Proofs of Unsatisfiability

Marijn J.H. Heule

THE UNIVERSITY OF
TEXAS
AT AUSTIN

SAT 2016 Industry Day
July 9, 2016
Outline

Introduction

Proof Checking

Proof Systems and Formats

Media and Applications

Conclusions
Introduction
Satisfiability (SAT) solving has many applications:

- Formal verification
- Graph theory
- Bioinformatics
- Train safety
- Planning
- Number theory
- Cryptography
- Rewrite termination

Encode → SAT solver → Decode

00101000101
11010101010
1010101
01010
10101010101
A Small Satisfiability (SAT) Problem

\[(x_5 \lor x_8 \lor \bar{x}_2) \land (x_2 \lor \bar{x}_1 \lor \bar{x}_3) \land (\bar{x}_8 \lor \bar{x}_3 \lor \bar{x}_7) \land (\bar{x}_5 \lor \bar{x}_3 \lor x_8) \land\]
\[(\bar{x}_6 \lor \bar{x}_1 \lor \bar{x}_5) \land (x_8 \lor \bar{x}_9 \lor x_3) \land (x_2 \lor x_1 \lor x_3) \land (\bar{x}_1 \lor x_8 \lor x_4) \land\]
\[(\bar{x}_9 \lor \bar{x}_6 \lor x_8) \land (x_8 \lor x_3 \lor \bar{x}_9) \land (x_9 \lor \bar{x}_3 \lor x_8) \land (x_6 \lor \bar{x}_9 \lor x_5) \land\]
\[(x_2 \lor \bar{x}_3 \lor \bar{x}_8) \land (x_8 \lor \bar{x}_6 \lor \bar{x}_3) \land (x_8 \lor \bar{x}_3 \lor \bar{x}_1) \land (\bar{x}_8 \lor x_6 \lor \bar{x}_2) \land\]
\[(x_7 \lor x_9 \lor \bar{x}_2) \land (x_8 \lor \bar{x}_9 \lor x_2) \land (\bar{x}_1 \lor \bar{x}_9 \lor x_4) \land (x_8 \lor x_1 \lor \bar{x}_2) \land\]
\[(x_3 \lor \bar{x}_4 \lor \bar{x}_6) \land (\bar{x}_1 \lor \bar{x}_7 \lor x_5) \land (\bar{x}_7 \lor x_1 \lor x_6) \land (\bar{x}_5 \lor x_4 \lor \bar{x}_6) \land\]
\[(\bar{x}_4 \lor x_9 \lor \bar{x}_8) \land (x_2 \lor x_9 \lor x_1) \land (x_5 \lor \bar{x}_7 \lor x_1) \land (\bar{x}_7 \lor \bar{x}_9 \lor \bar{x}_6) \land\]
\[(x_2 \lor x_5 \lor x_4) \land (x_8 \lor \bar{x}_4 \lor x_5) \land (x_5 \lor x_9 \lor x_3) \land (\bar{x}_5 \lor \bar{x}_7 \lor x_9) \land\]
\[(x_2 \lor \bar{x}_8 \lor x_1) \land (\bar{x}_7 \lor x_1 \lor x_5) \land (x_1 \lor x_4 \lor x_3) \land (x_1 \lor \bar{x}_9 \lor \bar{x}_4) \land\]
\[(x_3 \lor x_5 \lor x_6) \land (\bar{x}_6 \lor x_3 \lor \bar{x}_9) \land (\bar{x}_7 \lor x_5 \lor x_9) \land (x_7 \lor \bar{x}_5 \lor \bar{x}_2) \land\]
\[(x_4 \lor x_7 \lor x_3) \land (x_4 \lor \bar{x}_9 \lor \bar{x}_7) \land (x_5 \lor \bar{x}_1 \lor x_7) \land (x_5 \lor \bar{x}_1 \lor x_7) \land\]
\[(x_6 \lor x_7 \lor \bar{x}_3) \land (\bar{x}_8 \lor \bar{x}_6 \lor \bar{x}_7) \land (x_6 \lor x_2 \lor x_3) \land (\bar{x}_8 \lor x_2 \lor x_5) \land\]

Does there exist an assignment satisfying all clauses?
Search for a satisfying assignment (or proof none exists)

\[
(x_5 \lor x_8 \lor \overline{x}_2) \land (x_2 \lor \overline{x}_1 \lor \overline{x}_3) \land (x_8 \lor \overline{x}_3 \lor \overline{x}_7) \land (\overline{x}_5 \lor x_3 \lor x_8) \\
(\overline{x}_6 \lor \overline{x}_1 \lor \overline{x}_5) \land (x_8 \lor \overline{x}_9 \lor x_3) \land (x_2 \lor x_1 \lor x_3) \land (\overline{x}_1 \lor x_8 \lor x_4) \\
(\overline{x}_9 \lor \overline{x}_6 \lor x_8) \land (x_8 \lor x_3 \lor \overline{x}_9) \land (x_9 \lor \overline{x}_3 \lor x_8) \land (x_6 \lor \overline{x}_9 \lor x_5) \\
(x_2 \lor \overline{x}_3 \lor \overline{x}_8) \land (x_8 \lor \overline{x}_6 \lor \overline{x}_3) \land (x_8 \lor \overline{x}_3 \lor \overline{x}_1) \land (\overline{x}_8 \lor x_6 \lor \overline{x}_2) \\
(x_7 \lor x_9 \lor \overline{x}_2) \land (x_8 \lor \overline{x}_9 \lor x_2) \land (\overline{x}_1 \lor \overline{x}_9 \lor x_4) \land (x_8 \lor x_1 \lor \overline{x}_2) \\
(x_3 \lor \overline{x}_4 \lor \overline{x}_6) \land (\overline{x}_1 \lor \overline{x}_7 \lor x_5) \land (\overline{x}_7 \lor x_1 \lor x_6) \land (\overline{x}_5 \lor x_4 \lor \overline{x}_6) \\
(\overline{x}_4 \lor x_9 \lor \overline{x}_8) \land (x_2 \lor x_9 \lor x_1) \land (x_5 \lor \overline{x}_7 \lor x_1) \land (\overline{x}_7 \lor \overline{x}_9 \lor \overline{x}_6) \\
(x_2 \lor x_5 \lor x_4) \land (x_8 \lor \overline{x}_4 \lor x_5) \land (x_5 \lor x_9 \lor x_3) \land (\overline{x}_5 \lor \overline{x}_7 \lor x_9) \\
(x_2 \lor \overline{x}_8 \lor x_1) \land (\overline{x}_7 \lor x_1 \lor x_5) \land (x_1 \lor x_4 \lor x_3) \land (x_1 \lor \overline{x}_9 \lor \overline{x}_4) \\
(x_3 \lor x_5 \lor x_6) \land (\overline{x}_6 \lor x_3 \lor \overline{x}_9) \land (\overline{x}_7 \lor x_5 \lor x_9) \land (x_7 \lor \overline{x}_5 \lor \overline{x}_2) \\
(x_4 \lor x_7 \lor x_3) \land (x_4 \lor \overline{x}_9 \lor \overline{x}_7) \land (x_5 \lor \overline{x}_1 \lor x_7) \land (x_5 \lor \overline{x}_1 \lor x_7) \\
(x_6 \lor x_7 \lor \overline{x}_3) \land (\overline{x}_8 \lor \overline{x}_6 \lor \overline{x}_7) \land (x_6 \lor x_2 \lor x_3) \land (\overline{x}_8 \lor x_2 \lor x_5)
\]

Solutions are easy to verify, but what about unsatisfiability?
Original 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.
Demo: Validating Solver Output
Proof Checking
Resolution Rule and Resolution Chains

Resolution Rule

\[
\frac{(x \lor a_1 \lor \ldots \lor a_i) \quad (\bar{x} \lor b_1 \lor \ldots \lor b_j)}{(a_1 \lor \ldots \lor a_i \lor b_1 \lor \ldots \lor b_j)}
\]

- Many SAT techniques can be simulated by resolution.
Resolution Rule and Resolution Chains

Resolution Rule

\[
(x \lor a_1 \lor \ldots \lor a_i) \quad (\bar{x} \lor b_1 \lor \ldots \lor b_j)
\]
\[
\quad \Rightarrow (a_1 \lor \ldots \lor a_i \lor b_1 \lor \ldots \lor b_j)
\]

- Many SAT techniques can be simulated by resolution.

A resolution chain is a sequence of resolution steps. The resolution steps are performed from left to right.

Example

- \((c) := (\bar{a} \lor \bar{b} \lor c) \diamond (\bar{a} \lor b) \diamond (a \lor c)\)
- \((\bar{a} \lor c) := (\bar{a} \lor b) \diamond (a \lor c) \diamond (\bar{a} \lor \bar{b} \lor c)\)
- The order of the clauses in the chain matter
Resolution Proofs versus Clausal Proofs

Consider the formula $F := (\overline{b} \lor c) \land (a \lor c) \land (\overline{a} \lor b) \land (\overline{a} \lor \overline{b}) \land (a \lor \overline{b}) \land (b \lor \overline{c})$

A resolution graph of $F$ is:

A resolution proof consists of all nodes and edges of the resolution graph
- Graphs from SAT solvers have $\sim 400$ incoming edges per node
- Resolution proof logging can heavily increase memory usage ($\times 100$)

A clausal proof is a list of all nodes sorted by topological order
- Clausal proofs are easy to emit and relatively small
- Clausal proof checking requires to reconstruct the edges (costly)
Clausal Proof: Checker has to reconstruct resolution edges

\[ \overline{b} \lor c \lor a \lor c \lor \overline{a} \lor \overline{b} \lor \overline{a} \lor \overline{b} \lor \overline{c} \]
Clausal Proof: Checker has to reconstruct resolution edges

\[ \bar{b} \lor c \lor \bar{a} \lor \bar{b} \lor \bar{a} \lor b \lor \bar{b} \lor \bar{a} \lor c \]

Diagram:

- Box: \( \bar{b} \)
- \( \bar{a} \)
- \( c \)
- \( \epsilon \)
- \( \bar{b} \lor c \)
- \( a \lor c \)
- \( \bar{a} \lor b \)
- \( \bar{a} \lor \bar{b} \)
- \( a \lor \bar{b} \)
- \( b \lor \bar{c} \)
Clausal Proof: Checker has to reconstruct resolution edges

\( \overline{b} \)  

\( \overline{a} \)  

\( c \)  

\( \varepsilon \)  

\( \overline{b} \lor c \)  

\( a \lor c \)  

\( \overline{a} \lor b \)  

\( \overline{a} \lor \overline{b} \)  

\( a \lor \overline{b} \)  

\( b \lor \overline{c} \)
Clausal Proof: Checker has to reconstruct resolution edges
Clausal Proof: Checker has to reconstruct resolution edges
Improvement I: Backwards Checking

Goldberg and Novikov proposed checking the refutation backwards [DATE 2003]:

- start by validating the empty clause;
- mark all lemmas using conflict analysis;
- only validate marked lemmas.

Advantage: validate fewer lemmas.

Disadvantage: more complex.

We provide a fast open source implementation of this procedure.
Improvement II: Clause Deletion

We proposed to extend clausal proofs with deletion information [STVR 2014]:

- clause deletion is crucial for efficient solving;
- emit learning and deletion information;
- proof size might double;
- checking speed can be reduced significantly.

Clause deletion can be combined with backwards checking [FMCAD 2013]:

- ignore deleted clauses earlier in the proof;
- optimize clause deletion for trimmed proofs.
Improvement III: Core-first Unit Propagation

We propose a new unit propagation variant:
1. propagate using clauses already in the core;
2. examine non-core clauses only at fixpoint;
3. if a non-core unit clause is found, goto 1);
4. otherwise terminate.

Our variant, called Core-first Unit Propagation, can reduce checking costs considerably.

Also, the resulting core and proof are smaller.
Checking: Backwards + Core-first + Deletion

Core-first unit propagation results in smaller cores and proofs
Checking: Backwards + Core-first + Deletion

Core-first unit propagation results in smaller cores and proofs
Checking: Backwards + Core-first + Deletion

Core-first unit propagation results in smaller cores and proofs
Checking: Backwards + Core-first + Deletion

Core-first unit propagation results in smaller cores and proofs
Checking: Backwards + Core-first + Deletion

Core-first unit propagation results in smaller cores and proofs
Checking: Backwards + Core-first + Deletion

Core-first unit propagation results in smaller cores and proofs
Checking: Backwards + Core-first + Deletion

Core-first unit propagation results in smaller cores and proofs
Checking: Backwards + Core-first + Deletion

Core-first unit propagation results in smaller cores and proofs
Proof Systems Formats
Clausal Proof System [Järvisalo, Heule, and Biere 2012]

Learn: add a clause
* Preserve satisfiability

Unsatisfiable
* Learn empty clause

Satisfiable
* Forget last clause

Forget: remove a clause
* Preserve unsatisfiability
Ideal Properties of a Proof System for SAT Solvers

- **Easy to Emit**
  - Resolution Proofs
    - Zhang and Malik, 2003
    - Van Gelder, 2008; Biere, 2008
  - Clausal Proofs
    - Goldberg and Novikov, 2003
    - Van Gelder, 2008

- **Compact**
  - 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]

- **Checked Efficiently**
  - DRAT proofs (RAT + deletion)
    - Wetzler, Heule, and Hunt, Jr. [SAT 2014]
Ideal Properties of a Proof System for SAT Solvers

- **Easy to Emit**
  - Resolution Proofs
    - Zhang and Malik, 2003
    - Van Gelder, 2008; Biere, 2008

- **Compact**
  - Clausal Proofs
    - Goldberg and Novikov, 2003
    - Van Gelder, 2008

- **Checked Efficiently**
  - Clausal proofs + clause deletion
    - Heule, Hunt, Jr., and Wetzler [STVR 2014]
  - Optimized clausal proof checker
    - Heule, Hunt, Jr., and Wetzler [FMCAD 2013]

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

- **Verified**
  - DRAT proofs (RAT + deletion)
    - Wetzler, Heule, and Hunt, Jr. [SAT 2014]
Proof Formats: The Input Format DIMACS

$$E := (\bar{b} \lor c) \land (a \lor c) \land (\bar{a} \lor b) \land (\bar{a} \lor \bar{b}) \land (a \lor \bar{b}) \land (b \lor \bar{c})$$

The input format of SAT solvers is known as **DIMACS**

- header starts with `p cnf` followed by the number of variables ($n$) and the number of clauses ($m$)
- the next $m$ lines represent the clauses
- positive literals are positive numbers
- negative literals are negative numbers
- clauses are terminated with a 0

Most proof formats use a similar syntax.
Proof Formats: TraceCheck Overview

TraceCheck is the most popular resolution-style format.

\[ E := (\overline{b} \lor c) \land (a \lor c) \land (\overline{a} \lor b) \land (\overline{a} \lor \overline{b}) \land (a \lor \overline{b}) \land (b \lor \overline{c}) \]

TraceCheck is readable and resolution chains make it relatively compact

\[
\langle \text{trace} \rangle = \{ \langle \text{clause} \rangle \} \\
\langle \text{clause} \rangle = \langle \text{pos} \rangle \langle \text{literals} \rangle \langle \text{antecedents} \rangle \\
\langle \text{literals} \rangle = “*” | \{ \langle \text{lit} \rangle \} “0” \\
\langle \text{antecedents} \rangle = \{ \langle \text{pos} \rangle \} “0” \\
\langle \text{lit} \rangle = \langle \text{pos} \rangle | \langle \text{neg} \rangle \\
\langle \text{pos} \rangle = “1” | “2” | \cdots | \langle \text{max-idx} \rangle \\
\langle \text{neg} \rangle = “-“ \langle \text{pos} \rangle
\]

\[
\begin{array}{cccccc}
1 & -2 & 3 & 0 & 0 \\
2 & 1 & 3 & 0 & 0 \\
3 & -1 & 2 & 0 & 0 \\
4 & -1 & -2 & 0 & 0 \\
5 & 1 & -2 & 0 & 0 \\
6 & 2 & -3 & 0 & 0 \\
7 & -2 & 0 & 4 & 5 & 0 \\
8 & 3 & 0 & 1 & 2 & 3 & 0 \\
9 & 0 & 6 & 7 & 8 & 0
\end{array}
\]
Proof Formats: TraceCheck Examples

**TraceCheck** is the most popular resolution-style format.

\[ E := (\overline{b} \lor c) \land (a \lor c) \land (\overline{a} \lor b) \land (\overline{a} \lor \overline{b}) \land (a \lor \overline{b}) \land (b \lor \overline{c}) \]

TraceCheck is readable and resolution chains make it relatively compact.

The clauses 1 to 6 are input clauses.

Clause 7 is the resolvent 4 and 5:

- \((\overline{b}) := (\overline{a} \lor \overline{b}) \diamond (a \lor \overline{b})\)

Clause 8 is the resolvent 1, 2 and 3:

- \((c) := (\overline{b} \lor c) \diamond (\overline{a} \lor b) \diamond (a \lor c)\)
- **NB:** the antecedents are swapped!

Clause 9 is the resolvent 6, 7 and 8:

- \(\epsilon := (b \lor \overline{c}) \diamond (\overline{b}) \diamond (c)\)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>-3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>-2</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Proof Formats: TraceCheck Don’t Cares

Support for unsorted clauses, unsorted antecedents and omitted literals.

- Clauses are not required to be sorted based on the clause index

\[
\begin{array}{ccc}
8 & 3 & 0 \\
7 & -2 & 0 \\
\end{array}
\equiv
\begin{array}{ccc}
7 & -2 & 0 \\
8 & 3 & 0 \\
\end{array}
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 0 \\
\end{array}
\]

- The antecedents of a clause can be in arbitrary order

\[
\begin{array}{ccc}
7 & -2 & 0 \\
8 & 3 & 0 \\
\end{array}
\equiv
\begin{array}{ccc}
7 & -2 & 0 \\
8 & 3 & 0 \\
\end{array}
\begin{array}{ccc}
5 & 4 & 0 \\
3 & 1 & 2 \\
\end{array}
\]

- For learned clauses, the literals can be omitted using *

\[
\begin{array}{ccc}
7 & * & 5 \\
8 & * & 3 \\
\end{array}
\equiv
\begin{array}{ccc}
7 & -2 & 0 \\
8 & 3 & 0 \\
\end{array}
\begin{array}{ccc}
4 & 5 & 0 \\
1 & 2 & 3 \\
\end{array}
\]
Demo: Clausal Proof to TraceCheck
Proof Formats: Reverse Unit Propagation (RUP)

Unit Propagation
Given an assignment \( \varphi \), extend it by making unit clauses true — until fixpoint or a clause becomes false

Reverse Unit Propagation (RUP)
A clause \( C = (l_1 \lor l_2 \lor \cdots \lor l_k) \) has reverse unit propagation w.r.t. formula \( F \) if unit propagation of the assignment \( \varphi = \bar{C} = (\bar{l}_1 \land \bar{l}_2 \land \cdots \land \bar{l}_k) \) on \( F \) results in a conflict.
We write: \( F \land \bar{C} \vdash_1 \epsilon \)

A clause sequence \( C_1, \ldots, C_m \) is a RUP proof for formula \( F \)
- \( F \land C_1 \land \cdots \land C_{i-1} \land \bar{C}_i \vdash_1 \epsilon \)
- \( C_m = \epsilon \)
Proof Formats: RUP, DRUP, RAT, and DRAT

RUP and extensions is the most popular clausal-style format.

\[ E := (\bar{b} \lor c) \land (a \lor c) \land (\bar{a} \lor b) \land (\bar{a} \lor \bar{b}) \land (a \lor \bar{b}) \land (b \lor \bar{c}) \]

RUP is much more compact than TraceCheck because it does not includes the resolution steps.

\[
\langle \text{proof} \rangle = \{\langle \text{lemma} \rangle\}
\]
\[
\langle \text{lemma} \rangle = \langle \text{delete} \rangle\{\langle \text{lit} \rangle\} "0"
\]
\[
\langle \text{delete} \rangle = "\text{\textquoteleft\textquoteleft}" | "\text{\textasciitilde\textquoteright\textquoteright}" \\
\langle \text{lit} \rangle = \langle \text{pos} \rangle | \langle \text{neg} \rangle
\]
\[
\langle \text{pos} \rangle = "1" | "2" | \cdots | \langle \text{max} - \text{idx} \rangle
\]
\[
\langle \text{neg} \rangle = "\text{\textasciitilde}\text{\textquoteright\textquoteright}"\langle \text{pos} \rangle
\]

\[
E \land (b) \vdash_1 \epsilon
\]
\[
E \land (\bar{b}) \land (\bar{c}) \vdash_1 \epsilon
\]
\[
E \land (\bar{b}) \land (c) \vdash_1 \epsilon
\]

\[
\begin{array}{c c c c}
-2 & 0 \\
3 & 0 \\
0 &
\end{array}
\]
Proof Formats: Open Issues and Challenges

How get useful information from a proof?
- Clausal or variable core
- Resolution proof from a clausal proof
- Interpolant
- Proof minimization
- Inside the SAT solver or using an external tool?
- What would be a good API to manipulate proofs?

How to store proofs compactly?
- Question is important for resolution and clausal proofs
- Current formats are "readable" and hence large
- Recently we proposed a binary format, reducing size by a factor of three.
Media and Applications
Media: The Largest Math Proof Ever

Two-hundred-terabyte maths proof is largest ever

Computer Generates Largest Math Proof Ever At 200TB of Data

76 comments

Collqteral May 27, 2016 +2
200 Terabytes. Thats about 400 PS4s.
Applications: Erdős Discrepancy Conjecture

Erdős Discrepancy Conjecture was recently solved using SAT. The conjecture states that there exists no infinite sequence of $-1, +1$ such that for all $d, k$ holds that $(x_i \in \{-1, +1\})$:

$$\left| \sum_{i=1}^{k} x_{id} \right| \leq 2$$
Applications: Erdős Discrepancy Conjecture

Erdős Discrepancy Conjecture was recently solved using SAT. The conjecture states that there exists no infinite sequence of \(-1, +1\) such that for all \(d, k\) holds that \((x_i \in \{-1, +1\}):\)

\[
\left| \sum_{i=1}^{k} x_{id} \right| \leq 2
\]

The DRAT proof was 13Gb and checked with our tool DRAT-trim [SAT14]
Applications: SAT Competitions (mandatory proof logging)

DRAT proof logging supported by all the top-tier solvers:
  ▶ e.g. Lingeling, MiniSAT, Glucose, and CryptoMiniSAT

DRAT-trim validates proofs in a time similar to solving time.
  ▶ computes also unsatisfiable core;
  ▶ optimizes the proof for possible later validations; and
  ▶ can emit a resolution proof (typically huge).

Example run of DRAT-trim on Erdős Discrepancy Proof

fud$ ./DRAT-trim EDP2_1161.cnf EDP2_1161.drat
  c finished parsing
  c detected empty clause; start verification via backward checking
  c 23090 of 25142 clauses in core
  c 5757105 of 6812396 lemmas in core using 469808891 resolution steps
  c 16023 RAT lemmas in core; 5267754 redundant literals in core lemmas
  s VERIFIED
Applications: Ramsey Numbers

Ramsey Number $R(k)$: What is the smallest $n$ such that any graph with $n$ vertices has either a clique or a co-clique of size $k$?

\[
R(3) = 6 \\
R(4) = 18 \\
43 \leq R(5) \leq 49
\]

SAT solvers can determine that $R(4) = 18$ in 1 second using symmetry breaking; w/o symmetry breaking it requires weeks.

Symmetry breaking can be validated using DRAT [CADE’15]
Applications: Ramsey Numbers

Ramsey Number $R(k)$: What is the smallest $n$ such that any graph with $n$ vertices has either a clique or a co-clique of size $k$?

$$R(3) = 6$$
$$R(4) = 18$$
$$43 \leq R(5) \leq 49$$

SAT solvers can determine that $R(4) = 18$ in 1 second using symmetry breaking; w/o symmetry breaking it requires weeks.

Symmetry breaking can be validated using DRAT [CADE’15]
Applications: Ramsey Numbers

Ramsey Number $R(k)$: What is the smallest $n$ such that any graph with $n$ vertices has either a clique or a co-clique of size $k$?

$$R(3) = 6$$
$$R(4) = 18$$
$$43 \leq R(5) \leq 49$$

SAT solvers can determine that $R(4) = 18$ in 1 second using symmetry breaking; w/o symmetry breaking it requires weeks.

Symmetry breaking can be validated using DRAT [CADE’15]
Conclusions
Conclusions

Proofs of unsatisfiability useful for several applications:

▶ Validate results of SAT solvers;
▶ Extracting minimal unsatisfiable cores;
▶ Computing Interpolants;
▶ Tools that use SAT solvers, such as theorem provers.

Challenges:

▶ Reduce size of proofs on disk and in memory;
▶ Reduce the cost to validate clausal proofs;
▶ How to deal with Gaussian elimination, cardinality resolution, and pseudo-Boolean reasoning?
Thanks!