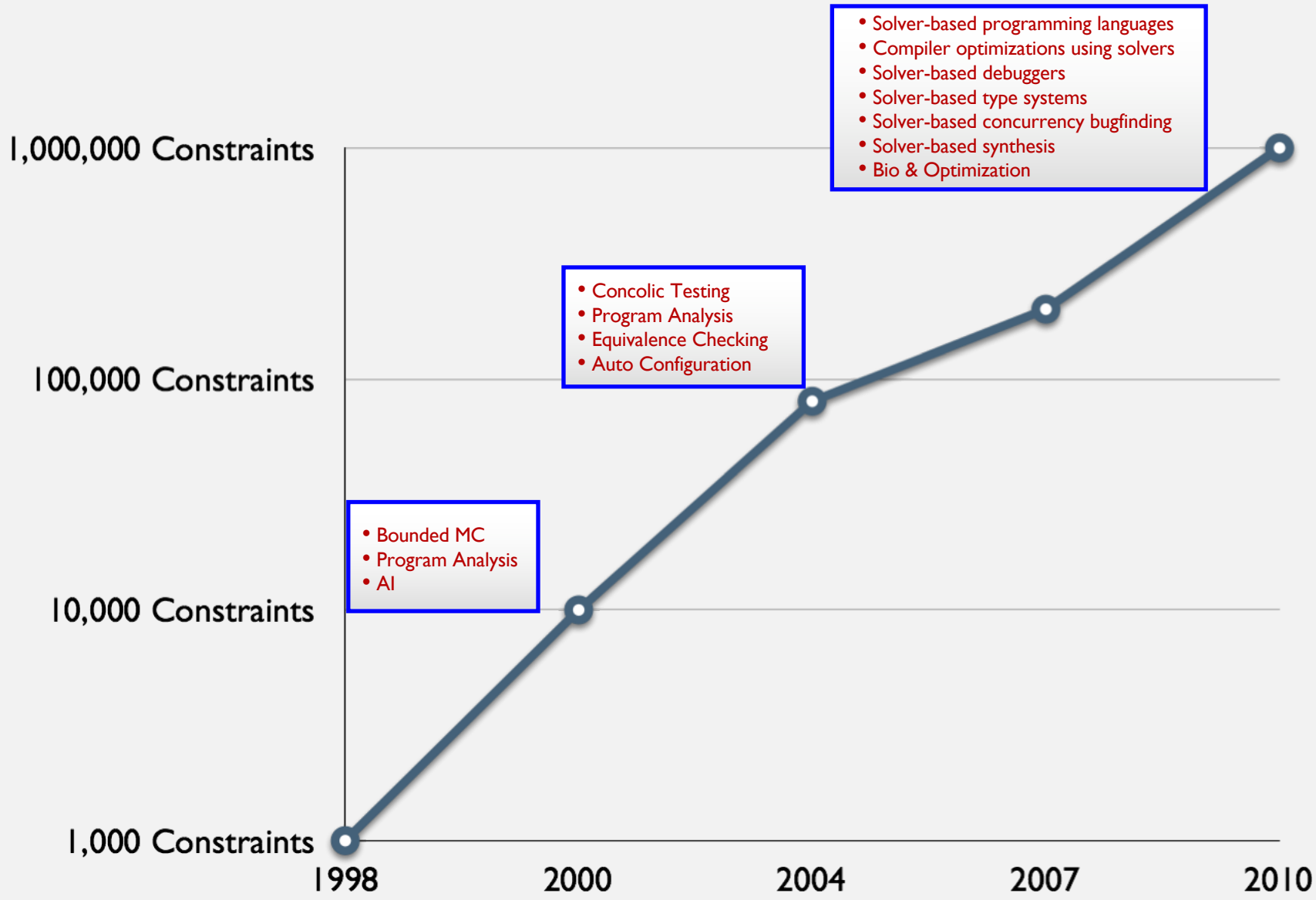


SOFTWARE ENGINEERING USING SOLVERS
ENGINEERING, USABILITY, NOVELTY

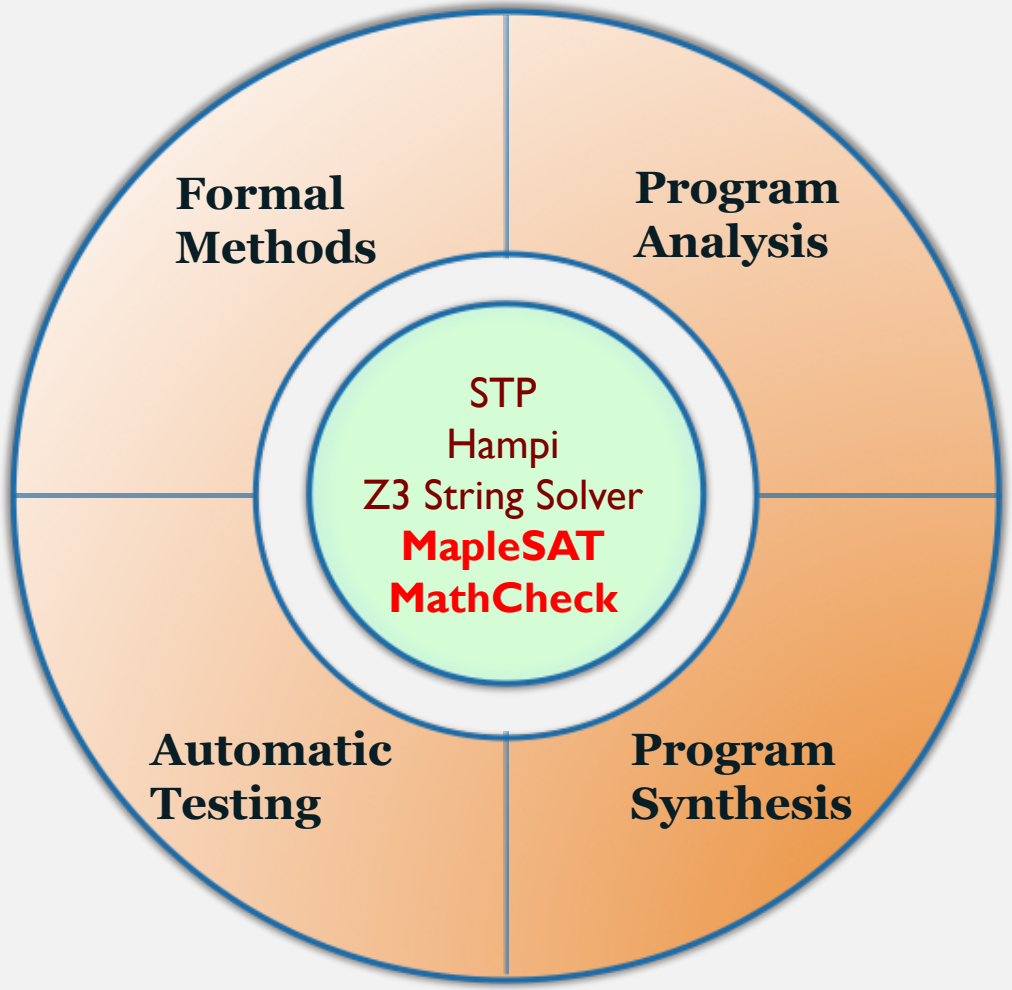


Program is correct?
or Generate Counterexamples (test cases)

SAT/SMT SOLVER RESEARCH STORY
A 1000X+ IMPROVEMENT



SOFTWARE ENGINEERING & SAT/SMT SOLVERS
AN INDISPENSABLE TACTIC FOR ANY STRATEGY



PART II

SAT SOLVER BASICS

POWER OF CONFLICT-DRIVEN CLAUSE-LEARNING

The Boolean SATisfiability Problem
Standard Definitions

- A **literal** p is a Boolean variable x or its negation $\neg x$. A **clause** C is a disjunction of literals: $x_2 \vee \neg x_{41} \vee x_{15}$. A 3-**CNF** formula is a conjunction of m clauses over n variables
- An **assignment** is a mapping from variables to Boolean values **(True, False).** A **unit clause** C is a clause with a single unbound literal
- The Boolean **SAT problem**:
  - Decide whether Boolean formulas in CNF are satisfiable, i.e., find an assignment such that each input clause has a true literal (aka input formula is SAT) OR establish that input formula has no solution (aka input formula is UNSAT)
- A **SAT Solver** is a program that solves the SAT problem. All known SAT solvers have **worst-case exponential time complexity** in the number of variables of input formula
  - SAT solvers typically output a solution if input is SAT (and produce proofs if input is UNSAT)

DPLL SAT Solver Architecture
The Basic Backtracking SAT Solver

DPLL(Θ_{cnf}, assign) {

Propagate unit clauses;

if *"conflict"*: **return** FALSE;

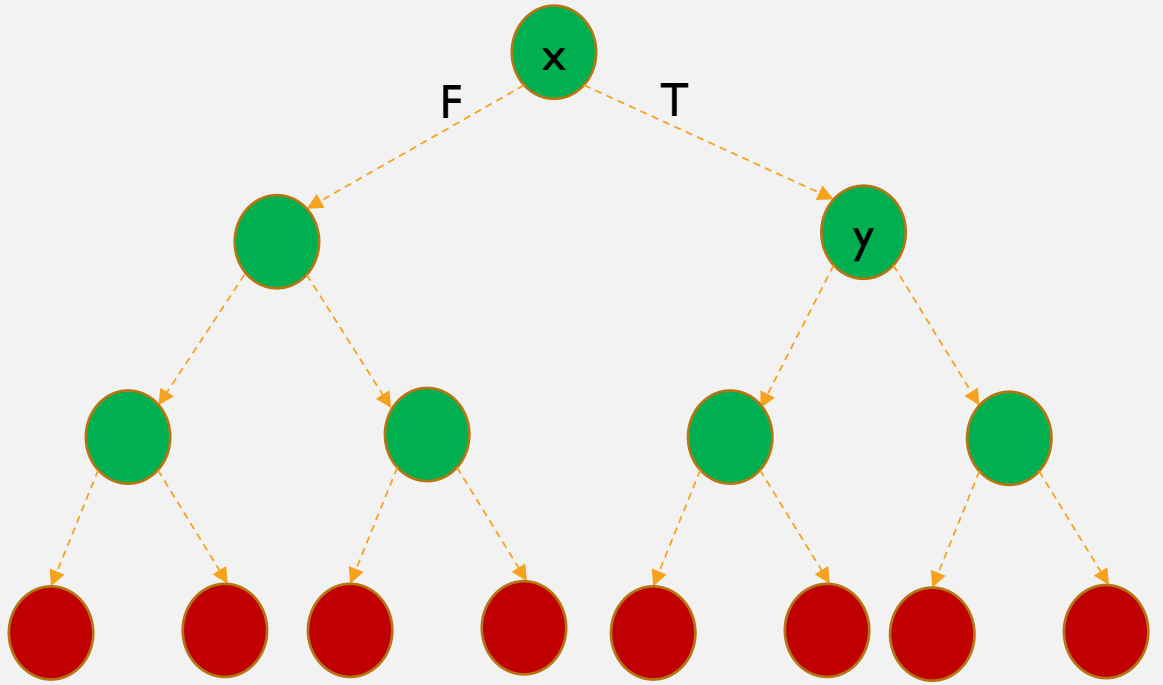
if *"complete assign"*: **return** TRUE;

"pick decision variable x";

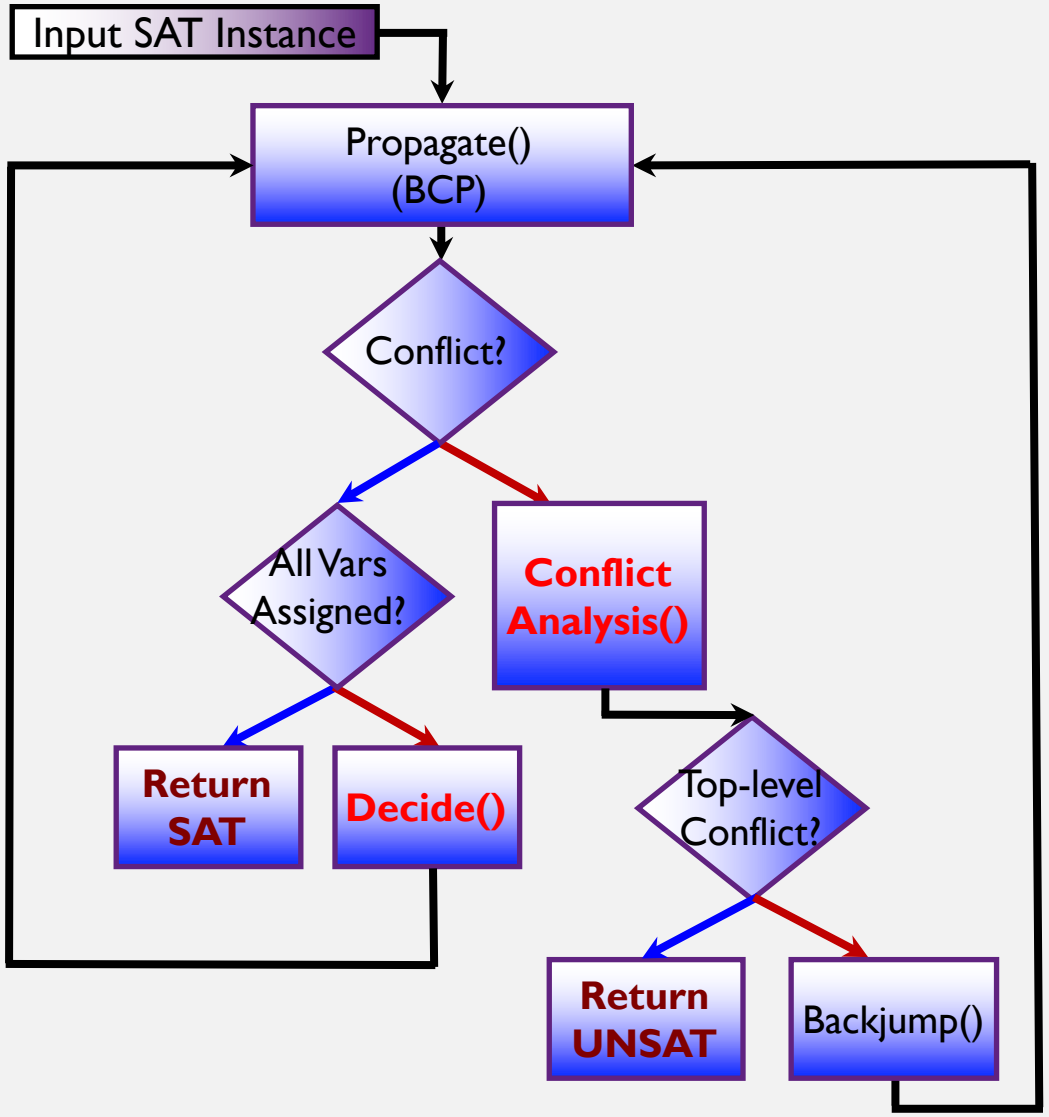
return

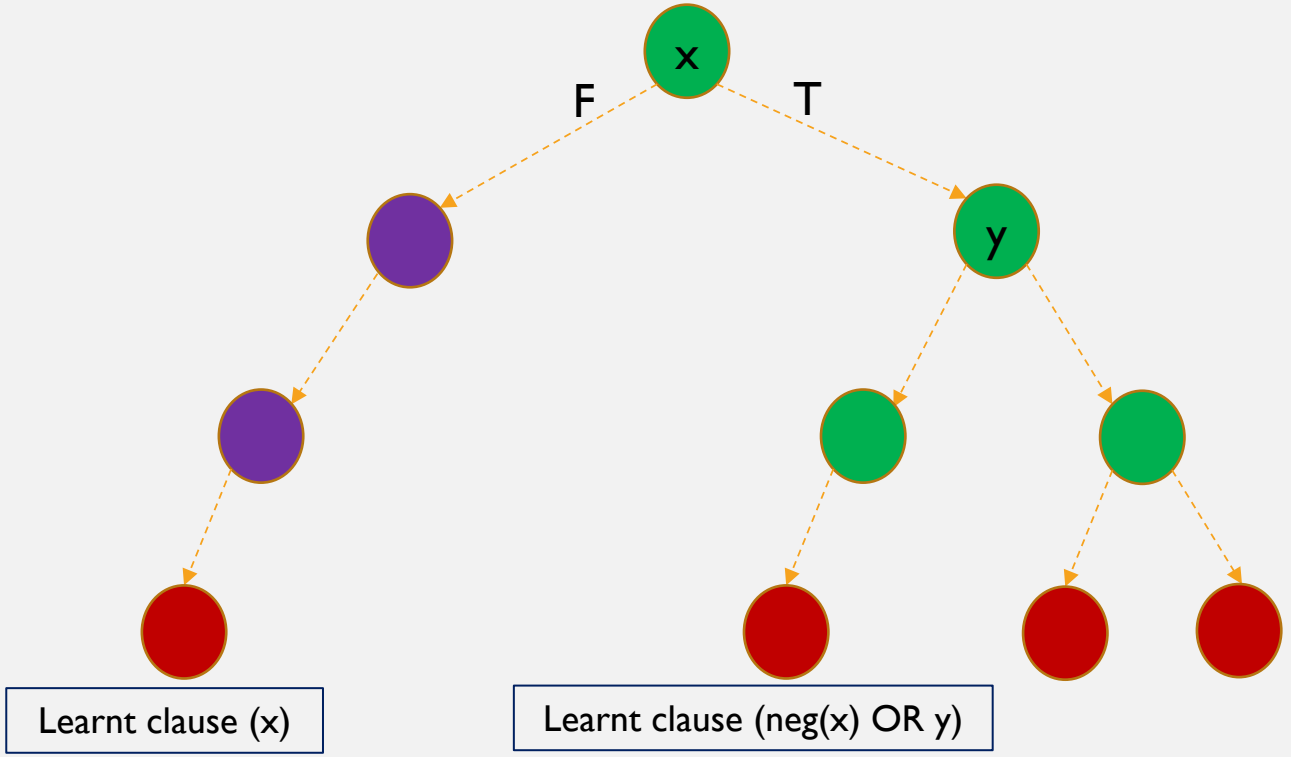
DPLL(Θ_{cnf} | $_{x=0}$, assign[x=0])
|| DPLL(Θ_{cnf} | $_{x=1}$, assign[x=1]);

}

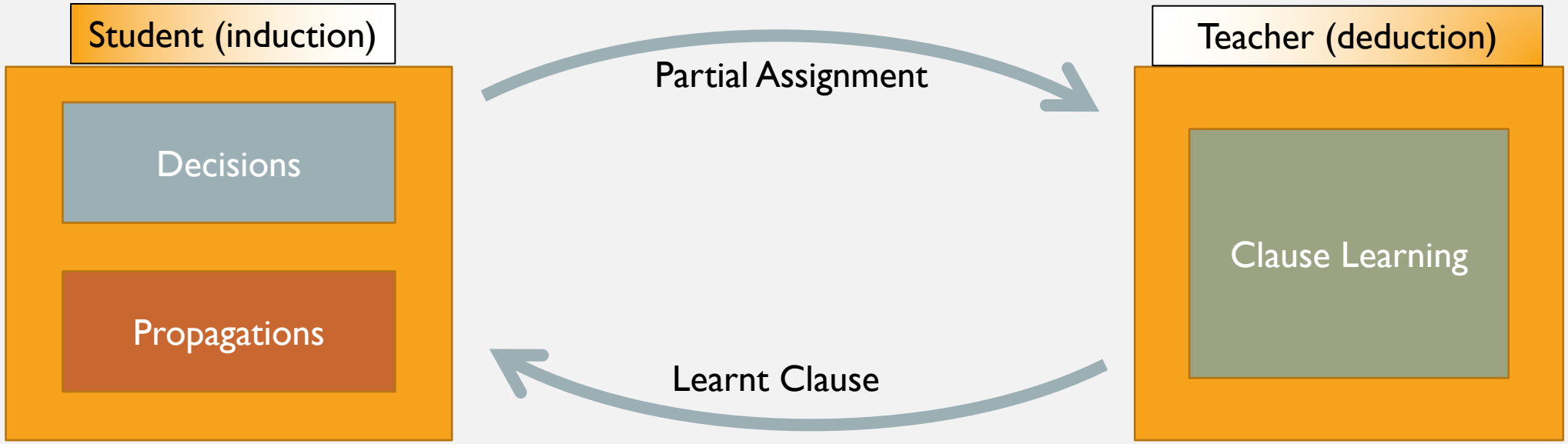


Modern CDCL SAT Solver Architecture
Key Steps: Conflict Driven Clause Learning and Smart Branching





AN ABSTRACTION OF A CDCL SAT SOLVER



1. This abstraction captures the most essential aspects of CDCL. Combines synthesis (induction) with verification (deduction)

2. There is similar class of algorithms in reinforcement learning

3. Enabled us to design a new class of branching heuristics

PART III

CAN WE DO BETTER?

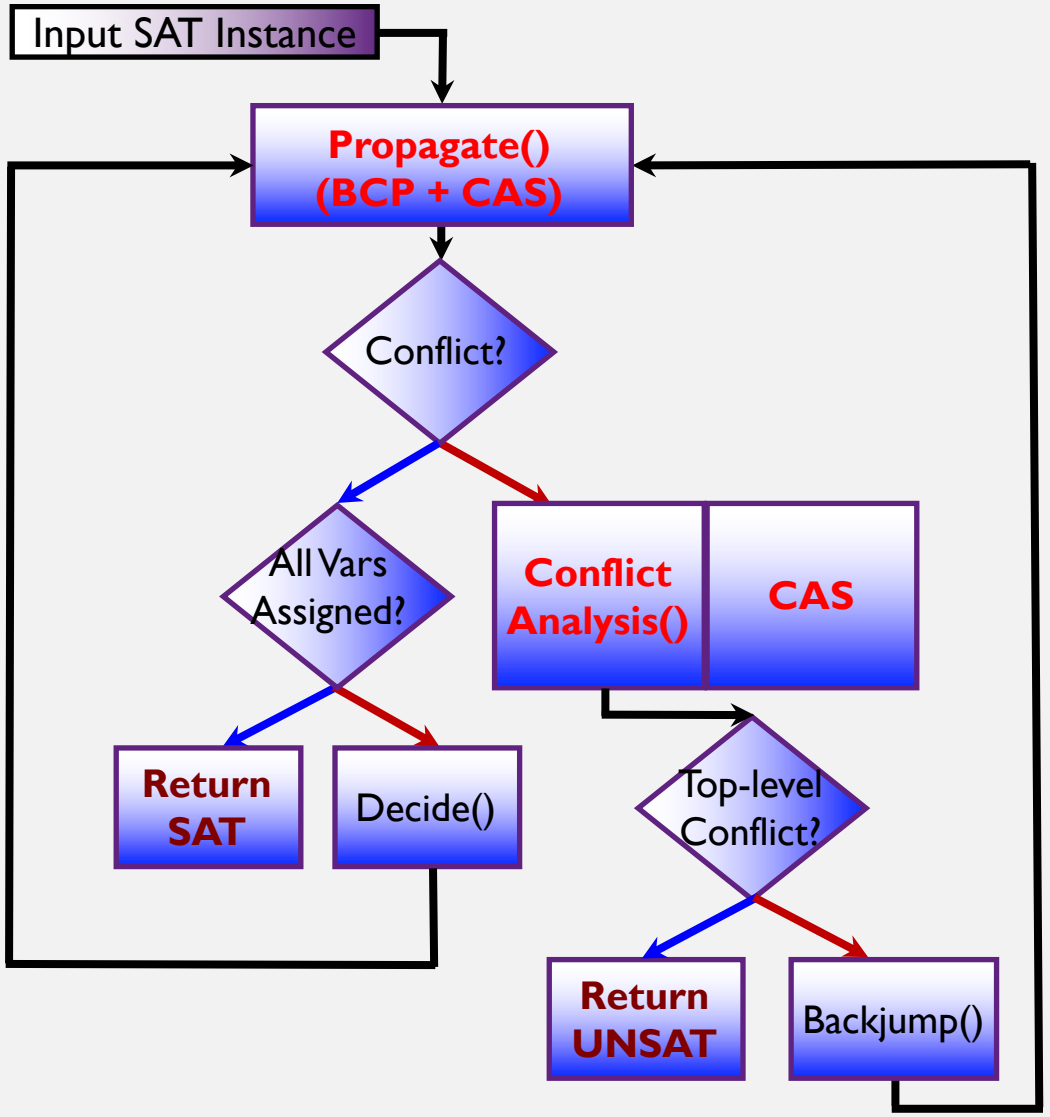
THE SAT+CAS PARADIGM

SAT+CAS PARADIGM
MOTIVATION AND RESEARCH QUESTION

	STRENGTHS	WEAKNESSES
SAT Solvers	Excellent at combinatorial search Flexible architecture, easily extensible	Input language (Boolean logic) is weak, i.e., predicates from rich fragments of mathematics are difficult/impossible to express
Computer Algebra Systems (CAS)	Deep repositories of mathematical knowledge Rich collection of mathematical algorithms for solving polynomials, linear algebra, transforms of many kinds, …	Lack good algorithms for general-purpose combinatorial search

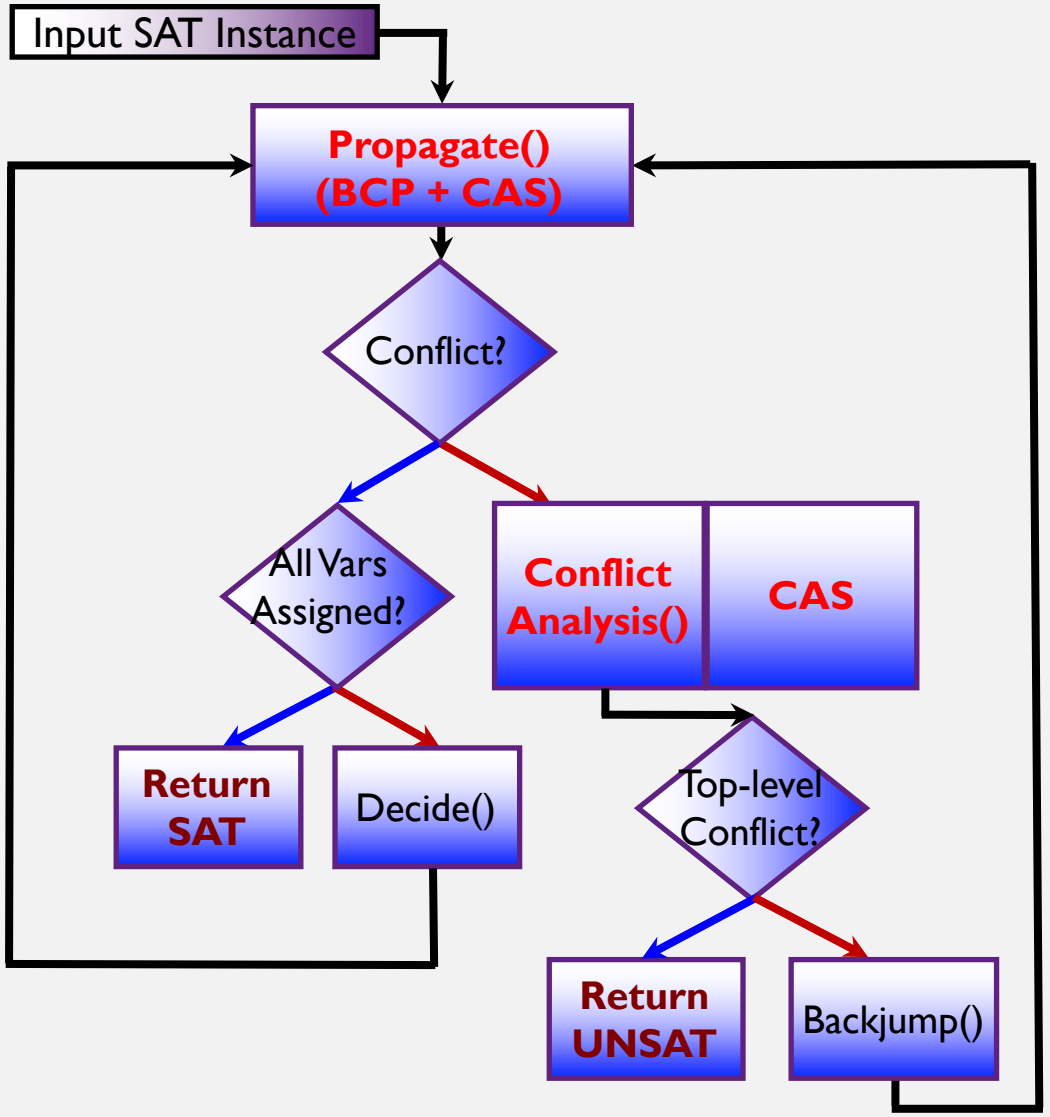
Research question: Can we somehow combine SAT and CAS to get the best of both worlds, aimed at constructing counterexamples or finitely verifying math conjectures?

SAT+CAS PARADIGM
CRUCIAL INSIGHTS



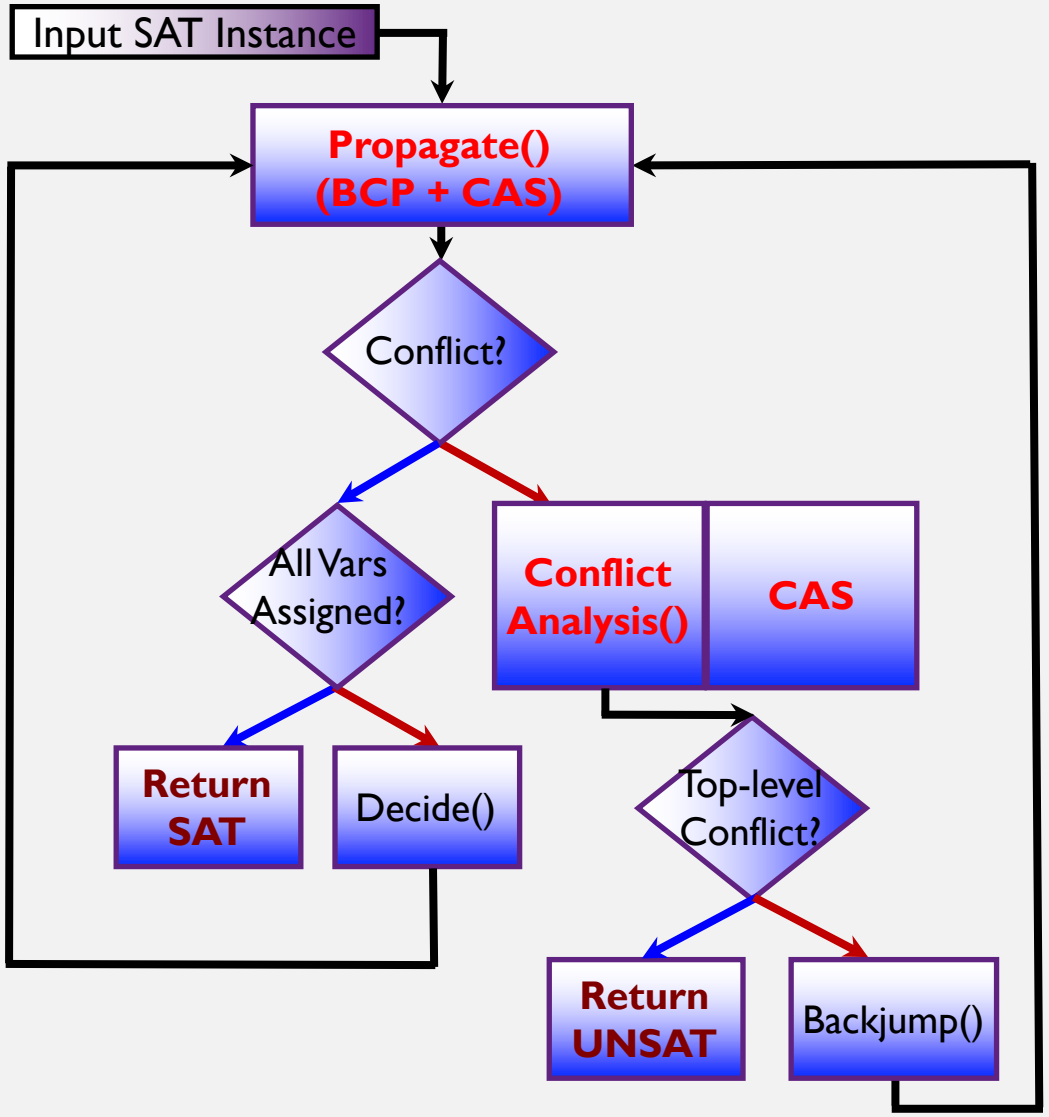
- Enhance/augment conflict analysis and BCP in SAT solvers with CAS (Inspired by DPLL(T))
- Student-Teacher model – SAT is the combinatorial student, and CAS is the domain-expert teacher
- Under what conditions do such combinations make sense?
  - Problem can be 'Booleanized' easily, to a certain extent
  - However, the problem also has rich mathematical structure which may be difficult or impossible to encode in Boolean logic
  - SAT solver explores the search space. The CAS system provides feedback lazily (in an on-demand fashion) in the form of conflict clauses

SAT+CAS PARADIGM
SOME DETAILS



- SAT solver reaches an inconclusive state, i.e., a partial assignment that is neither a conflict nor a complete satisfying assignment
- The CAS system using domain knowledge (in the form of lemmas and theorems) checks if the partial assignment can be extended in manner consistent with its domain knowledge
- If the partial assignment A is inconsistent with its domain knowledge, return a conflict clause to the SAT solver to block all extensions of A thus dramatically pruning the search
- Implementation details: The CAS is called by the solver via a Callback mechanism in the inner loop of Programmatic MapleSAT

SAT+CAS PARADIGM
MATHCHECK RESULTS



- Williamson conjecture: For the first time, showed that Williamson matrices exist for all even orders up to 70. Verified that order 35 is the smallest counterexample to this conjecture.

- Ruskey-Savage Conjecture: Every matching of a hypercube of dimension d can be extended to a Hamiltonian cycle. We verified this conjecture up to dimension 5 for the first time.

- Complex Golay Conjecture: Enumerated all complex Golay sequences for lengths up to 25, except 23. Verified the Craigen–Holzmann–Kharaghani conjecture that such sequences do not exist in length 23

PART IV

A CASE STUDY

THE WILLIAMSON CONJECTURE

THE WILLIAMSON AND HADAMARD CONJECTURES
STATEMENT AND BACKGROUND

Hadamard conjecture (open since 1893):

A Hadamard matrix is an order n matrix H with entries in {+1,-1} such that any pair of rows (or columns) have inner product 0. The conjecture states that there exist a Hadamard matrix for order n = 4k for every positive integer k.

Given the difficulty of resolving this question, many special cases have been proposed. One of them is the class of Williamson matrices.

Williamson conjecture (first stated in 1972 by Turyn. Disproved in 1993 by Djokovic):

Let A, B, C, D be symmetric, circulant order n matrices with entries in {+1,-1} such that $A^2 + B^2 + C^2 + D^2 = 4nI_n$, *where* I_n *is the identity matrix of order n. The conjecture states that such matrices exist for every n.*

Such objects are called Williamson matrices of order n, and can be used to construct Hadamard matrices of order 4n.

Djokovic established, via some special-purpose computations, that there are no Williamson matrices of order 35.

WILLIAMSON SEQUENCES
STATEMENT AND BACKGROUND

Williamson sequences:

Fortunately, Williamson matrices can equivalently be defined using sequences of length n:

- *Sequences A, B, C, D of {+1,-1} of length n each, denoted as* $\{x_0, x_1, \ldots, x_{n-1}\}$
- *Sequences must symmetric, i.e.,* $x_i = x_{n-i}$ *for* $i = 1, \ldots, n-1$
- $PSD_A(s) + PSD_B(s) + PSD_C(s) + PSD_D(s) = 4n$ *for all* $s \in \mathbb{Z}$, *where PSD (Power Spectral Density) of a sequence X is the squared absolute values of the discrete Fourier transform of X.*
- *More precisely,*

$$PSD_A(s) = |DFT_A(s)|^2$$

$$where\ DFT_A(s) = \sum_{k=0}^{n-1} a_k e^{\left\{\frac{2\pi i k s}{n}\right\}}$$

Now the question is how do we take advantage of both SAT solvers and properties like the PSDs to efficiently find Williamson matrices?

CONSTRUCTING WILLIAMSON SEQUENCES
SAT+CAS ENCODING

Encoding Williamson sequences as a SAT+CAS problem:

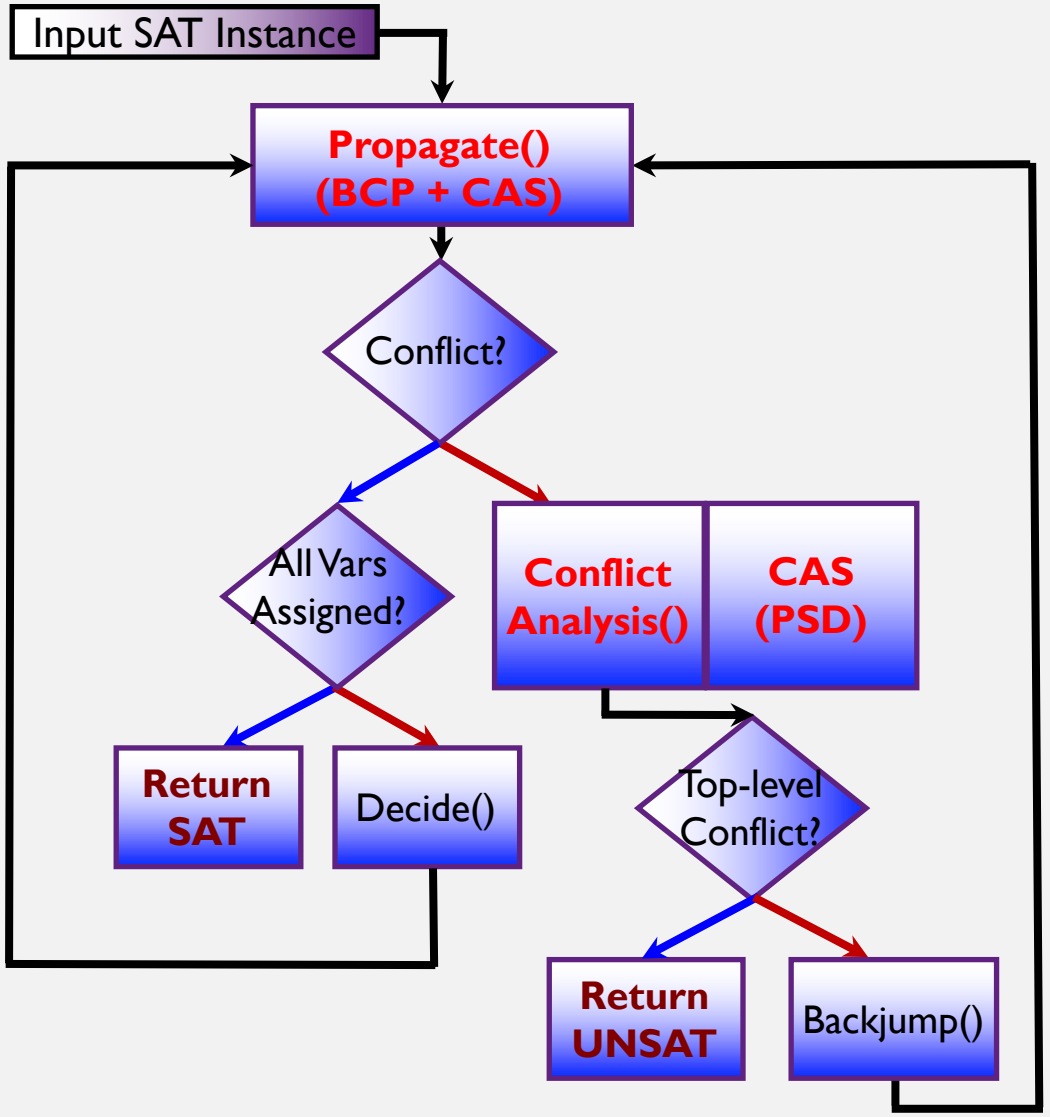
First, the SAT part:

- *Encode the domain values {+1,-1} for Williamson sequences as {true, false} in the corresponding SAT encoding*
- *Sequences A, B, C, D of {+1,-1} are now encoded as sets of Boolean variables* $\{x_0^A, x_1^A, \ldots, x_{n-1}^A\}$ *for each sequence*
- *Since the sequences must symmetric, we can cut down the number of Boolean variables in the encoding by half*
- *For arithmetic (e.g.,* $A^2 + B^2 + C^2 + D^2 = 4nI_n$*) and cardinality operations we use efficient Boolean circuits encodings, represented in conjunctive normal form*

Now, the CAS part:

- *There is no easy way to directly encode PSDs in SAT, other than to use an external CAS to perform these computations*

CONSTRUCTING WILLIAMSON SEQUENCES
SAT+CAS ENCODING



- SAT solver solves Williamson constraints and assigns values to the Boolean variables corresponding to the A,B,C, and D sequences
- When the solver reaches an inconclusive state (partial assignment that is neither satisfying nor a conflict), the CAS is invoked
- The CAS computes the PSD over the partial assignments:
  - Observe that since PSD values are non-negative and $PSD_A(s) + PSD_B(s) + PSD_C(s) + PSD_D(s) = 4n$,

    if $PSD_X(s) > 4n$ for any X, then X is not part of Williamson sequence. The CAS can reject such partial assignments (and thus all its extensions) by giving the SAT an appropriate conflict clause

Programmatic results

- The programmatic approach was found to perform much better than an approach which encoded the Williamson sequence definition using CNF clauses:

order n	programmatic speedup
20	4.33
22	7.00
24	7.12
26	27.00
28	52.56
30	52.21
32	58.16
34	138.37
36	317.61
38	377.84
40	428.71
42	1195.99
44	2276.09

ADDITIONAL PREDICATES FOR CAS COMPRESSIONS AND DIOPHANTINE EQUATIONS

- The PSD Criteria for s=0 essentially boils down to a Diophantine equation as follows:

$$\text{rowsum}(A)^2 + \text{rowsum}(B)^2 + \text{rowsum}(C)^2 + \text{rowsum}(D)^2 = 4n$$

- In other words, every Williamson sequence provides a decomposition of 4n into a sum of four squares

- There are usually only a few such decompositions. CAS (e.g., Maple) have functions to compute these

- The CAS system analyzes partial assignments from the SAT solver. If they don't satisfy the above criteria, then these partial assignments are converted into conflict clauses, and fed back to the SAT solver

Results

n	Time (h)	# instances	$\#W_n$
40	0.01	4032	1499
42	0.02	2945	301
44	0.01	1163	249
45	1.12	15542	1
46	0.03	1538	50
48	0.09	4008	9800
50	0.45	3715	275
51	4.57	17403	2
52	0.78	4535	926
54	3.00	25798	498
56	0.98	18840	40315
57	61.26	58376	1
58	15.97	9908	73
60	27.14	256820	4083
62	64.74	19418	61
63	1670.95	466561	2
64	65.52	34974	69960
66	764.96	109566	262
68	593.77	122150	1113
69	8162.50	600338	1
70	957.96	71861	98

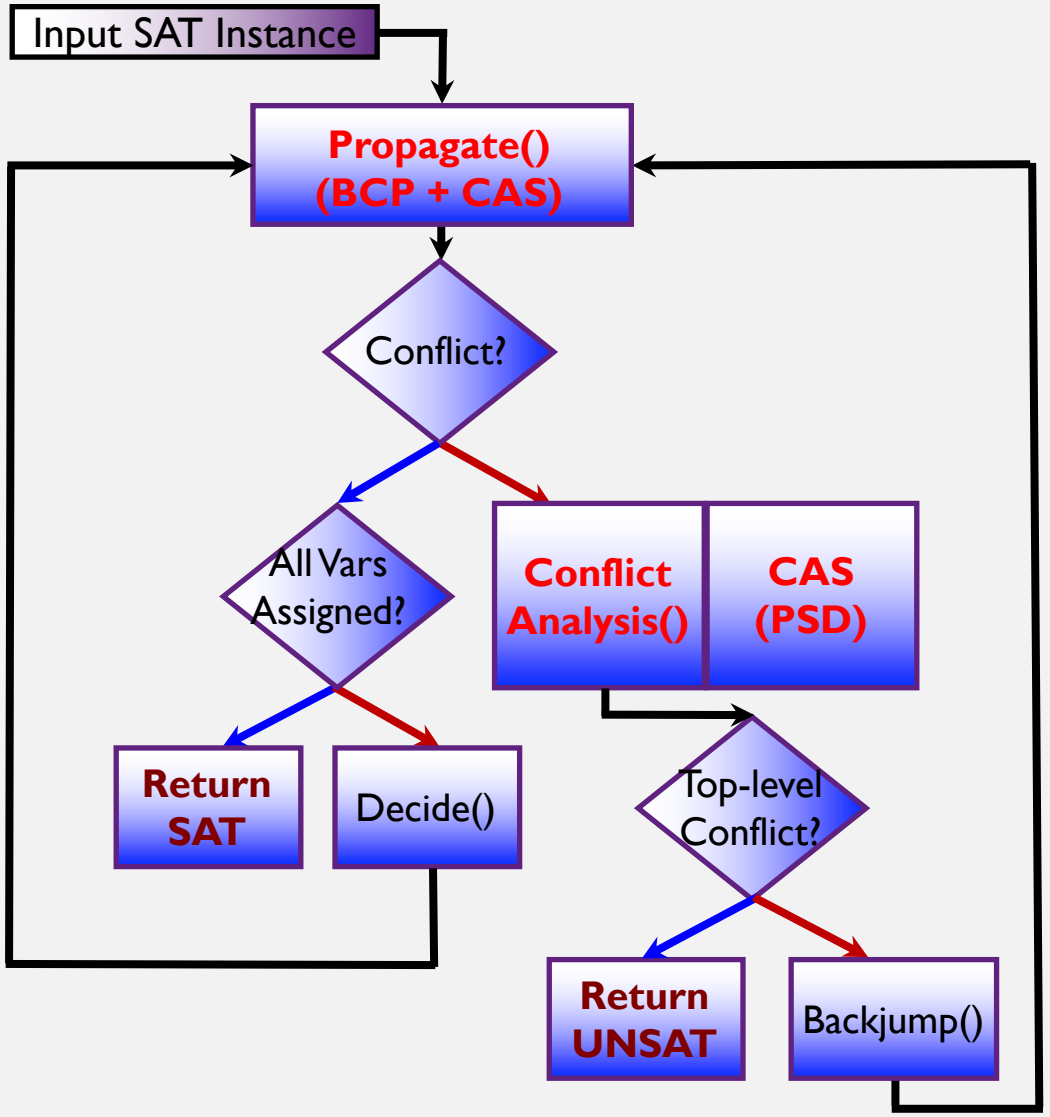
The amount of time used to generate and solve the SAT instances, the number of instances generated, and the number of Williamson sequences found ($\#W_n$).

PART V

SAT+CAS

TAKEAWAYS

SAT+CAS METHOD
TAKEAWAYS



- A powerful combination of SAT's general-purpose search ability and domain-specific capabilities of CAS for solving combinatorial problems
- Constructed counterexamples or finitely verified several open conjectures
- We are at a very beginning stages of a new way of solving combinatorial problems
- Next steps involve understanding better encodings, improved ways of CAS to provide feedback, and integrating SAT+CAS in novel ways

PART VI

Understanding SAT and SMT

A PROOF COMPLEXITY-THEORETIC VIEW

THE CONTEXT
PROOF SYSTEMS AND THEIR COMPLEXITY

General resolution — The rule is form of modus ponens. Proof is a directed acyclic graph (DAG).

$$\frac{(x_1 \vee \cdots \vee x_n) \quad (\neg x_n \vee y_1 \ldots \vee y_m)}{(x_1 \vee \cdots \vee x_{n-1} \vee y_1 \ldots \vee y_m)}$$

Merge resolution — Derived clauses have to share literals to apply rule. Proof is a DAG.

$$\frac{(x_1 \vee \cdots \vee x_n) \quad (\neg x_n \vee \cdots x_{n-1})}{(x_1 \vee \cdots \vee x_{n-1})}$$

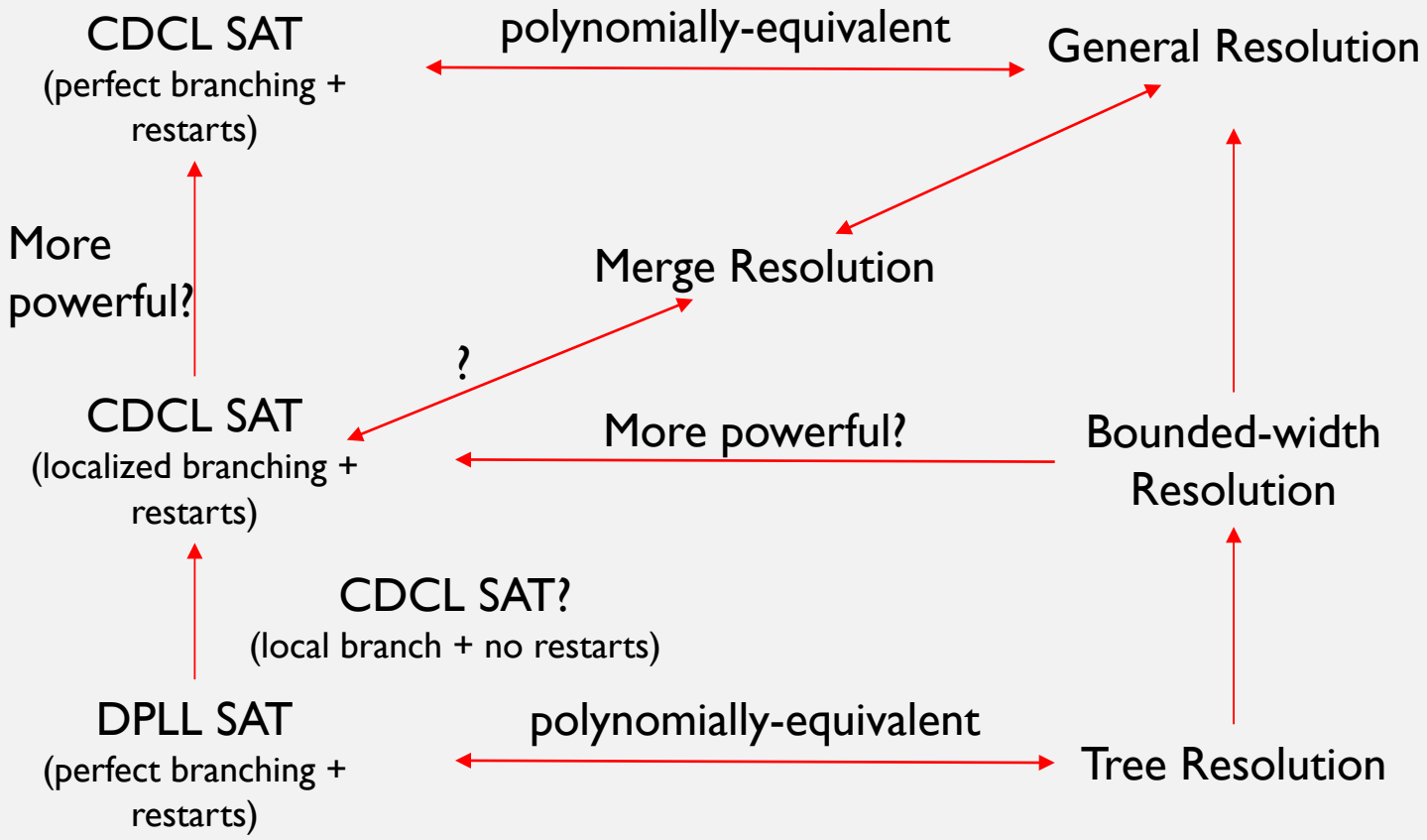
Unit resolution — One clause must be unit. Proof is a DAG.

$$\frac{(x_n) \quad (\neg x_n \vee y_1 \ldots y_m)}{(y_1 \vee \cdots \vee y_m)}$$

Tree resolution — Same rules as general resolution. Proof is a tree. Tree proofs are not allowed to reuse lemmas unlike DAG proofs.

THE CONTEXT
PROOF-COMPLEXITY FOR SAT/SMT SOLVERS

- SAT solvers are sound and complete proof systems for Boolean logic
- Proof systems = axioms + proof rules
- Parameterized proof complexity is an ideal lens to study SAT solvers
- The proof complexity-theoretic research program can be extended to all of formal methods
- What is needed is the discovery of appropriate parameters
- Conversely, building practical proof system can lend insights into hard proof complexity-theoretic questions



MERGE RESOLUTION
MERGEABILITY AND CDCL SAT SOLVERS

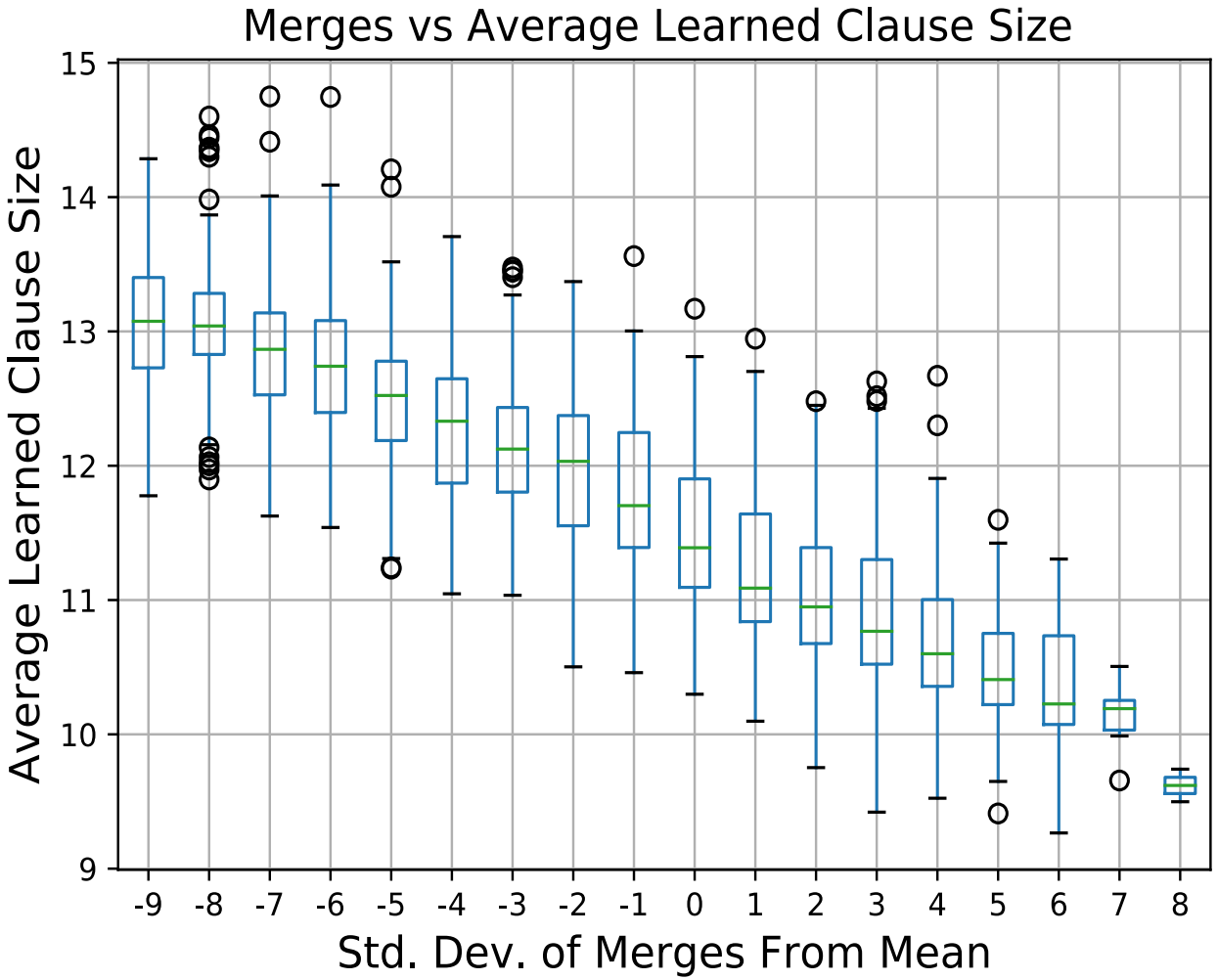
$$\frac{(\alpha \vee x_n) \quad (\neg x_n \vee \beta)}{(\alpha \vee \beta)} \qquad \frac{(\gamma \vee y_n) \quad (\neg y_n \vee \delta)}{(\gamma \vee \delta)}$$

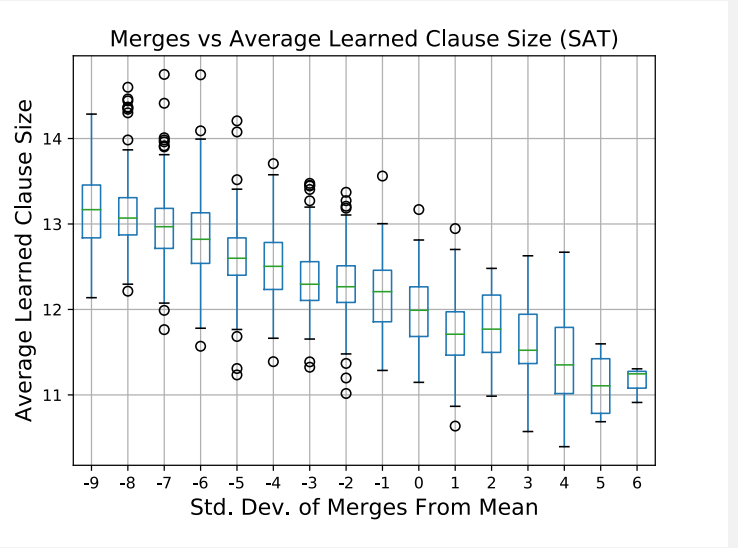
...

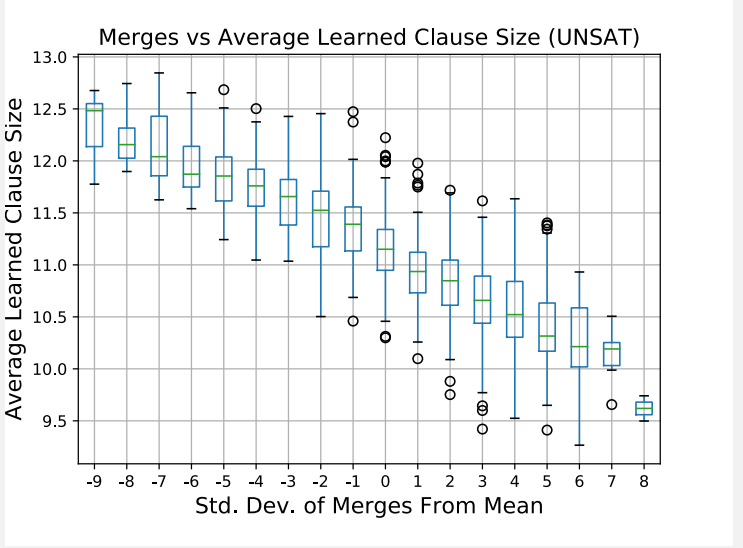
$$\frac{(x_i \vee x_j) \quad (\neg x_j \vee x_i)}{(x_i)} \qquad \frac{(\neg x_i \vee x_k) \quad (\neg x_k \vee \neg x_i)}{(\neg x_i)}$$

$$\frac{(x_i) \quad (\neg x_i)}{\bot}$$

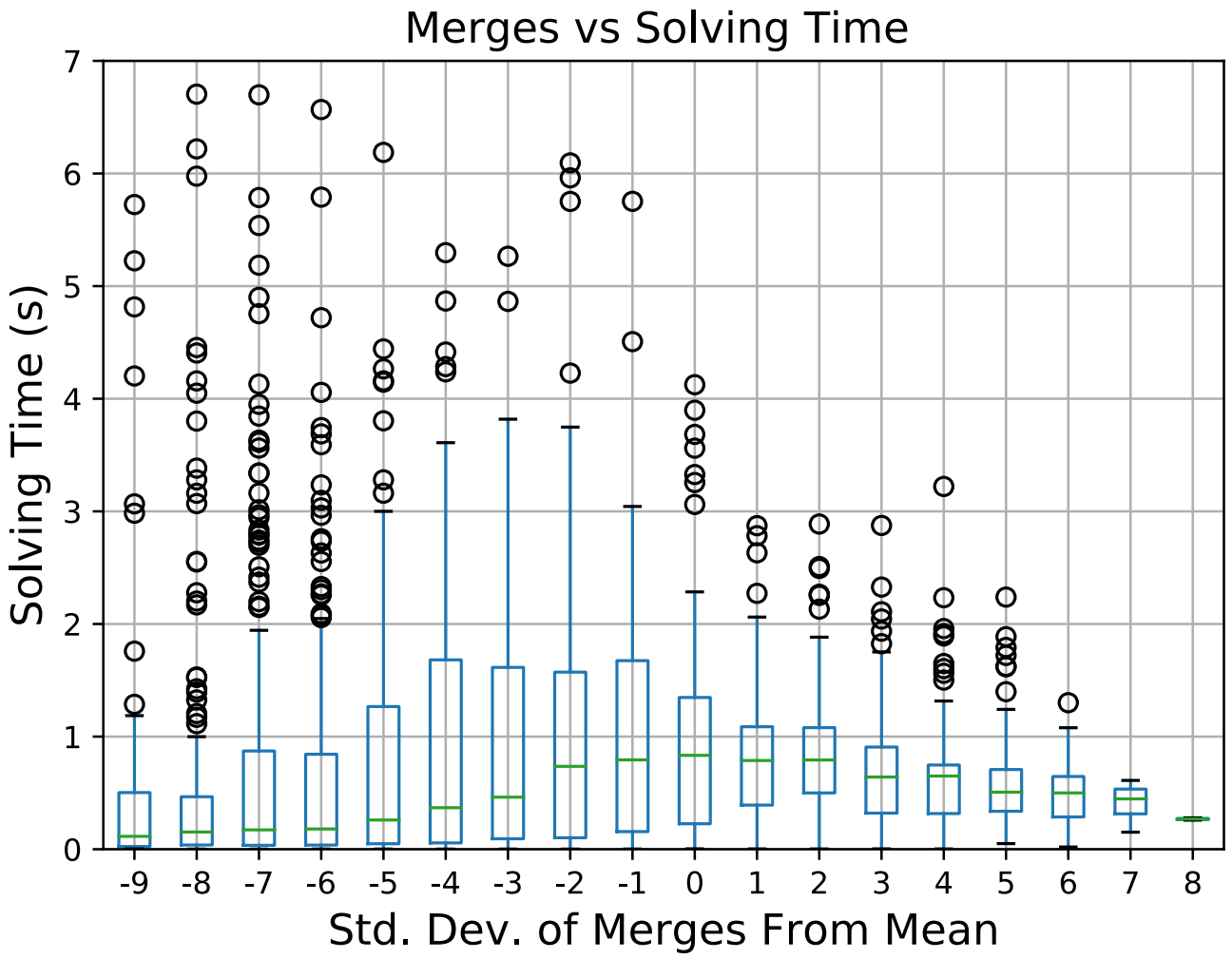
MERGEABILITY
NUMBER OF MERGES VS. LEARNT CLAUSE SIZE RESULT

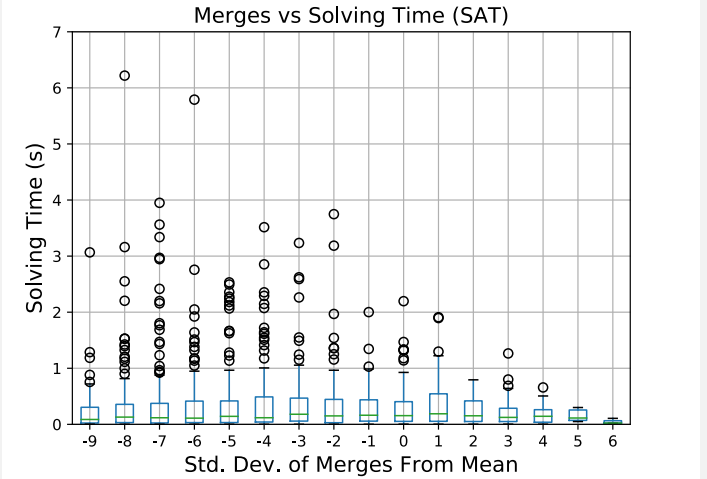


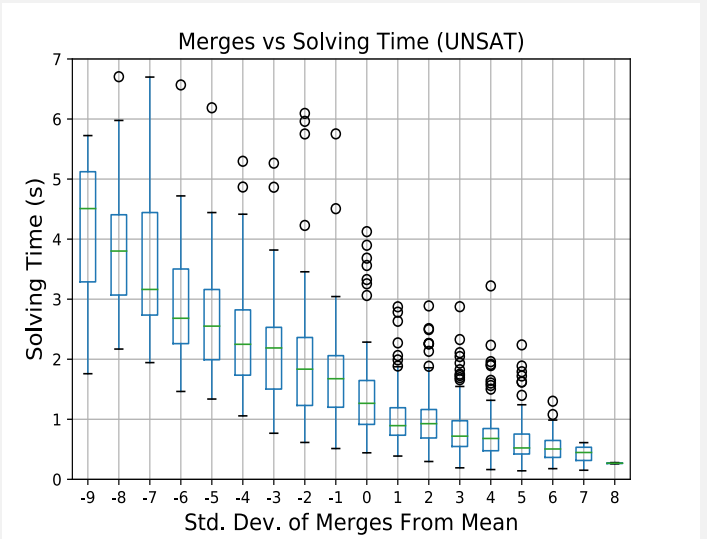




MERGEABILITY
NUMBER OF MERGES VS. SOLVER RUNTIME RESULT







MERGEABILITY
EXPERIMENTAL AND THEORETICAL RESULTS

- Number of merges: for every pair of resolvable clauses, count number of overlapping literals (normalized by size of resolvable clauses)

- Three results

  - Number of merges in the input formula correlate well with solver runtime for industrial instances

  - Scaling study: As we increase the number of merges for random k-SAT instances, solver runtime falls (esp. for UNSAT instances) and learnt clause length decreases (for all instances)

  - Also, we prove that CDCL SAT solvers are polynomially equivalent to merge resolution

PROOF COMPLEXITY OF SMT SOLVERS
UNDERSTANDING THE POWER OF CDCL(T)

Result I: We show that CDCL(T) solvers (aka SMT solvers) can be modeled as Res(T) proof systems, i.e., general resolution enhanced with theory reasoning.

Result II: If the theory solver is allowed to introduce new variables, then CDCL(T) solvers are far more powerful, and are equivalent to Res*(T) proof systems.

Result III: Even for very weak theories such as EUF, we further show that Res*(EUF) is as powerful as Frege, which in turn is far more powerful than general resolution.

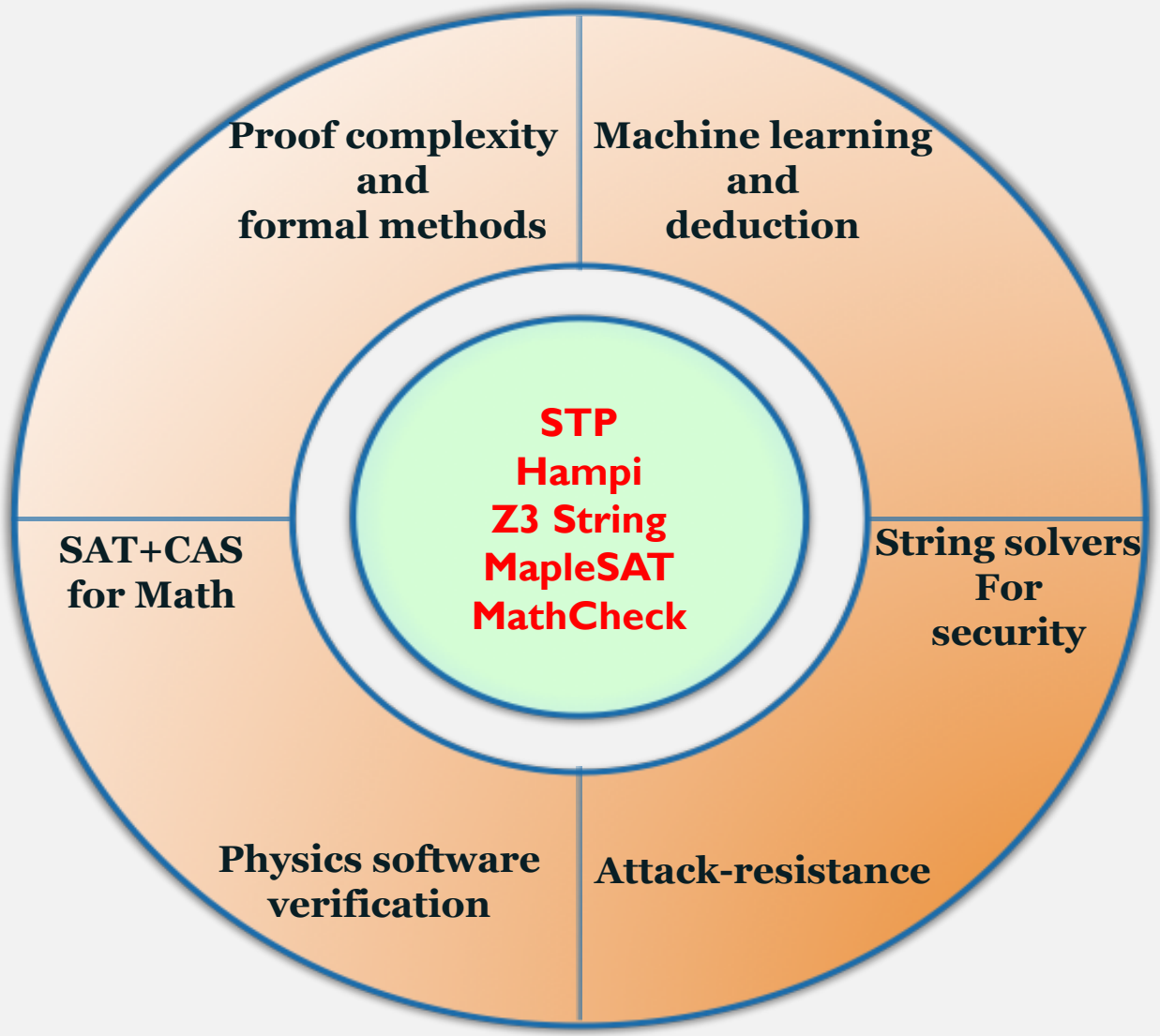
Message: We can use proof complexity to establish the relative strength of SAT solvers and CDCL(T) solvers. Can we extend this to suitable SAT+CAS approaches to better understand when to apply the power of CAS?

PART VII

OTHER RESEARCH

SAT, SMT, AND APPLICATIONS

CURRENT RESEARCH PROGRAM



CURRENT PROJECTS AND CONTRIBUTIONS

Name	Key Concept	Impact	Publication venues
Maple Series of SAT Solvers, and understanding SAT	Machine learning for solver heuristics, and community structure	Won gold and silver medals at SAT Competition 2016 and 2017	AAAI 2016, SAT 2016/17 HVC 2015, SAT 2014[3]
Z3str3 String and Integer Solver[1]	Novel techniques for string + integer combination	Analysis of Web Apps (Part of Z3 codebase)	FSE 2013, CAV 2015[3] FMSD 2017
MathCheck Conjecture Verifier	CAS+SAT combination a la DPLL(CAS)	Williamson, complex Golay, Ruskey-Savage conjectures,..	IJCAI 2016, CASC 2016, JAR 2017, CADE 2015[3]
Undecidability results for theories over strings	Boundary between decidability and undecidability	Solved problems open since early 2000's	HVC 2012
Attack-resistance	A new approach to formally establishing the efficacy of security defenses	Mathematical guarantee of software trustworthiness even in the presence of bugs	Euro S&P 2017 PLAS 2015

1. 100+ research projects use STP, HAMPI, and Z3str2.
2. STP won the SMTCOMP 2006/2010 and second in 2011/2014 competitions for bit-vector solvers
3. Best paper awards/honors at various conferences including SAT, DATE, SPLC, CAV, and CADE
4. Retargetable Compiler (DATE 1999, 2008). Ten Year Most-influential paper award at DATE 2008. ACM Test of Time Award 2016.

PAST CONTRIBUTIONS

Name	Key Concept	Impact	Publication venues
STP Bit-vector & Array Solver[1,2]	Abstraction-refinement for solving	Concolic Testing	CAV 2007 CCS 2006 TISSEC 2008
HAMPI String Solver[1]	App-driven bounding for solving	Analysis of Web Apps	ISSTA 2009[3] TOSEM 2012 CAV 2011
Taint-based Fuzzing	Data flow is cheaper than concolic	Scales better than concolic	ICSE 2009
Automatic Input Rectification	Automatic security envelope	New security approach	ICSE 2012

1. 100+ research projects use STP, HAMPI, and Z3str2.
2. STP won the SMTCOMP 2006/2010 and second in 2011/2014 competitions for bit-vector solvers
3. Best paper awards/honors at various conferences including SAT, DATE, SPLC, CAV, and CADE
4. Retargetable Compiler (DATE 1999, 2008). Ten Year Most-influential paper award at DATE 2008.

TAKEAWAYS

Message1: SAT+CAS is a powerful approach for many problems in combinatorial mathematics. We have addressed several problems (e.g., Williamson conjecture) and are working on many more

Message 2: CDCL(T) or SMT solvers can be viewed as Res*(T) proof systems, and CDCL SAT solvers as merge resolution proof systems

Message 3: As we develop SAT+CAS, we will need methods from proof complexity to understand the strengths and weaknesses of each version (SAT+CAS' vs. SAT+CAS") of this approach

REFERENCES

- [BKS03] Beame, P., Kautz, H., Sabharwal, A., 2003. Understanding the Power of Clause Learning. In IJCAI 2003
- [WGS03] Williams, R., Gomes, C.P. and Selman, B., 2003. Backdoors to typical case complexity. In IJCAI 2003
- [BSG09] Dilkina, B., Gomes, C.P. and Sabharwal, A., 2009. Backdoors in the context of learning. In SAT 2009
- [AL12] Ansótegui, C., Giráldez-Cru, J. and Levy, J., 2012. The community structure of SAT formulas. In SAT 2012
- [NGFAS14] Newsham, Z., Ganesh, V., Fischmeister, S., Audemard, G. and Simon, L., 2014. Impact of community structure on SAT solver performance. In SAT 2014
- [KSM11] Katebi, H., Sakallah, K.A. and Marques-Silva, J.P., 2011. Empirical study of the anatomy of modern SAT solvers. In SAT 2011
- [LGZC15] Liang, J.H., Ganesh, V., Zulkoski, E., Zaman, A. and Czarnecki, K., 2015. Understanding VSIDS branching heuristics in CDCL SAT solvers. In HVC 2015.
- [MMZZM01] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L. and Malik, S., 2001. Chaff: Engineering an efficient SAT solver. In DAC 2001
- [B09] Biere, A., 2008. Adaptive restart strategies for conflict driven SAT solvers. In SAT 2008
- [LGPC16] Liang, J.H., Ganesh, V., Poupart, P., and Czarnecki, K. Learning Rate Based Branching Heuristic for SAT Solvers. In SAT 2016
- [LGPC+16] Liang, J.H., Ganesh, V., Poupart, P., and Czarnecki, K. Conflict-history Based Branching Heuristic for SAT Solvers. In AAAI 2016
- [LGRC15] Liang, J.H., Ganesh, V., Raman, V., and Czarnecki, K. SAT-based Analysis of Large Real-world Feature Models are Easy. In SPLC 2015