

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 $i \in \{1, 2, \dots, n\}$, 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., $a = 3, b = 4, c = 5$.

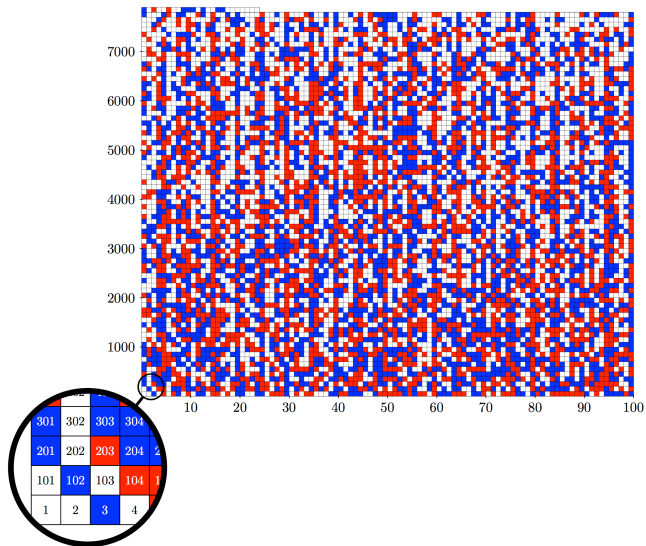
Two-Coloring

- ▶ For integers $i \in \{1, 2, \dots, n\}$, 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
 - $\bar{x}_a \vee \bar{x}_b \vee \bar{x}_c$ At least one colored blue

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

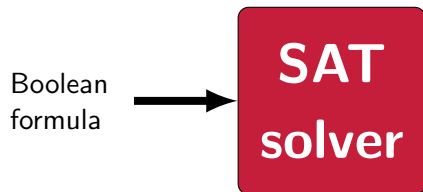


SAT Application: Coloring Pythagorean Triples, $n \geq 7825$

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

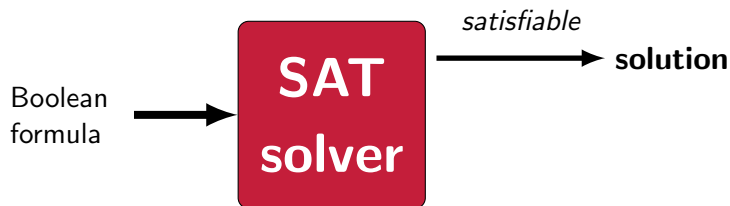
Boolean Satisfiability Solvers



SAT Solvers Useful

- ▶ Optimization
- ▶ Formal verification
- ▶ Mathematical proofs

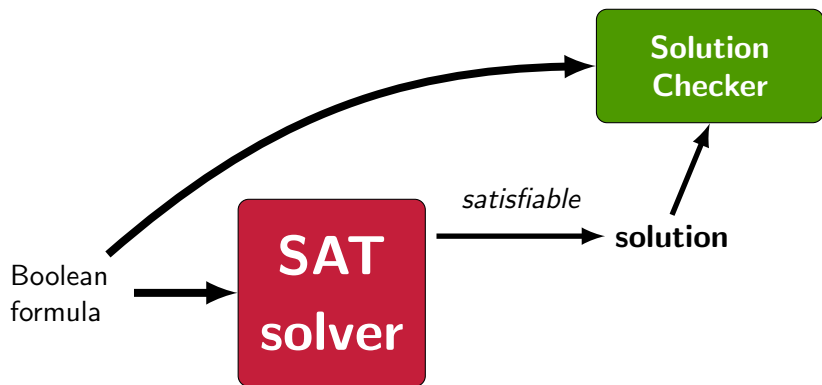
Boolean Satisfiability Solvers



SAT Solvers Useful

- ▶ Optimization
- ▶ Formal verification
- ▶ Mathematical proofs

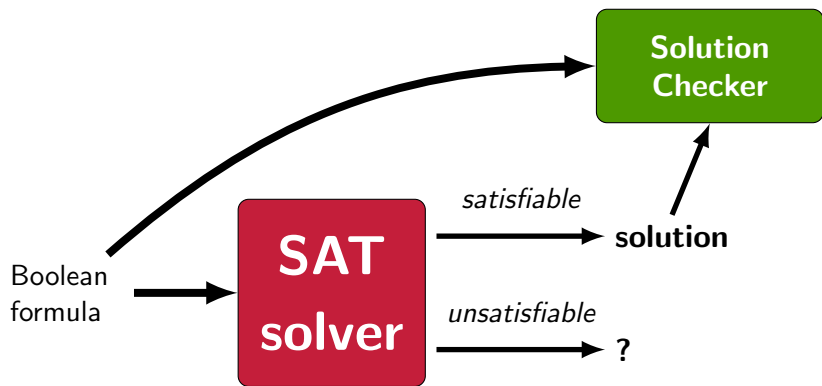
Boolean Satisfiability Solvers



SAT Solvers Useful

- ▶ Optimization
- ▶ Formal verification
- ▶ Mathematical proofs

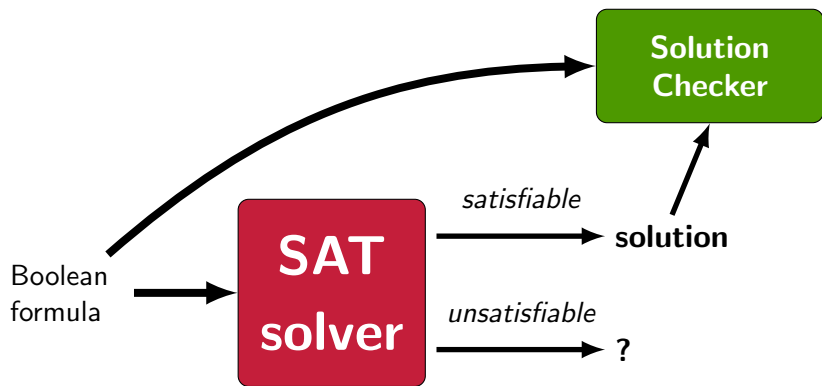
Boolean Satisfiability Solvers



SAT Solvers Useful

- ▶ Optimization
- ▶ Formal verification
- ▶ Mathematical proofs

Boolean Satisfiability Solvers



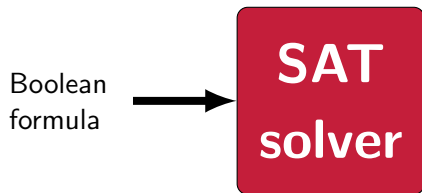
SAT Solvers Useful

- ▶ Optimization
- ▶ Formal verification
- ▶ Mathematical proofs

Can We Trust Them?

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

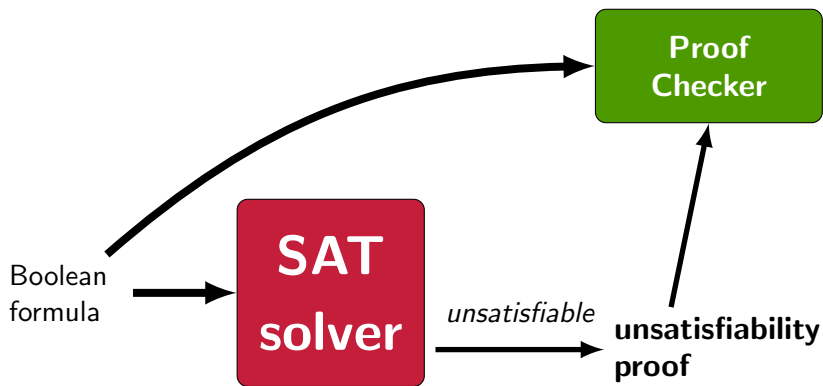
Proof Generating Solvers



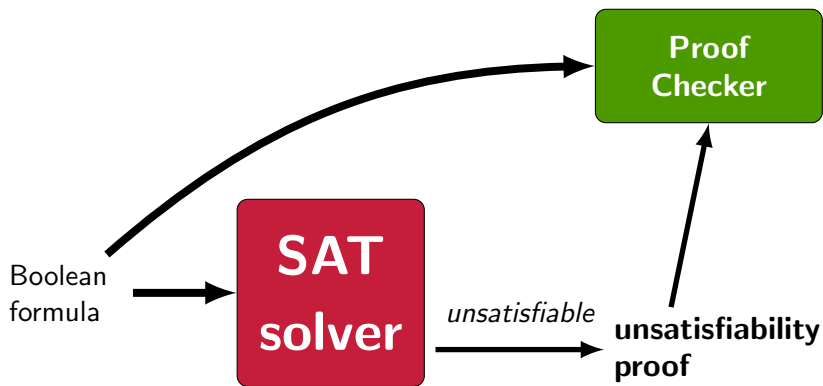
Proof Generating Solvers



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

Literals

- ▶ Variable x
- ▶ Complemented variable \bar{x} .

Clauses

- ▶ $C = \{l_1, l_2, \dots, l_k\}$ Set of literals
- ▶ $\bar{a} \vee b \vee \bar{c}$
- ▶ $\perp = \emptyset$ Empty clause (False)

Formula

- ▶ $\phi = \{C_1, C_2, \dots, C_m\}$
- ▶ $C_1 \wedge C_2 \wedge \dots \wedge C_m$ Conjunction of clauses

Clausal Thinking

Useful tricks when writing CNF

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

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

Clausal Thinking: Parity Encodings

Boolean Formula	CNF	Explanation
<i>OddParity</i> (a, b, c)	$(\bar{a} \vee \bar{b} \vee c)$	Even number of negations
	$(\bar{a} \vee b \vee \bar{c})$	
	$(a \vee \bar{b} \vee \bar{c})$	
	$(a \vee b \vee c)$	
<i>EvenParity</i> (a, b, c)	$(\bar{a} \vee \bar{b} \vee \bar{c})$	Odd number of negations
	$(a \vee b \vee \bar{c})$	
	$(a \vee \bar{b} \vee c)$	
	$(\bar{a} \vee b \vee c)$	

Clausal Thinking: Parity Encodings

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

Parity Encoding with Intermediate Variables

Task

- ▶ Encode $OddParity(x_1, x_2, \dots, x_n)$
- ▶ 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 $OddParity(z, x_3, x_4, \dots, x_n)$:
 - ▶ If $x_1 \oplus x_2 = 0$, then $z = 0$ and $OddParity(x_3, x_4, \dots, x_n)$
 - ▶ If $x_1 \oplus x_2 = 1$, then $z = 1$ and $EvenParity(x_3, x_4, \dots, x_n)$

Parity Encoding with Intermediate Variables

Decomposition

- ▶ Directly encode $EvenParity(x_1, x_2, z)$
- ▶ Recursively encode $OddParity(z, x_3, x_4, \dots, x_n)$:

General Form

$$z_2 = x_1 \oplus x_2$$

$$z_3 = z_2 \oplus x_3$$

...

$$z_{n-2} = x_{n-2} \oplus x_{n-3}$$

$$z_{n-2} \oplus x_{n-1} \oplus x_n = 1$$

Complexity

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

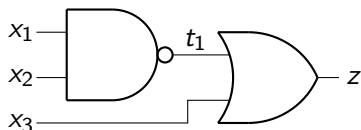
Clausal Thinking: Cardinality Constraints

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

Constraint	a_i	t
<i>AtLeastOne</i>	$\{0, 1\}$	1
<i>AtMostOne</i>	$\{0, -1\}$	-1

Boolean Formula	CNF
<i>AtLeastOne</i> (a, b, c)	$a \vee b \vee c$
<i>AtMostOne</i> (a, b, c)	$(\bar{a} \vee \bar{b}) \wedge (\bar{a} \vee \bar{c}) \wedge (\bar{b} \vee \bar{c})$
<i>AtMostOne</i> (a, b, c, d)	$(\bar{a} \vee \bar{b}) \wedge (\bar{a} \vee \bar{c}) \wedge (\bar{a} \vee \bar{d}) \wedge (\bar{b} \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (\bar{c} \vee \bar{d})$

Encoding Arbitrary Formulas / Circuits



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

Tseitin Encoding

- ▶ Introduce variables for intermediate values
- ▶ Linear complexity

Proof Rules: Resolution

- ▶ Robinson, 1965

$$\frac{\bar{a} \vee b \vee x \quad \bar{x} \vee c \vee \bar{d}}{(\bar{a} \vee b) \vee (c \vee \bar{d})}$$

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

Proof Rules: Resolution

- ▶ Robinson, 1965

$$(a \wedge \bar{b}) \rightarrow x$$

$$x \rightarrow (c \vee \bar{d})$$

$$\frac{\bar{a} \vee b \vee x \quad \bar{x} \vee c \vee \bar{d}}{(\bar{a} \vee b) \vee (c \vee \bar{d})}$$

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

Proof Rules: Resolution

- ▶ Robinson, 1965

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

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

$$(a \wedge \bar{b}) \rightarrow (c \vee \bar{d})$$

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

Resolution Principle Nuances

OK To Have Repeated Literal

$$\frac{\bar{a} \vee b \vee x \quad \bar{x} \vee b \vee \bar{d}}{\bar{a} \vee b \vee \bar{d}}$$

Not OK to Have Multiple Resolution Variables

$$\frac{\bar{a} \vee d \vee x \quad \bar{x} \vee c \vee \bar{d}}{\top}$$

Proof Rules: Subsumption

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

- ▶ 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	$\bar{a} \vee \bar{b} \vee \bar{c}$	-1 -2 -3 0
2	$\bar{a} \vee \bar{b} \vee c$	-1 -2 3 0
3	$a \vee \bar{d}$	1 -4 0
4	$a \vee d$	1 4 0
5	$b \vee \bar{d}$	2 -4 0
6	$b \vee d$	2 4 0

Example Proof

- Derive empty clause \perp through set of resolution steps

$$\begin{array}{c} \frac{\bar{a} \vee \bar{b} \vee c}{\bar{b} \vee c} \quad \frac{\frac{a \vee d \quad a \vee \bar{d}}{a} \quad \bar{a} \vee \bar{b} \vee \bar{c}}{\bar{b} \vee \bar{c}} \quad \frac{b \vee \bar{d} \quad b \vee d}{b} \\ \frac{\bar{b} \quad \bar{b}}{\perp} \end{array}$$

But how can a program find such a proof?