Trustworthy Boolean Reasoning 1: (Un)Satisfiability and Proofs

Randal E. Bryant

Carnegie Mellon University

June, 2022

Important Ideas for These Lectures

- SAT solvers are useful tools
  - Many practical problems reducible to SAT
  - Need to learn effective encoding techniques
- For many applications, formulas should be unsatisfiable
  - Program should generate a checkable proof
  - There is a well-developed proof infrastructure
- Binary Decision Diagrams (BDDs) can play important role
  - In supplementing current SAT algorithms
  - In proof generation

SAT Application: Bit-Level Program Verification

Can the assertion fail?

```
int abs_bits(int x) {
  int m = x>>31;
  return (x^m) + ~m + 1;
}

int abs_ref(int x) {
  return x < 0 ? -x : x;
}
```

```
int main() {
  /* Value of t arbitrary */
  int t = random();
  int ar = abs_ref(t);
  int ab = abs_bits(t);
  int err = (ar != ab);
  assert(!err);
}
```

Assume for `int`:

- 32-bit word
- Two's complement representation

Application: Bit-Level Program Verification

C Bounded Model Checker (CBMC)

- Clarke, Kroening, Lerda TACAS 2004

Reduces Program Verification to SAT

- Unroll loops by bounded amount
- Encode arithmetic and logical operations at Boolean level
- Formula satisfied if `err` can be nonzero
  - *Unsatisfiable when no error can occur*

Widely Used in Industry

- Accurately models low-level program behavior
- Good for short, but tricky programs

SAT Application: Coloring Pythagorean Triples

Pythagorean Triple (P-Triple)

- Positive integers a, b, c such that $a^2 + b^2 = c^2$
- E.g., $a = 3$, $b = 4$, $c = 5$.

Two-Coloring

- For integers $i \in \{1, 2, \ldots, n\}$, assign $C_i \in \{\text{red}, \text{blue}\}$
- For every P-Triple a, b, c, cannot have $C_a = C_b = C_c$.

Question

- What is the maximum n for which a two-coloring exists?
- Answer unknown until 2016

SAT Application: Coloring Pythagorean Triples

Pythagorean Triple (P-Triple)

- Positive integers a, b, c such that $a^2 + b^2 = c^2$
- E.g., $a = 3$, $b = 4$, $c = 5$.

Two-Coloring

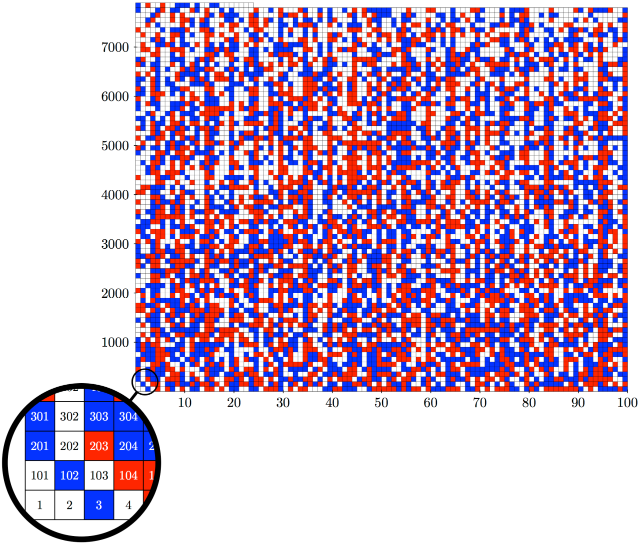
- For integers $i \in \{1, 2, \ldots, n\}$, assign $C_i \in \{\text{red}, \text{blue}\}$
- For every P-Triple a, b, c, cannot have $C_a = C_b = C_c$.

SAT Encoding $PTC(n)$

- n Boolean variables
- Variable $x_a = 1$ if a colored red, $= 0$ if colored blue
- Clauses for each P-Triple a, b, c, such that $1 \leq a < b < c \leq n$:

$x_a \vee x_b \vee x_c$	At least one colored red
$\overline{x}_a \vee \overline{x}_b \vee \overline{x}_c$	At least one colored blue

SAT Application: Coloring Pythagorean Triples, $n = 7824$

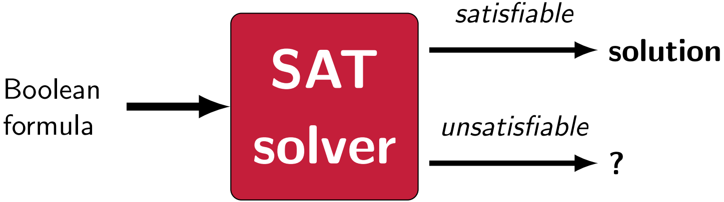


SAT Application: Coloring Pythagorean Triples, $n \geq 7825$

Formula $PTC(7825)$ **unsatisfiable**

- Heule, Kullmann, Marek, SAT 2016
- Partitioned into 10^6 subproblems
  - By enumerating assignments for some of the variables
- Ran on 800-core supercomputer for two days
- Generated 10^6 proofs of unsatisfiability
  - 200 Terabytes total
  - Validated with proof checker
  - A very long and very tedious collection of proofs!
- Unsatisfiability proof provides mathematical rigor

Boolean Satisfiability Solvers



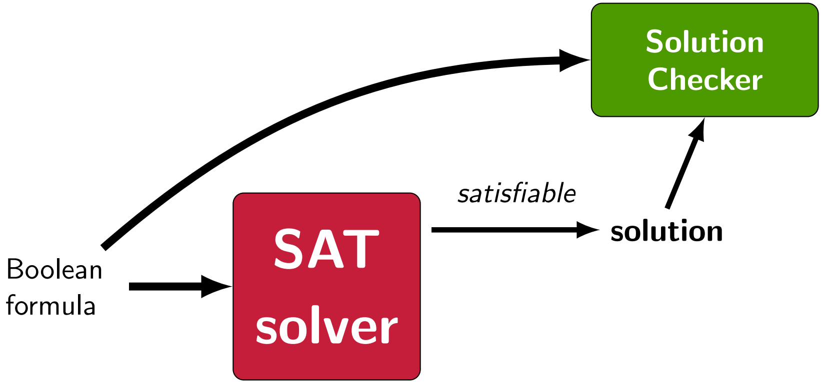
SAT Solvers Useful

- Optimization
- Formal verification
- Mathematical proofs

Can We Trust Them?

- No!
- Complex software
- e.g., KISSAT: 35K lines of code

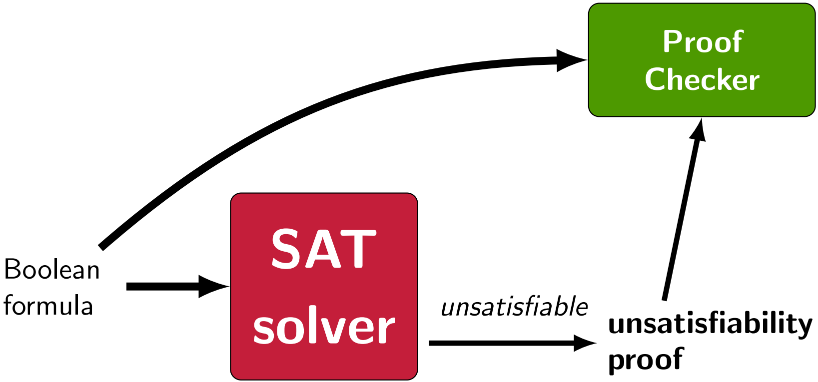
Boolean Satisfiability Solvers



SAT Solvers Useful

- Optimization
- Formal verification
- Mathematical proofs

Proof Generating Solvers



Unsatisfiability Proof

- Step-by-step proof in some logical framework

Proof Checker

- Simple program
- May be formally verified

Impact of Proof Checking

Adoption

- Required for SAT competition entrants since 2016

Benefits

- Can clearly judge competition submissions
- Developers have improved quality of their solvers
- Firm foundation for use in mathematical proofs

Unintended Consequences

- Narrowed focus to single SAT algorithm
  - Conflict-Driven Clause Learning (CDCL)
  - Search for solution, but learn conflicts
- Other powerful solution methods have languished.

Impact of Proof Checking

Adoption

- Required for SAT competition entrants since 2016

Benefits

- Can clearly judge competition submissions
- Developers have improved quality of their solvers
- Firm foundation for use in mathematical proofs

Unintended Consequences

- Narrowed focus to single SAT algorithm
  - Conflict-Driven Clause Learning (CDCL)
  - Search for solution, but learn conflicts
- Other powerful solution methods have languished.

My Long-Term Goals

- Enable proof generation for other SAT algorithms
- Develop checkable proof infrastructure for other domains

Conjunctive Normal Form (CNF) Formulas

Variables

- Input: $X = \{x_1, x_2, \ldots, x_n\}$
- Informally: $a, b, c, \ldots$

Literals

- Variable x
- Complemented variable $\overline{x}$.

Clauses

- $C = \{\ell_1, \ell_2, \ldots, \ell_k\}$ Set of literals
- $\overline{a} \vee b \vee \overline{c}$
- $\perp = \emptyset$ Empty clause (False)

Formula

- $\phi = \{C_1, C_2, \ldots, C_m\}$
- $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ Conjunction of clauses

Clausal Thinking

Useful tricks when writing CNF

Boolean Formula	CNF
$a \wedge b \rightarrow c$	$\overline{a} \vee \overline{b} \vee c$
$a \rightarrow b \vee c$	$\overline{a} \vee b \vee c$
$(a \vee b) \rightarrow c$	$(\overline{a} \vee c) \wedge (\overline{b} \vee c)$
$a \rightarrow (b \wedge c)$	$(\overline{a} \vee b) \wedge (\overline{a} \vee c)$
$ITE(a, b, c)$	$(\overline{a} \vee b) \wedge (a \vee c)$

- Advice: think in terms of implication.
- E.g., $ITE(a, b, c) = (a \rightarrow b) \wedge (\overline{a} \rightarrow c)$

Clausal Thinking: Parity Encodings

Boolean Formula	CNF	Explanation
$\mathit{OddParity}(a, b, c)$	$(\overline{a} \vee \overline{b} \vee c) \quad \wedge$ $(\overline{a} \vee b \vee \overline{c}) \quad \wedge$ $(a \vee \overline{b} \vee \overline{c}) \quad \wedge$ $(a \vee b \vee c)$	Even number of negations
$\mathit{EvenParity}(a, b, c)$	$(\overline{a} \vee \overline{b} \vee \overline{c}) \quad \wedge$ $(a \vee b \vee \overline{c}) \quad \wedge$ $(a \vee \overline{b} \vee c) \quad \wedge$ $(\overline{a} \vee b \vee c)$	Odd number of negations

Clausal Thinking: Parity Encodings

Boolean Formula	CNF
$\mathit{OddParity}(a, b, c)$	$(\overline{a} \vee \overline{b} \vee c) \quad \wedge$ $(\overline{a} \vee b \vee \overline{c}) \quad \wedge$ $(a \vee \overline{b} \vee \overline{c}) \quad \wedge$ $(a \vee b \vee c)$
$\mathit{OddParity}(a, b, c, d)$	$(\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{d}) \quad \wedge$ $(a \vee b \vee \overline{c} \vee \overline{d}) \quad \wedge$ $(a \vee \overline{b} \vee c \vee \overline{d}) \quad \wedge$ $(\overline{a} \vee b \vee c \vee \overline{d}) \quad \wedge$ $(\overline{a} \vee \overline{b} \vee c \vee d) \quad \wedge$ $(\overline{a} \vee b \vee \overline{c} \vee d) \quad \wedge$ $(a \vee \overline{b} \vee \overline{c} \vee d) \quad \wedge$ $(a \vee b \vee c \vee d)$

Parity Encoding with Intermediate Variables

Task

- Encode $OddParity(x_1, x_2, \ldots, x_n)$
- Direct encoding requires 2^{n-1} clauses
- All combinations with even number of negative literals

Decomposition

- Introduce new variable z
- Directly encode $EvenParity(x_1, x_2, z)$
- Recursively encode $OddParity(z, x_3, x_4, \ldots, x_n)$:
  - If $x_1 \oplus x_2 = 0$, then $z = 0$ and $OddParity(x_3, x_4, \ldots, x_n)$
  - If $x_1 \oplus x_2 = 1$, then $z = 1$ and $EvenParity(x_3, x_4, \ldots, x_n)$

Parity Encoding with Intermediate Variables

Decomposition

- Directly encode $\mathit{EvenParity}(x_1, x_2, z)$
- Recursively encode $\mathit{OddParity}(z, x_3, x_4, \ldots, x_n)$:

General Form

$$\begin{array}{rcl} z_2 & = & x_1 \oplus x_2 \\ z_3 & = & z_2 \oplus x_3 \\ & \cdots & \\ z_{n-2} & = & x_{n-2} \oplus x_{n-3} \\ z_{n-2} \oplus x_{n-1} \oplus x_n & = & 1 \end{array}$$

Complexity

- $n - 3$ additional variables
- $4(n - 2)$ clauses

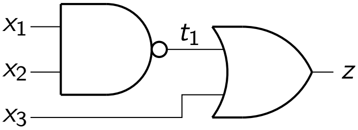
Clausal Thinking: Cardinality Constraints

$$a_1\, x_1 + a_2\, x_2 + \cdots + a_n\, x_n \quad \geq \quad t$$

Constraint	a_i	t
AtLeastOne	$\{0, 1\}$	1
AtMostOne	$\{0, -1\}$	-1

Boolean Formula	CNF
$\mathit{AtLeastOne}(a, b, c)$	$a \vee b \vee c$
$\mathit{AtMostOne}(a, b, c)$	$(\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee \overline{c}) \wedge (\overline{b} \vee \overline{c})$
$\mathit{AtMostOne}(a, b, c, d)$	$(\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee \overline{c}) \wedge (\overline{a} \vee \overline{d}) \quad \wedge$ $(\overline{b} \vee \overline{c}) \wedge (\overline{b} \vee \overline{d}) \wedge (\overline{c} \vee \overline{d})$

Encoding Arbitrary Formulas / Circuits



	Encode NAND gate	Encode OR gate
Formula	$\overline{t}_1 \leftrightarrow x_1 \wedge x_2$	$z \leftrightarrow t_1 \vee x_3$
Clauses	$t_1 \vee x_1$ $t_1 \vee x_2$ $\overline{t}_1 \vee \overline{x}_1 \vee \overline{x}_2$	$\overline{z} \vee t_1 \vee x_3$ $z \vee \overline{t}_1$ $z \vee \overline{x}_3$

Tseitin Encoding

- Introduce variables for intermediate values
- Linear complexity

Proof Rules: Resolution

- Robinson, 1965

$$\frac{\overline{a} \vee b \vee x \qquad \overline{x} \vee c \vee \overline{d}}{(\overline{a} \vee b) \vee (c \vee \overline{d})}$$

- Generalization of implication
- See https://en.wikipedia.org/wiki/Resolution_(logic)

Proof Rules: Resolution

- Robinson, 1965

$$(a \land \overline{b}) \rightarrow x \qquad\qquad\qquad\qquad x \rightarrow (c \lor \overline{d})$$

$$\frac{\overline{a} \lor b \lor x \qquad\qquad \overline{x} \lor c \lor \overline{d}}{(\overline{a} \lor b) \lor (c \lor \overline{d})}$$

$$(a \land \overline{b}) \rightarrow (c \lor \overline{d})$$

- Generalization of implication
- See https://en.wikipedia.org/wiki/Resolution_(logic)

Resolution Principle Nuances

OK To Have Repeated Literal

$$\frac{\overline{a} \vee b \vee x \qquad \overline{x} \vee b \vee \overline{d}}{\overline{a} \vee b \vee \overline{d}}$$

Not OK to Have Multiple Resolution Variables

$$\frac{\overline{a} \vee d \vee x \qquad \overline{x} \vee c \vee \overline{d}}{\top}$$

Proof Rules: Subsumption

$$\frac{\overline{a} \vee b \vee \overline{c}}{\overline{a} \vee b \vee \overline{c} \vee d}$$

- General Principle: $F \rightarrow F \vee d$

Example Formula

DIMACS Format

- Standard for all solvers
- Positive integers for variables
- Negative integers for their negations
- Lists terminated with `0`

ID	Clause	DIMACS Encoding
		`p cnf 4 6`
1	$\overline{a} \vee \overline{b} \vee \overline{c}$	`-1 -2 -3 0`
2	$\overline{a} \vee \overline{b} \vee c$	`-1 -2 3 0`
3	$a \vee \overline{d}$	`1 -4 0`
4	$a \vee d$	`1 4 0`
5	$b \vee \overline{d}$	`2 -4 0`
6	$b \vee d$	`2 4 0`

Example Proof

- Derive empty clause $\perp$ through set of resolution steps

$$\frac{a \lor d \qquad a \lor \overline{d}}{a \qquad\qquad a}$$

$$\frac{\overline{a} \lor \overline{b} \lor c \qquad a}{\overline{b} \lor c} \qquad\qquad \frac{a \qquad \overline{a} \lor \overline{b} \lor \overline{c}}{\overline{b} \lor \overline{c}}$$

$$\frac{\overline{b} \lor c \qquad \overline{b} \lor \overline{c}}{\overline{b}} \qquad\qquad \frac{b \lor \overline{d} \qquad b \lor d}{b}$$

$$\frac{\overline{b} \qquad b}{\perp}$$

But how can a program find such a proof?

Unit Propagation

Unit clauses

- If formula contains clause (x), then x must be assigned 1.
- If formula contains clause $(\overline{x})$, then x must be assigned 0.

Propagating Unit Literal ℓ

- If $\ell \in C$ and $\ell = 0$, then $C \leftarrow C - \{\ell\}$
- If $\ell \in C$ and $\ell = 1$, then C satisfied
- If any clause becomes unit, then iterate

Step	Formula	Units
1	$(\boldsymbol{a} \vee \overline{\boldsymbol{c}}) \wedge (\overline{\boldsymbol{a}} \vee \overline{\boldsymbol{b}} \vee \overline{\boldsymbol{c}}) \wedge (\boldsymbol{a} \vee \boldsymbol{b} \vee \boldsymbol{c}) \wedge (\boldsymbol{c})$	$c = 1$
2	$(\boldsymbol{a} \vee \overline{c}) \wedge (\overline{\boldsymbol{a}} \vee \overline{\boldsymbol{b}} \vee \overline{c}) \wedge (\boldsymbol{a} \vee \boldsymbol{b} \vee \boldsymbol{c}) \wedge (\boldsymbol{c})$	$a = 1$
3	$(\boldsymbol{a} \vee \overline{c}) \wedge (\overline{a} \vee \overline{\boldsymbol{b}} \vee \overline{c}) \wedge (\boldsymbol{a} \vee \boldsymbol{b} \vee \boldsymbol{c}) \wedge (\boldsymbol{c})$	$b = 0$
–	$(\boldsymbol{a} \vee \overline{c}) \wedge (\overline{a} \vee \overline{\boldsymbol{b}} \vee \overline{c}) \wedge (\boldsymbol{a} \vee b \vee \boldsymbol{c}) \wedge (\boldsymbol{c})$	

Basic CDCL Operation

Conflict-Driven Clause Learning

- Algorithm in state-of-the art solvers
- Search, but learn from dead ends

```
while(True):
    depth ⟵ 0
    while(True):
        UnitPropagate()
        if all clauses satisfied
            return solution
        if ConflictDetected():
            Generate conflict clause
            break
        Choose variable and assign 0 or 1
        depth ⟵ depth + 1
    if depth = 0:
        return UNSAT
```

CDCL Execution Example

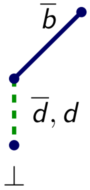
- No initial unit propagations

ID	Clause	UProp?
1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
2	$\overline{a} \vee \overline{b} \vee c$	
3	$a \vee \overline{d}$	
4	$a \vee d$	
5	$b \vee \overline{d}$	
6	$b \vee d$	

•

CDCL Execution Example

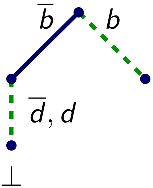
- Setting $b = 0$ causes conflict. Learn clause b.



ID	Clause	UProp?
1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
2	$\overline{a} \vee \overline{b} \vee c$	
3	$a \vee \overline{d}$	
4	$a \vee d$	
5	$b \vee \overline{d}$	*
6	$b \vee d$	*
7	b	

CDCL Execution Example

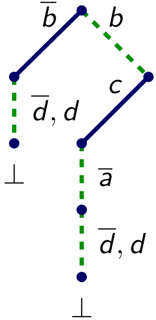
- Unit propagate b. No conflict



ID	Clause	UProp?
1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
2	$\overline{a} \vee \overline{b} \vee c$	
3	$a \vee \overline{d}$	
4	$a \vee d$	
5	$b \vee \overline{d}$	
6	$b \vee d$	
7	b	

CDCL Execution Example

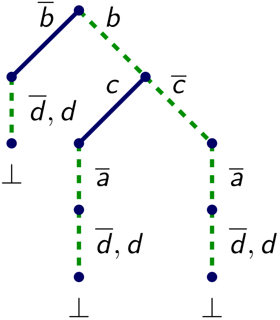
- Setting $c = 1$ causes conflict. Learn clause $\overline{c}$.



ID	Clause	UProp?
1	$\overline{a} \vee \overline{b} \vee \overline{c}$	*
2	$\overline{a} \vee \overline{b} \vee c$	
3	$a \vee \overline{d}$	*
4	$a \vee d$	*
5	$b \vee \overline{d}$	
6	$b \vee d$	
7	b	*
8	$\overline{c}$	

CDCL Execution Example

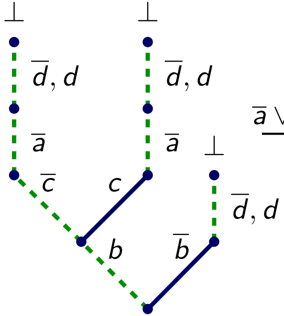
- Unit propagate b and $\overline{c}$. Causes conflict. UNSAT!



ID	Clause	UProp?
1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
2	$\overline{a} \vee \overline{b} \vee c$	*
3	$a \vee \overline{d}$	*
4	$a \vee d$	*
5	$b \vee \overline{d}$	
6	$b \vee d$	
7	b	*
8	$\overline{c}$	*
9	$\perp$	

Proof from CDCL Run

Proof Follows Branching Structure of CDCL



$$\dfrac{\dfrac{\dfrac{\overline{a}\lor\overline{b}\lor c \qquad a}{\overline{b}\lor c} \qquad \dfrac{a \qquad \overline{a}\lor\overline{b}\lor\overline{c}}{\overline{b}\lor\overline{c}}}{\overline{b}} \qquad \dfrac{b\lor\overline{d} \qquad b\lor d}{b}}{\bot}
\qquad\text{where}\qquad \dfrac{a\lor d \qquad a\lor\overline{d}}{a}$$

Reverse Unit Propagation (RUP)

Purpose

- Simple and efficient rule for use by proof checkers
- Good match to operation of CDCL solvers

Operation

- Each RUP application forms one step of unsatisfiability proof
- Performs a linear sequence of resolutions steps + subsumption

Objective

- $C = \{\ell_1, \ell_2, \ldots, \ell_m\}$ Clause to be added to proof
- $D_1, D_2, \ldots, D_k$ Previous clauses ("Antecedents")
- Prove: $D_1 \wedge D_2 \wedge \cdots \wedge D_k \rightarrow C$

Reverse Unit Propagation (RUP)

Objective

- $C = \{\ell_1, \ell_2, \ldots, \ell_m\}$ Clause to be added to proof
- $D_1, D_2, \ldots, D_k$ Previous clauses ("Antecedents")
- Prove: $D_1 \wedge D_2 \wedge \cdots \wedge D_k \rightarrow C$

Method

- Assume $\neg C = \overline{\ell}_1 \wedge \overline{\ell}_2 \wedge \cdots \wedge \overline{\ell}_m$
  - m unit clauses!
- Show contradiction with $D_1 \wedge D_2 \wedge \cdots \wedge D_k$
  - Accumulate unit clauses, starting with those for $\neg C$.
  - Accrue more unit clauses from $D_1, D_2, \ldots, D_{k-1}$.
  - When encounter D_k, should have contradiction

RUP Proof Example

ID	Clause	Antecedents
C_1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
C_2	$\overline{a} \vee \overline{b} \vee c$	
C_3	$a \vee \overline{d}$	
C_4	$a \vee d$	
C_5	$b \vee \overline{d}$	
C_6	$b \vee d$	

RUP Proof Example

ID	Clause	Antecedents
C_1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
C_2	$\overline{a} \vee \overline{b} \vee c$	
C_3	$a \vee \overline{d}$	
C_4	$a \vee d$	
C_5	$b \vee \overline{d}$	
C_6	$b \vee d$	
C_7	b	C_5, C_6

$$
\begin{array}{c|ccccc}
 & \overline{b} & & \overline{d} & & \perp \\
b & & C_5 & & C_6 &
\end{array}
$$

RUP Proof Example

ID	Clause	Antecedents
C_1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
C_2	$\overline{a} \vee \overline{b} \vee c$	
C_3	$a \vee \overline{d}$	
C_4	$a \vee d$	
C_5	$b \vee \overline{d}$	
C_6	$b \vee d$	
C_7	b	C_5, C_6
C_8	$\overline{c}$	C_7, C_1, C_3, C_4

$$\begin{array}{c|ccccccccc} & c & & b & & \overline{a} & & \overline{d} & & \bot \\ \overline{c} & & C_7 & & C_1 & & C_3 & & C_4 & \end{array}$$

RUP Proof Example

ID	Clause	Antecedents
C_1	$\overline{a} \lor \overline{b} \lor \overline{c}$	
C_2	$\overline{a} \lor \overline{b} \lor c$	
C_3	$a \lor \overline{d}$	
C_4	$a \lor d$	
C_5	$b \lor \overline{d}$	
C_6	$b \lor d$	
C_7	b	C_5, C_6
C_8	$\overline{c}$	C_7, C_1, C_3, C_4
C_9	$\bot$	C_7, C_8, C_2, C_3, C_4

$$\bot \;\Big|\; C_7 \xrightarrow{b} C_8 \xrightarrow{\overline{c}} C_2 \xrightarrow{\overline{a}} C_3 \xrightarrow{\overline{d}} C_4 \xrightarrow{\bot}$$

Proof File Examples

Proof

C_7	b	C_5, C_6
C_8	$\overline{c}$	C_7, C_1, C_3, C_4
C_9	$\perp$	C_7, C_8, C_2, C_3, C_4

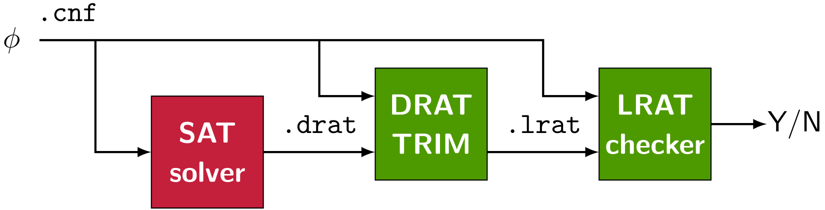
DRAT Proof File

```
2 0
-3 0
0
```

LRAT Proof File

`7`	`2 0`	`5 6 0`
`8`	`-3 0`	`7 1 3 4 0`
`9`	`0`	`7 8 2 3 4 0`

Proof Checking Infrastructure



Operation:

- DRAT-TRIM adds antecedents to proof steps
- LRAT checker simply checks each proof step

LRAT Checkers:

- LRAT-CHECK written in C.
  - Fast and high capacity
  - Designed to be simple enough to easily understand
- Formally verified ones. Built on ACL2, Coq, HOL, ...
  - Integrity not compromised if solver or DRAT-TRIM has bug

Resolution and CDCL

CDCL $\approx$ Resolution

- *Strength:* CDCL solver can readily generate resolution proofs
- *Weakness:* Lower bound on performance

Example: Pigeonhole Principle(PHP)

- Problem:
  - n holes, $n+1$ pigeons
  - Assign pigeons to holes:
    - Each pigeon is assigned to some hole
    - Each hole has at most one pigeon
- SAT Encoding:
  - Variables: $p_{i,j}$: Pigeon j in hole i. $1 \leq i \leq n$, $1 \leq j \leq n+1$.
  - $n+1$ at-least-one constraints
  - n at-most-one constraints
  - $O(n^3)$ total clauses
- PHP(n) resolution proofs are exponential in n [Haken, 1985]

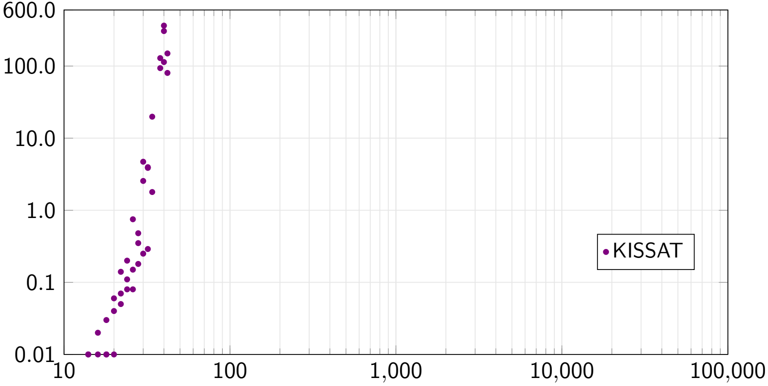
Parity Benchmark

- Chew and Heule, SAT 2020
- For random permtuation π:

$$\begin{array}{ccccccccccc} x_1 & \oplus & x_2 & \oplus & \cdots & \oplus & x_n & = & 1 & & \text{Odd parity} \\ x_{\pi(1)} & \oplus & x_{\pi(2)} & \oplus & \cdots & \oplus & x_{\pi(n)} & = & 0 & & \text{Even parity} \end{array}$$

- Conjunction unsatisfiable
- Very challenging for CDCL solvers
- Unit propagation of limited value
  - k-way parity constraint
  - Only propagate when $k-1$ variables assigned

Parity Benchmark Runtime



- KISSAT: State-of-the-art CDCL solver
- Tried 3-different seeds for each value of n
- Limited to $n \leq 42$ within 600 seconds

Extended Resolution

- Tseitin, 1967

Can introduce extension variables

- Variable z that has not yet occurred in proof
- Must add *defining* clauses
  - Encode constraint of form $z \leftrightarrow F$
  - Boolean formula z over input and earlier extension variables

Extension variable z becomes shorthand for formula F

- Repeated use can yield exponentially smaller proof

Similar to use of encoding variables in SAT formulas

- That's why they're called "Tseitin variables"
- But here they become part of proof, not of input formula

Extended RUP Proof Example

ID	Clause	Antecedents
C_1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
C_2	$\overline{a} \vee \overline{b} \vee c$	
C_3	$a \vee \overline{d}$	
C_4	$a \vee d$	
C_5	$b \vee \overline{d}$	
C_6	$b \vee d$	

Strategy

- Use z to encode $a \wedge b$.
- E.g., C_1 becomes $\overline{z} \vee \overline{c}$.

Extended RUP Proof Example

ID	Clause	Antecedents
C_1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
C_2	$\overline{a} \vee \overline{b} \vee c$	
C_3	$a \vee \overline{d}$	
C_4	$a \vee d$	
C_5	$b \vee \overline{d}$	
C_6	$b \vee d$	
C_7	$\overline{z} \vee a$	Defining Clauses
C_8	$\overline{z} \vee b$	
C_9	$z \vee \overline{a} \vee \overline{b}$	

Extended RUP Proof Example

ID	Clause	Antecedents
C_1	$\overline{a} \vee \overline{b} \vee \overline{c}$	
C_2	$\overline{a} \vee \overline{b} \vee c$	
C_3	$a \vee \overline{d}$	
C_4	$a \vee d$	
C_5	$b \vee \overline{d}$	
C_6	$b \vee d$	
C_7	$\overline{z} \vee a$	Defining Clauses
C_8	$\overline{z} \vee b$	
C_9	$z \vee \overline{a} \vee \overline{b}$	
C_{10}	$\overline{z} \vee \overline{c}$	C_7, C_8, C_1
C_{11}	$\overline{z}$	C_7, C_8, C_2, C_{10}
C_{12}	d	C_4, C_6, C_9, C_{11}
C_{13}	$\bot$	$C_{12}, C_3, C_5, C_9, C_{11}$

Can Extended Resolution Yield Faster SAT Solvers?

PHP Proof Complexity

- Exponential for ordinary resolution
- $O(n^4)$ for extended resolution [Cook, 1976]

Use in SAT?

- No clear way to choose which formulas to abbreviate
- No clear way to shorten search by using abbreviations