Solving and Verifying Hard Problems using SAT

Marijn J.H. Heule

THE UNIVERSITY OF TEXAS AT AUSTIN

SAT Solving and Verification

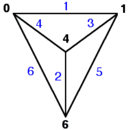
Solving Framework for Hard Problems

The Future: Verified SAT via Proofs

SAT Solving and Verification

Satisfiability (SAT) solving has many applications

formal verification

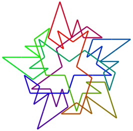


graph theory

bioinformatics

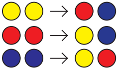
train safety

planning



number theory

cryptography



rewrite termination

encode → **SAT solver** → **decode**

A Small Satisfiability (SAT) Problem

$$
\begin{aligned}
&(x_5 \vee x_8 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_3 \vee \bar{x}_7) \wedge (\bar{x}_5 \vee x_3 \vee x_8) \wedge \\
&(\bar{x}_6 \vee \bar{x}_1 \vee \bar{x}_5) \wedge (x_8 \vee \bar{x}_9 \vee x_3) \wedge (x_2 \vee x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_8 \vee x_4) \wedge \\
&(\bar{x}_9 \vee \bar{x}_6 \vee x_8) \wedge (x_8 \vee x_3 \vee \bar{x}_9) \wedge (x_9 \vee \bar{x}_3 \vee x_8) \wedge (x_6 \vee \bar{x}_9 \vee x_5) \wedge \\
&(x_2 \vee \bar{x}_3 \vee \bar{x}_8) \wedge (x_8 \vee \bar{x}_6 \vee \bar{x}_3) \wedge (x_8 \vee \bar{x}_3 \vee \bar{x}_1) \wedge (\bar{x}_8 \vee x_6 \vee \bar{x}_2) \wedge \\
&(x_7 \vee x_9 \vee \bar{x}_2) \wedge (x_8 \vee \bar{x}_9 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_9 \vee x_4) \wedge (x_8 \vee x_1 \vee \bar{x}_2) \wedge \\
&(x_3 \vee \bar{x}_4 \vee \bar{x}_6) \wedge (\bar{x}_1 \vee \bar{x}_7 \vee x_5) \wedge (\bar{x}_7 \vee x_1 \vee x_6) \wedge (\bar{x}_5 \vee x_4 \vee \bar{x}_6) \wedge \\
&(\bar{x}_4 \vee x_9 \vee \bar{x}_8) \wedge (x_2 \vee x_9 \vee x_1) \wedge (x_5 \vee \bar{x}_7 \vee x_1) \wedge (\bar{x}_7 \vee \bar{x}_9 \vee \bar{x}_6) \wedge \\
&(x_2 \vee x_5 \vee x_4) \wedge (x_8 \vee \bar{x}_4 \vee x_5) \wedge (x_5 \vee x_9 \vee x_3) \wedge (\bar{x}_5 \vee \bar{x}_7 \vee x_9) \wedge \\
&(x_2 \vee \bar{x}_8 \vee x_1) \wedge (\bar{x}_7 \vee x_1 \vee x_5) \wedge (x_1 \vee x_4 \vee x_3) \wedge (x_1 \vee \bar{x}_9 \vee \bar{x}_4) \wedge \\
&(x_3 \vee x_5 \vee x_6) \wedge (\bar{x}_6 \vee x_3 \vee \bar{x}_9) \wedge (\bar{x}_7 \vee x_5 \vee x_9) \wedge (x_7 \vee \bar{x}_5 \vee \bar{x}_2) \wedge \\
&(x_4 \vee x_7 \vee x_3) \wedge (x_4 \vee \bar{x}_9 \vee \bar{x}_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge \\
&(x_6 \vee x_7 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_6 \vee x_2 \vee x_3) \wedge (\bar{x}_8 \vee x_2 \vee x_5)
\end{aligned}
$$

Does there exist an assignment satisfying all clauses?

Search for a satisfying assignment (or proof none exists)

$$
\begin{aligned}
&(x_5 \lor x_8 \lor \bar{x}_2) \land (x_2 \lor \bar{x}_1 \lor \bar{x}_3) \land (\bar{x}_8 \lor \bar{x}_3 \lor \bar{x}_7) \land (\bar{x}_5 \lor x_3 \lor x_8) \land \\
&(\bar{x}_6 \lor \bar{x}_1 \lor \bar{x}_5) \land (x_8 \lor \bar{x}_9 \lor x_3) \land (x_2 \lor x_1 \lor x_3) \land (\bar{x}_1 \lor x_8 \lor x_4) \land \\
&(\bar{x}_9 \lor \bar{x}_6 \lor x_8) \land (x_8 \lor x_3 \lor \bar{x}_9) \land (x_9 \lor \bar{x}_3 \lor x_8) \land (x_6 \lor \bar{x}_9 \lor x_5) \land \\
&(x_2 \lor \bar{x}_3 \lor \bar{x}_8) \land (x_8 \lor \bar{x}_6 \lor \bar{x}_3) \land (x_8 \lor \bar{x}_3 \lor \bar{x}_1) \land (\bar{x}_8 \lor x_6 \lor \bar{x}_2) \land \\
&(x_7 \lor x_9 \lor \bar{x}_2) \land (x_8 \lor \bar{x}_9 \lor x_2) \land (\bar{x}_1 \lor \bar{x}_9 \lor x_4) \land (x_8 \lor x_1 \lor \bar{x}_2) \land \\
&(x_3 \lor \bar{x}_4 \lor \bar{x}_6) \land (\bar{x}_1 \lor \bar{x}_7 \lor x_5) \land (\bar{x}_7 \lor x_1 \lor x_6) \land (\bar{x}_5 \lor x_4 \lor \bar{x}_6) \land \\
&(\bar{x}_4 \lor x_9 \lor \bar{x}_8) \land (x_2 \lor x_9 \lor x_1) \land (x_5 \lor \bar{x}_7 \lor x_1) \land (\bar{x}_7 \lor \bar{x}_9 \lor \bar{x}_6) \land \\
&(x_2 \lor x_5 \lor x_4) \land (x_8 \lor \bar{x}_4 \lor x_5) \land (x_5 \lor x_9 \lor x_3) \land (\bar{x}_5 \lor \bar{x}_7 \lor x_9) \land \\
&(x_2 \lor \bar{x}_8 \lor x_1) \land (\bar{x}_7 \lor x_1 \lor x_5) \land (x_1 \lor x_4 \lor x_3) \land (x_1 \lor \bar{x}_9 \lor \bar{x}_4) \land \\
&(x_3 \lor x_5 \lor x_6) \land (\bar{x}_6 \lor x_3 \lor \bar{x}_9) \land (\bar{x}_7 \lor x_5 \lor x_9) \land (x_7 \lor \bar{x}_5 \lor \bar{x}_2) \land \\
&(x_4 \lor x_7 \lor x_3) \land (x_4 \lor \bar{x}_9 \lor \bar{x}_7) \land (x_5 \lor \bar{x}_1 \lor x_7) \land (x_5 \lor \bar{x}_1 \lor x_7) \land \\
&(x_6 \lor x_7 \lor \bar{x}_3) \land (\bar{x}_8 \lor \bar{x}_6 \lor \bar{x}_7) \land (x_6 \lor x_2 \lor x_3) \land (\bar{x}_8 \lor x_2 \lor x_5)
\end{aligned}
$$

Solutions are easy to verify, but what about unsatisfiability?

Motivation for validating unsatisfiability proofs

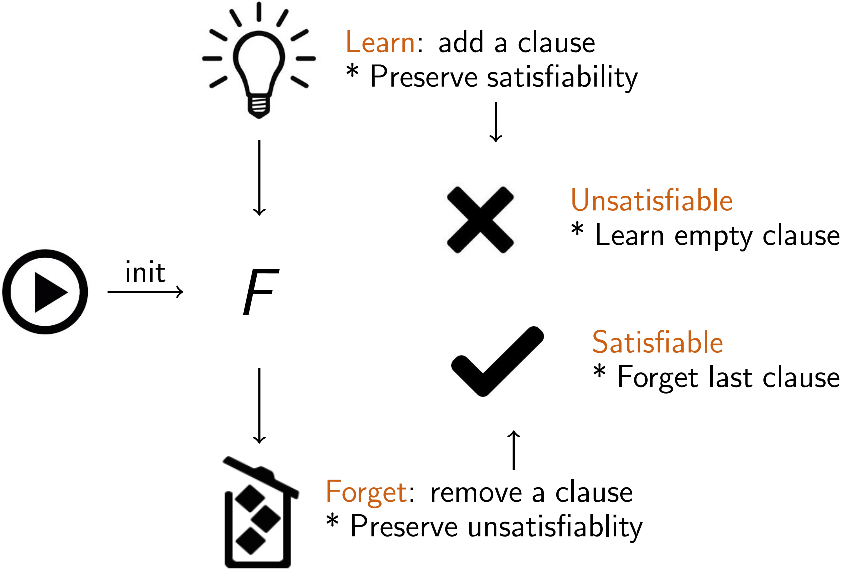
Satisfiability solvers are used in amazing ways...

- Hardware and software verification (Intel and Microsoft)
- Hard-Combinatorial problems:
  - van der Waerden numbers [Dransfield, Marek, and Truszczynski, 2004; Kouril and Paul, 2008]
  - Gardens of Eden in Conway's Game of Life [Hartman, Heule, Kwekkeboom, and Noels, 2013]
  - Erdős Discrepancy Problem [Konev and Lisitsa, 2014]

..., but satisfiability solvers have errors and only return yes/no.

- Documented bugs in SAT, SMT, and QBF solvers [Brummayer and Biere, 2009; Brummayer et al., 2010]
- Implementation errors often imply conceptual errors
- Mathematical results require a stronger justification than a simple yes/no by a solver. UNSAT must be checkable.

Clausal Proof System [Järvisalo, Heule, and Biere 2012]



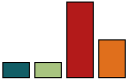
Ideal Properties of a Proof System for SAT Solvers

Easy to Emit

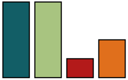
Compact

Checked Efficiently

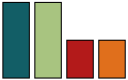
Expressive



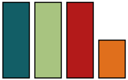
Resolution Proofs
Zhang and Malik, 2003
Van Gelder, 2008; Biere, 2008



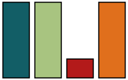
Clausal Proofs
Goldberg and Novikov, 2003
Van Gelder, 2008



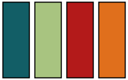
Clausal proofs + clause deletion
Heule, Hunt, Jr., and Wetzler [STVR 2014]



Optimized clausal proof checker
Heule, Hunt, Jr., and Wetzler [FMCAD 2013]



Clausal RAT proofs
Heule, Hunt, Jr., and Wetzler [CADE 2013]



DRAT proofs (RAT + deletion)
Wetzler, Heule, and Hunt, Jr. [SAT 2014]

Ideal Properties of a Proof System for SAT Solvers

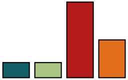
Easy to Emit

Compact

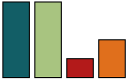
Checked Efficiently

Expressive

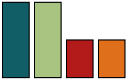
Verified



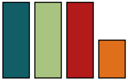
Resolution Proofs
Zhang and Malik, 2003
Van Gelder, 2008; Biere, 2008



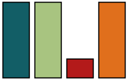
Clausal Proofs
Goldberg and Novikov, 2003
Van Gelder, 2008



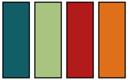
Clausal proofs + clause deletion
Heule, Hunt, Jr., and Wetzler [STVR 2014]



Optimized clausal proof checker
Heule, Hunt, Jr., and Wetzler [FMCAD 2013]



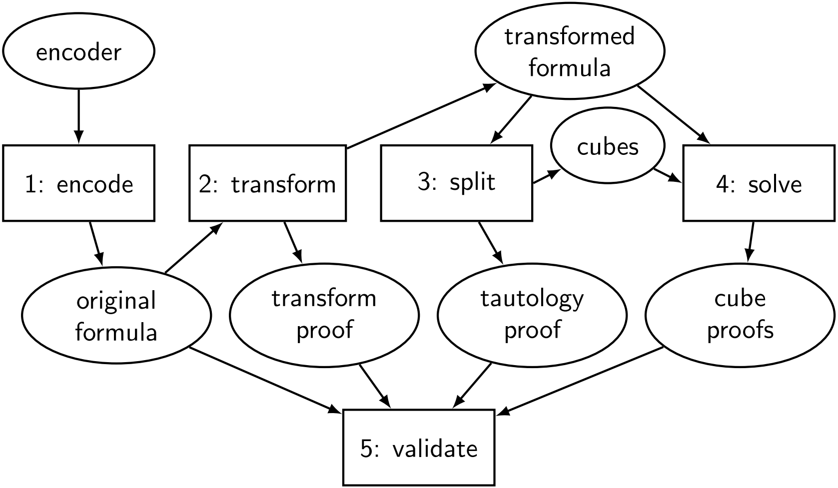
Clausal RAT proofs
Heule, Hunt, Jr., and Wetzler [CADE 2013]



DRAT proofs (RAT + deletion)
Wetzler, Heule, and Hunt, Jr. [SAT 2014]

Solving Framework for Hard-Combinatorial Problems

Overview of Solving Framework



Case Study: Pythagorean Triples Problem [Graham 1980]

Can the set of natural numbers $\{1, 2, 3, \ldots\}$ be partitioned into two parts such that no part contains a Pythagorean triple ($a, b, c \in \mathbb{N}$ with $a^2 + b^2 = c^2$)?

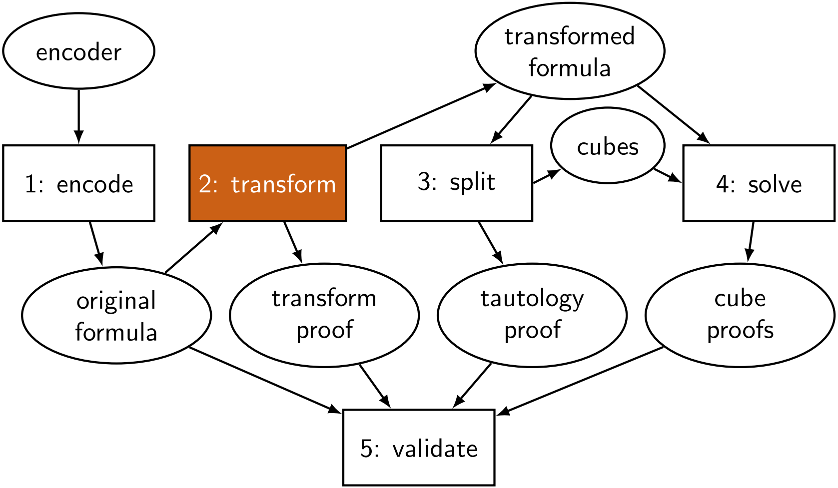
A computer program can partition the first several thousands numbers ($\{1, \ldots, 7664\}$) [Cooper and Overstreet 2015].

A partition into two parts is encoded using Boolean variables x_i with $i \in \{1, 2, 3, \ldots, n\}$ such that $x_i = 1$ ($= 0$) means that i occurs in Part 1 (Part 2). For each Pythagorean triple (a, b, c) two clauses are added: $(x_a \vee x_b \vee x_c) \wedge (\bar{x}_a \vee \bar{x}_b \vee \bar{x}_c)$.

Theorem (Main result via parallel SAT solving)

$[1, 7824]$ *can be partitioned into two parts, such that no part contains a Pythagorean triple. This is impossible for* $[1, 7825]$.

Highlight: Phase 2



Phase 2: Transform

Input: original CNF formula

Output: transformed CNF formula and a transformation proof

Goal: optimize the formula regarding the later (solving) phases

We applied two transformations (realized via blocked clauses):

- Pythagorean Triple Elimination removes Pythagorean Triples that contain an element that does not occur in any other Pythagorean Triple, e.g. $3^2 + 4^2 = 5^2$. (till fixpoint)
- Symmetry breaking places the number most frequently occurring in Pythagorean triples (2520) in Part 1 (encode).

All transformation (pre-processing) techniques can be expressed using RAT steps [Järvisalo, Heule, and Biere 2012].

Phase 2: Blocked Clauses [Kullmann'99]

Definition (Blocking literal)

A literal l in a clause C of a CNF F blocks C w.r.t. F if for every clause $D \in F_{\bar{l}}$, the resolvent $(C \setminus \{l\}) \cup (D \setminus \{\bar{l}\})$ obtained from resolving C and D on l is a tautology.

With respect to a fixed CNF and its clauses we have:

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Phase 2: Blocked Clauses [Kullmann'99]

Definition (Blocking literal)

A literal l in a clause C of a CNF F blocks C w.r.t. F if for every clause $D \in F_{\bar{l}}$, the resolvent $(C \setminus \{l\}) \cup (D \setminus \{\bar{l}\})$ obtained from resolving C and D on l is a tautology.

With respect to a fixed CNF and its clauses we have:

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Example (Blocking literals and blocked clauses)

Consider the formula $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$.
First clause is not blocked.
Second clause is blocked by both a *and* $\bar{c}$.
Third clause is blocked by c.

Phase 2: Blocked Clauses [Kullmann'99]

Definition (Blocking literal)

A literal l in a clause C of a CNF F blocks C w.r.t. F if for every clause $D \in F_{\bar{l}}$, the resolvent $(C \setminus \{l\}) \cup (D \setminus \{\bar{l}\})$ obtained from resolving C and D on l is a tautology.

With respect to a fixed CNF and its clauses we have:

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Example (Blocking literals and blocked clauses)

Consider the formula $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$.
First clause is not blocked.
Second clause is blocked by both a *and* $\bar{c}$.
Third clause is blocked by c.

Proposition

Removal of an arbitrary blocked clause preserves unsatisfiability.

Phase 2: Blocked Clause Elimination (BCE)

Definition (BCE)

While a clause C in a formula F is blocked, remove C from F.

Example (BCE)

Consider $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$*.*
After removing either $(a \vee \bar{b} \vee \bar{c})$ *or* $(\bar{a} \vee c)$*, the clause* $(a \vee b)$ *becomes blocked (no clause with either* $\bar{b}$ *or* $\bar{a}$*).*
An extreme case in which BCE removes all clauses!

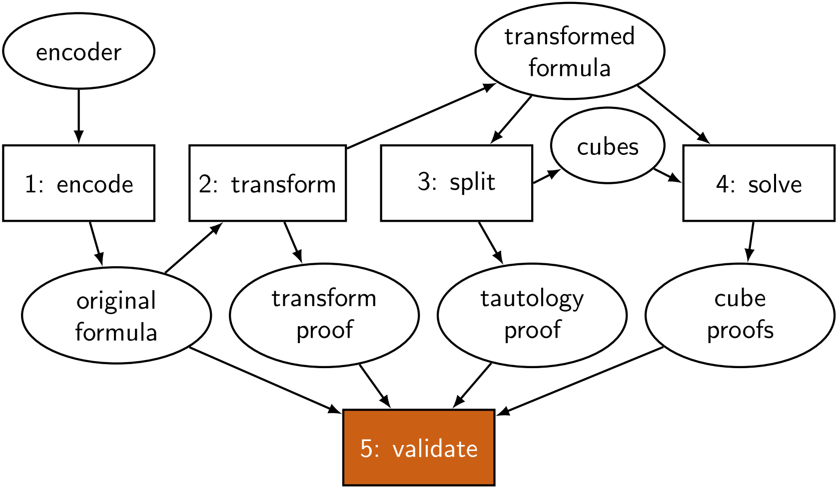
Example (Pythagorean Triples)

The clauses $(x_3 \vee x_4 \vee x_5)$ *and* $(\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5)$ *are blocked in* F_{7824} *and* F_{7825} *(actually in any* F_n*).*
BCE (F_{7824}) *has* 3740 *variables and* 14652 *clauses, and*
BCE (F_{7825}) *has* 3745 *variables and* 14672 *clauses.*

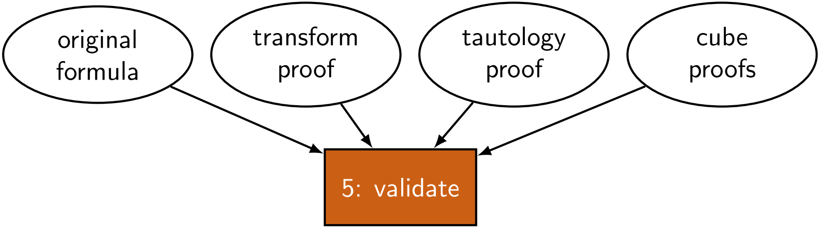
BCE can simulate many high-level reasoning techniques.

[Järvisalo, Biere, and Heule 2010]

Highlight: Phase 5



Phase 5: Validate Pythagorean Triples Proofs.



We check the proofs with the DRAT-trim checker, which has been used to validate the UNSAT results of the international SAT Competitions since 2013.

Recently it was shown how to validate DRAT proofs in parallel [Heule and Biere 2015].

The size of the merged proof is almost 200 terabyte and has been validated in 16,000 CPU hours.

The Future:
Verified SAT via Proofs

Next Step: Verify the Proof Checker

Since 2013 all results of the SAT Competitions are validated.
The main question asked: can you verify the checker?

The verified RAT checker [Wetzler, Heule, and Hunt 2013] is too slow for practical use (say 1000$\times$ slower compared to C).

The ACL2 theorem prover allows implementing (and verifying) a checker that is only about 60% slower compared to C.

Nathan Wetlzer started to work on this during his PhD thesis, but still quite some work is required.

Second Next Step: Verified SAT Solving

Question: Would it be possible to implement a fast mechanically-verified SAT solver?

SAT solvers are constantly being improved using a vast amount of complex techniques making mechanical verification hard.

However, given a fast mechanically-verified proof checker, SAT solver implementations do not have to be verified.

SAT solver can simply be used as an oracle to produce proofs.

Future: Combining it all to have proofs for hard problems

Next steps in verified SAT solving:

- Develop a fast mechanically-verified, clausal proof checker;
- Implement a fast, proof-producing SAT solving in ACL2.

Integrate the solving framework in a theorem prover:

- Show that the encoding of problems into SAT is correct;
- Show the correctness of clausal proof decomposition.

Apply our solving framework to various hard problems:

- Obtain and verify a proof for Radziszowski's and McKay's big result [1995] in Ramsey Theory: $R(4,5) = 25$;
- Century-old open problems appear solvable now, such as Schur number $S(5)$.

Future: Combining it all to have proofs for hard problems

Next steps in verified SAT solving:

- Develop a fast mechanically-verified, clausal proof checker;
- Implement a fast, proof-producing SAT solving in ACL2.

Integrate the solving framework in a theorem prover:

- Show that the encoding of problems into SAT is correct;
- Show the correctness of clausal proof decomposition.

Apply our solving framework to various hard problems:

- Obtain and verify a proof for Radziszowski's and McKay's big result [1995] in Ramsey Theory: $R(4,5) = 25$;
- Century-old open problems appear solvable now, such as Schur number $S(5)$.

Thanks!