Trustworthy Boolean Reasoning
1A: (Un)Satisfiability

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

Are these functions equivalent?

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

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

Assume for `int`:

- 32-bit word
- Two's complement representation

SAT Application: Bit-Level Program Verification

Can this program call ERROR?

```
int abs_new(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 vn = abs_new(t);
  int vr = abs_ref(t);
  int err = (vn != vr);
  if (err != 0)
    ERROR();
}
```

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

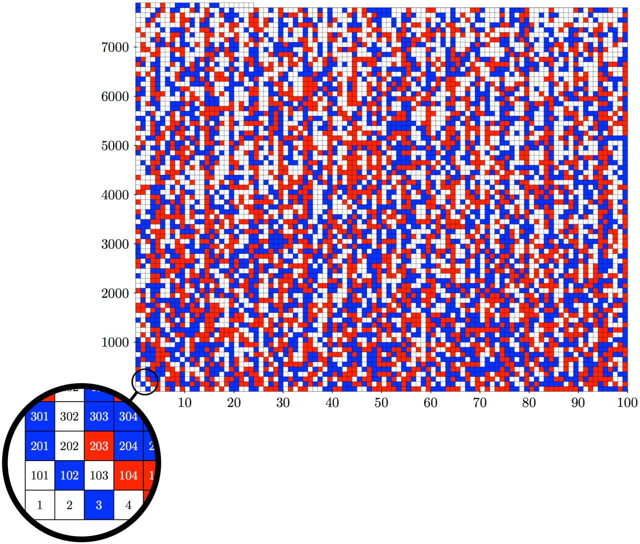
- For integers $i \in \{1, 2, \ldots, n\}$, assign $C_i \in \{\text{red}, \text{blue}\}$
- For every P-Triple a, b, c, cannot have $C_a = C_b = C_c$.

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, $n = 7824$

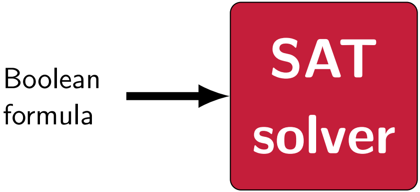


SAT Application: Coloring Pythagorean Triples, $n \geq 7825$

Formula $PTN(7825)$ **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

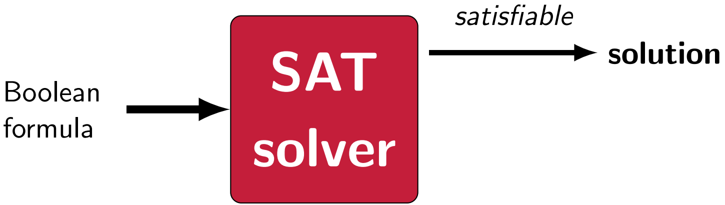
Boolean Satisfiability Solvers



SAT Solvers Useful

- Optimization
- Formal verification
- Mathematical proofs

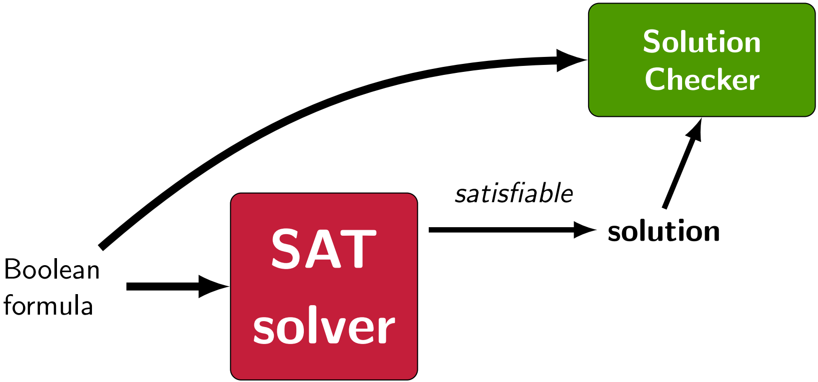
Boolean Satisfiability Solvers



SAT Solvers Useful

- Optimization
- Formal verification
- Mathematical proofs

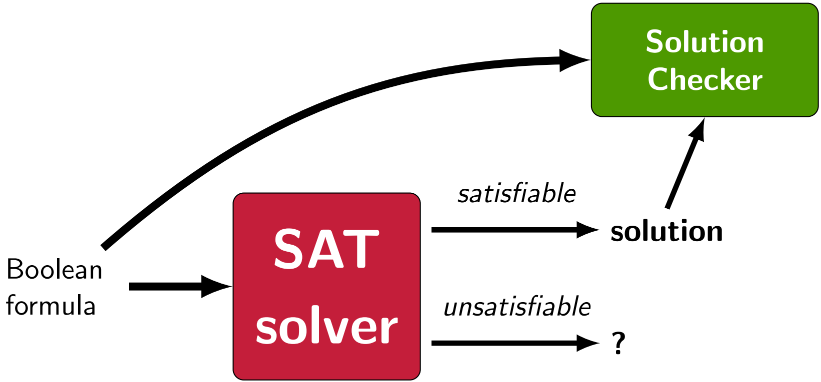
Boolean Satisfiability Solvers



SAT Solvers Useful

- Optimization
- Formal verification
- Mathematical proofs

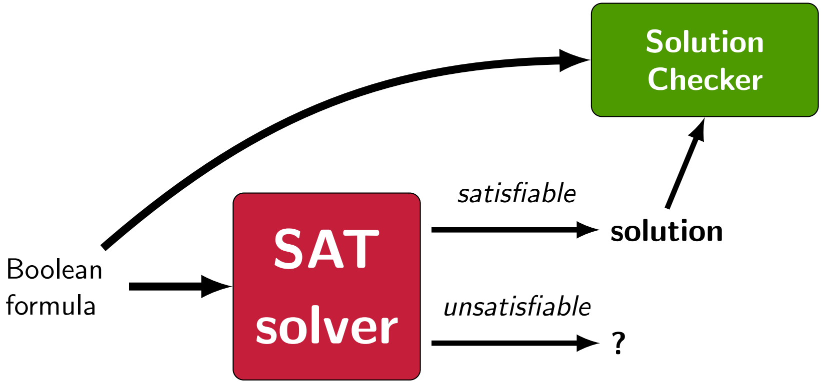
Boolean Satisfiability Solvers



SAT Solvers Useful

- Optimization
- Formal verification
- Mathematical proofs

Boolean Satisfiability Solvers



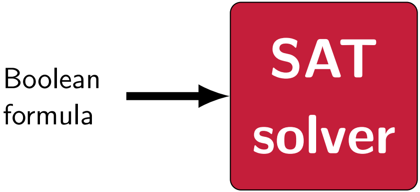
SAT Solvers Useful

- Optimization
- Formal verification
- Mathematical proofs

Can We Trust Them?

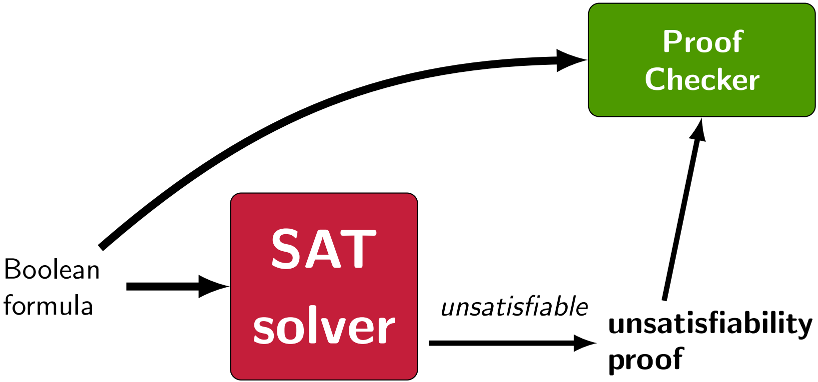
- No!
- Complex software
- e.g., KISSAT: 35K lines of code

Proof Generating Solvers

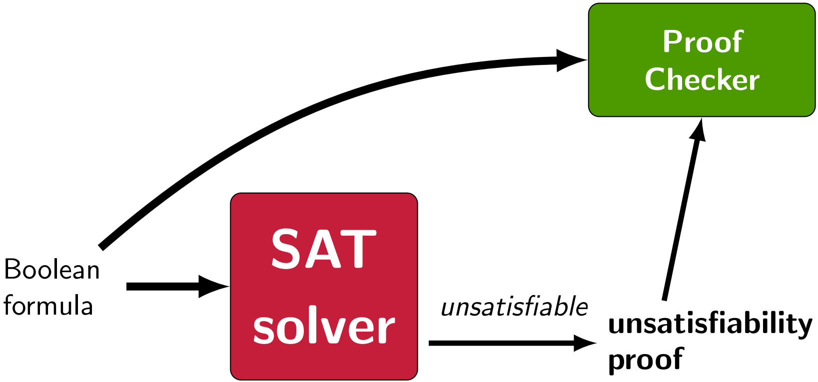


Proof Generating Solvers

Proof Generating Solvers



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

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
- $\overline{a} \vee b \vee \overline{c}$
- $\bot = \emptyset$ 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
$\mathit{OddParity}(a, b, c)$	$(\overline{a} \lor \overline{b} \lor c) \quad \land$ $(\overline{a} \lor b \lor \overline{c}) \quad \land$ $(a \lor \overline{b} \lor \overline{c}) \quad \land$ $(a \lor b \lor c)$
$\mathit{OddParity}(a, b, c, d)$	$(\overline{a} \lor \overline{b} \lor \overline{c} \lor \overline{d}) \quad \land$ $(a \lor b \lor \overline{c} \lor \overline{d}) \quad \land$ $(a \lor \overline{b} \lor c \lor \overline{d}) \quad \land$ $(\overline{a} \lor b \lor c \lor \overline{d}) \quad \land$ $(\overline{a} \lor \overline{b} \lor c \lor d) \quad \land$ $(\overline{a} \lor b \lor \overline{c} \lor d) \quad \land$ $(a \lor \overline{b} \lor \overline{c} \lor d) \quad \land$ $(a \lor b \lor c \lor d)$

Parity Encoding with Intermediate Variables

Task

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

Decomposition

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

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

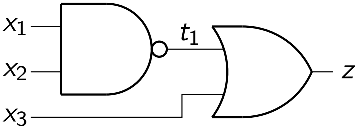
Clausal Thinking: Cardinality Constraints

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

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	$\overline{t}_1 \leftrightarrow x_1 \land x_2$	$z \leftrightarrow t_1 \lor x_3$
Clauses	$t_1 \lor x_1$ $t_1 \lor x_2$ $\overline{t}_1 \lor \overline{x}_1 \lor \overline{x}_2$	$\overline{z} \lor t_1 \lor x_3$ $z \lor \overline{t}_1$ $z \lor \overline{x}_3$

Tseitin Encoding

- Introduce variables for intermediate values
- Linear complexity

Proof Rules: Resolution

- Robinson, 1965

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

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

Proof Rules: Resolution

- Robinson, 1965

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

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

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

Proof Rules: Resolution

- Robinson, 1965

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

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

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

- 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

$$\dfrac{\dfrac{\dfrac{\overline{a} \vee \overline{b} \vee c \qquad \dfrac{a \vee d \qquad a \vee \overline{d}}{a}}{\overline{b} \vee c} \qquad \dfrac{a \qquad \overline{a} \vee \overline{b} \vee \overline{c}}{\overline{b} \vee \overline{c}}}{\overline{b}} \qquad \dfrac{b \vee \overline{d} \qquad b \vee d}{b}}{\perp}$$

But how can a program find such a proof?