Proofs of Unsatisfiability

Marijn J.H. Heule

THE UNIVERSITY OF TEXAS AT AUSTIN

SAT 2016 Industry Day

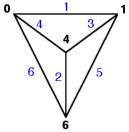
July 9, 2016

Outline

Introduction

Satisfiability (SAT) solving has many applications

formal verification

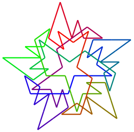


graph theory

bioinformatics

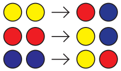
train safety

planning



number theory

cryptography



rewrite termination

$$\begin{tikzcd}
\text{encode} \arrow[r] & \boxed{\text{SAT solver}} \arrow[r] & \text{decode}
\end{tikzcd}$$

A Small Satisfiability (SAT) Problem

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

Does there exist an assignment satisfying all clauses?

Search for a satisfying assignment (or proof none exists)

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

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

$$\frac{(x \lor a_1 \lor \ldots \lor a_i) \qquad (\bar{x} \lor b_1 \lor \ldots \lor b_j)}{(a_1 \lor \ldots \lor a_i \lor b_1 \lor \ldots \lor b_j)}$$

- Many SAT techniques can be simulated by resolution.

Resolution Rule and Resolution Chains

Resolution Rule

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

- Many SAT techniques can be simulated by resolution.

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

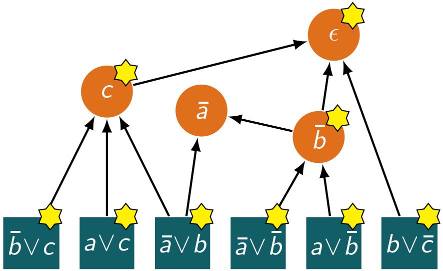
Example

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

Resolution Proofs versus Clausal Proofs

Consider the formula $F := (\bar{b} \vee c) \wedge (a \vee c) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee \bar{b}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c})$

A resolution graph of F is:



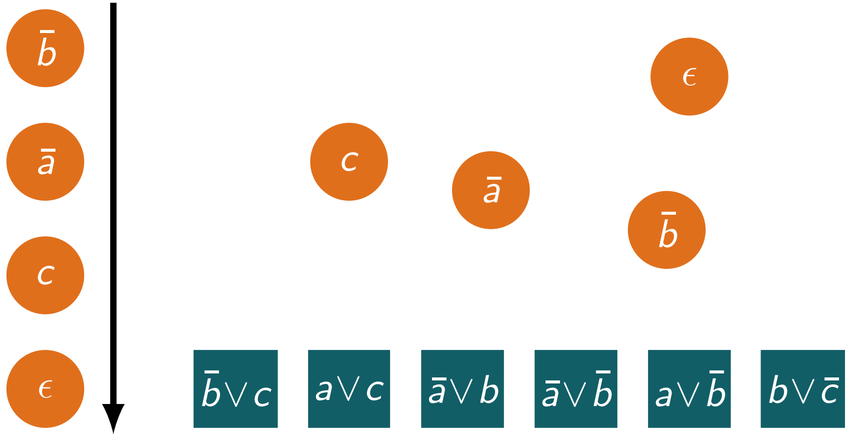
A resolution proof consists of all nodes and edges of the resolution graph

- Graphs from SAT solvers have ~ 400 incoming edges per node
- Resolution proof logging can heavily increase memory usage ($\times 100$)

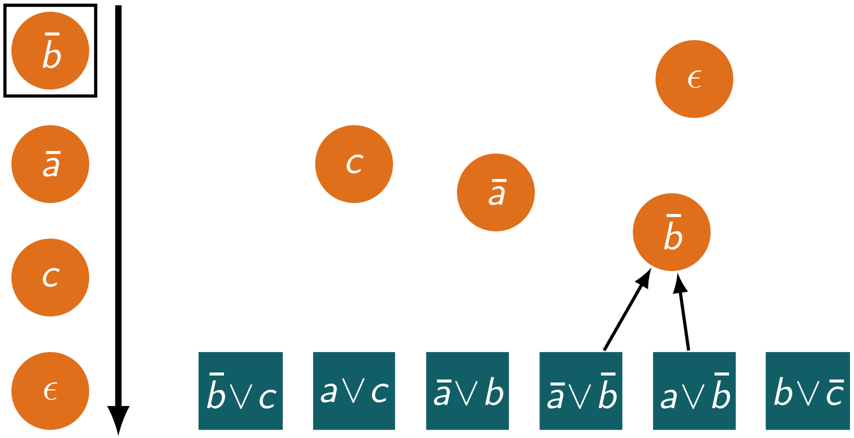
A clausal proof is a list of all nodes sorted by topological order

- Clausal proofs are easy to emit and relatively small
- Clausal proof checking requires to reconstruct the edges (costly)

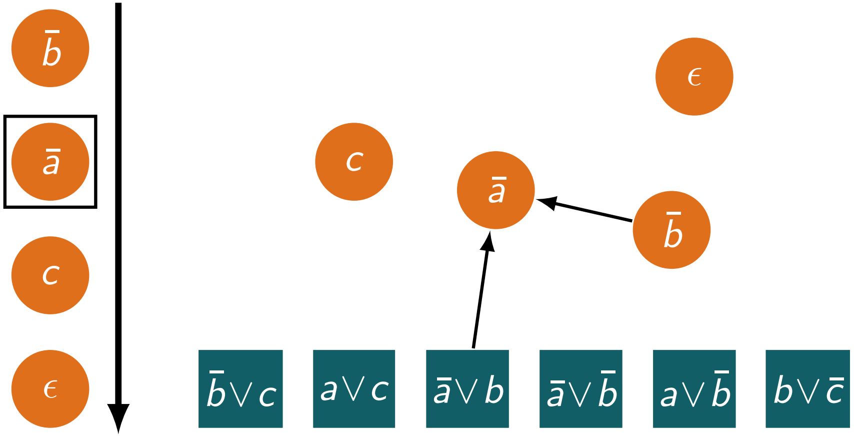
Clausal Proof: Checker has to reconstruct resolution edges



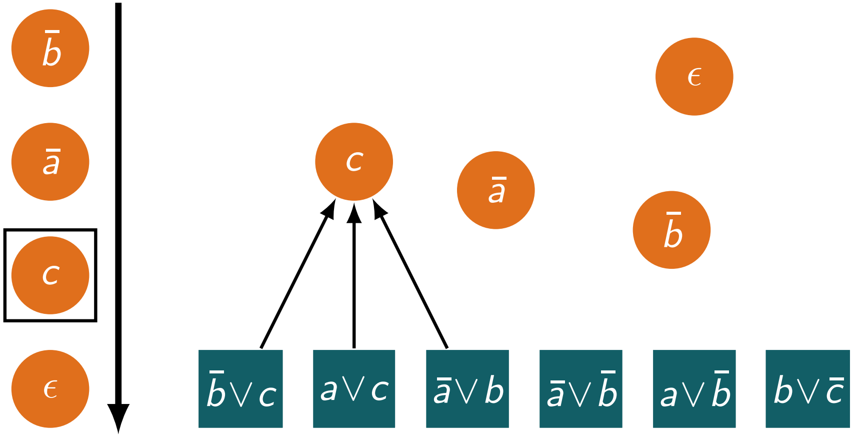
Clausal Proof: Checker has to reconstruct resolution edges



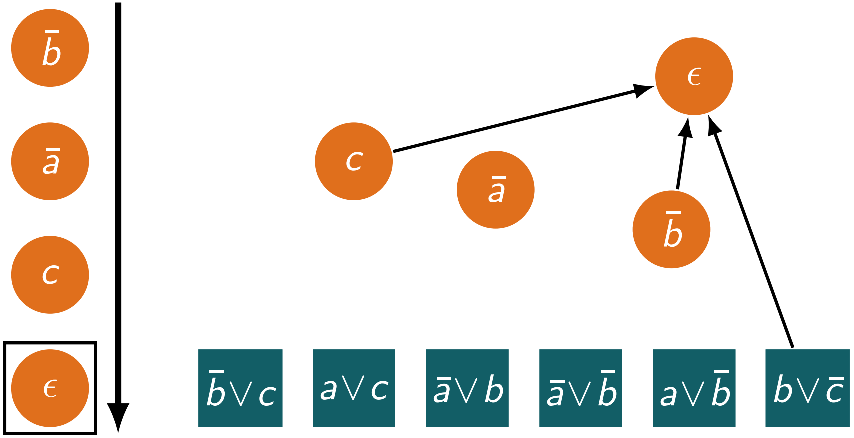
Clausal Proof: Checker has to reconstruct resolution edges



Clausal Proof: Checker has to reconstruct resolution edges



Clausal Proof: Checker has to reconstruct resolution edges



Improvement I: Backwards Checking

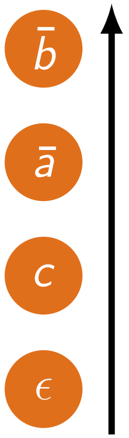
Goldberg and Novikov proposed checking the refutation backwards [DATE 2003]:

- 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.



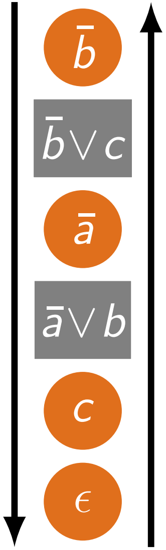
Improvement II: Clause Deletion

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.



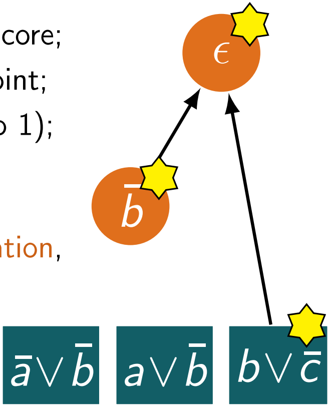
Improvement III: Core-first Unit Propagation

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.

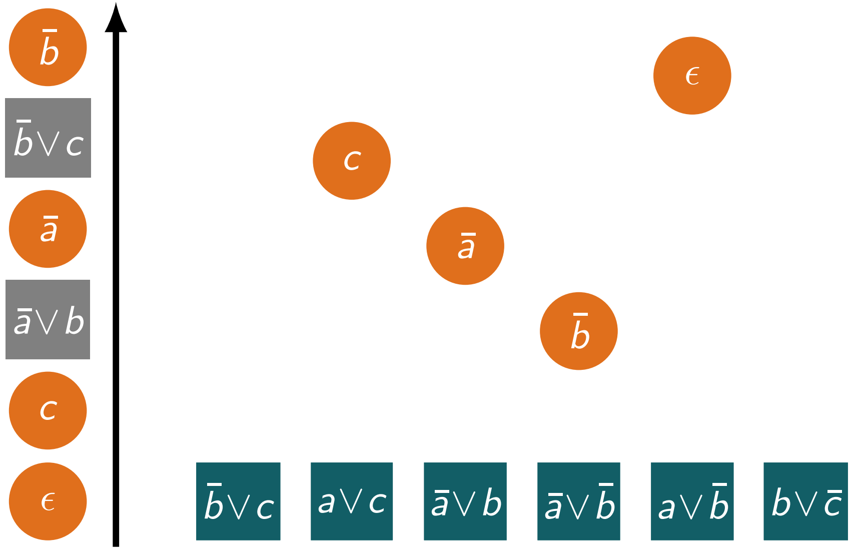
Our variant, called Core-first Unit Propagation, can reduce checking costs considerably.

Fast propagation in a checker is different than fast propagation in a SAT solver.



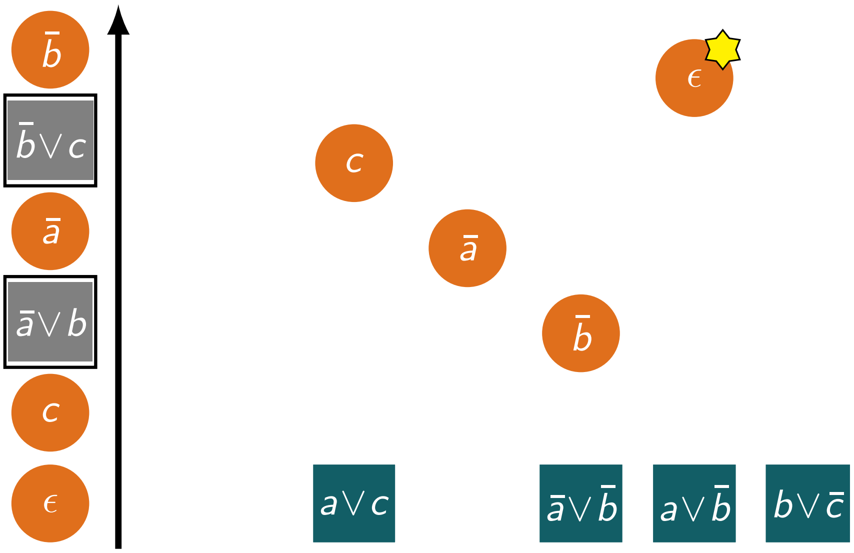
Also, the resulting core and proof are smaller

Checking: Backwards + Core-first + Deletion



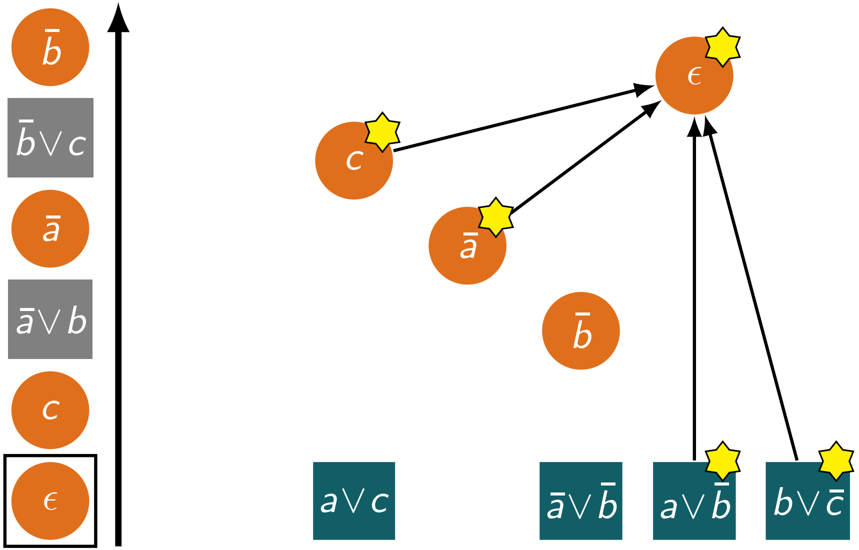
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



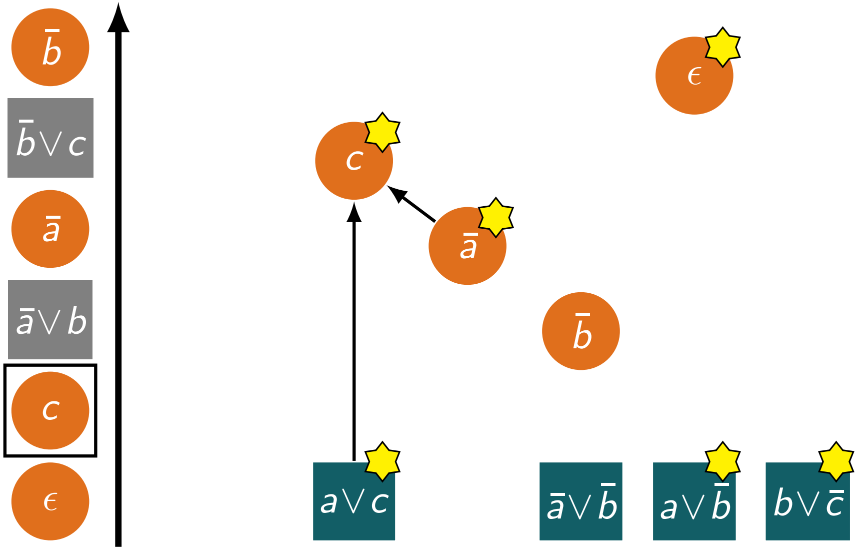
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



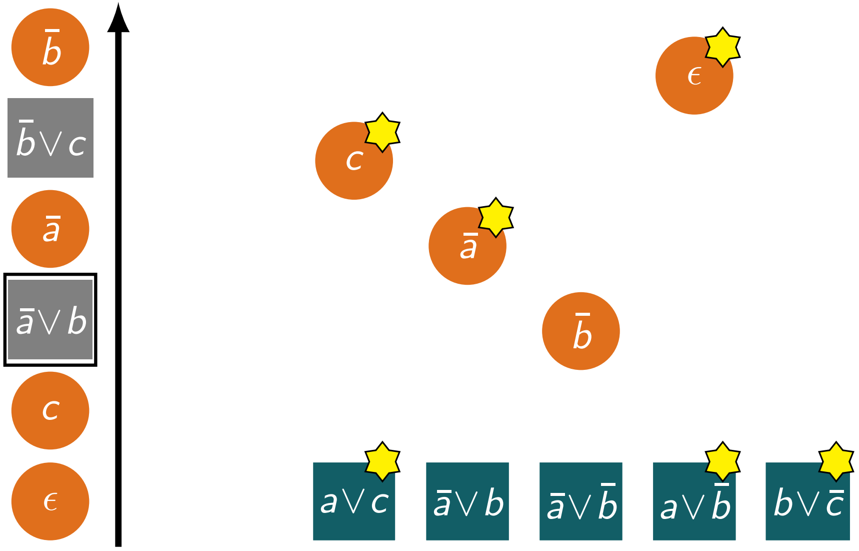
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



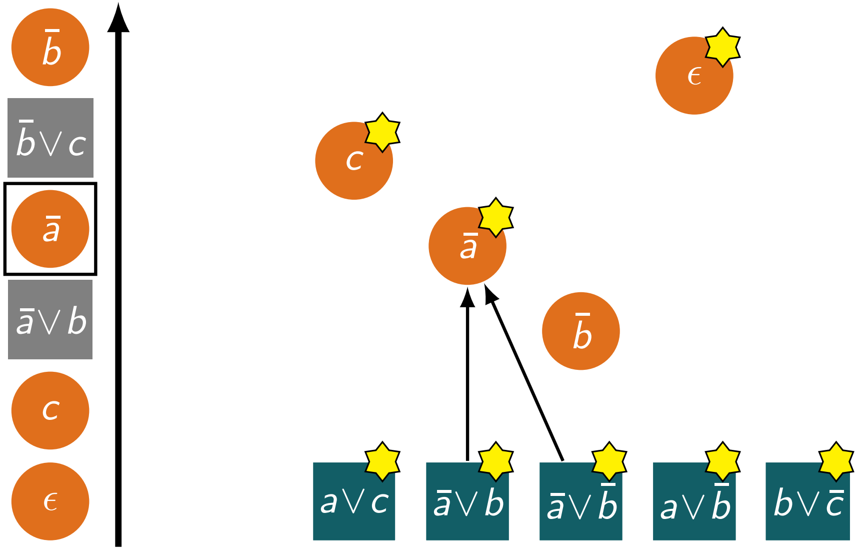
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



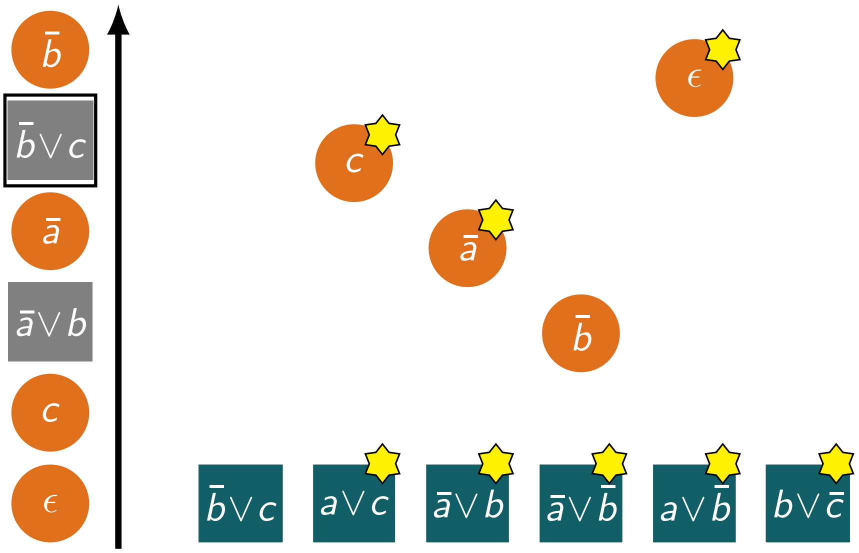
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



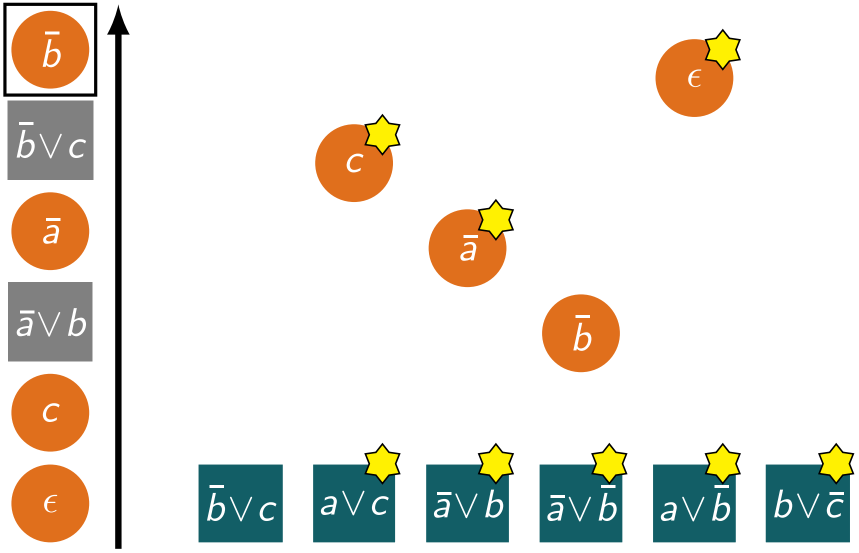
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



Core-first unit propagation results in smaller cores and proofs

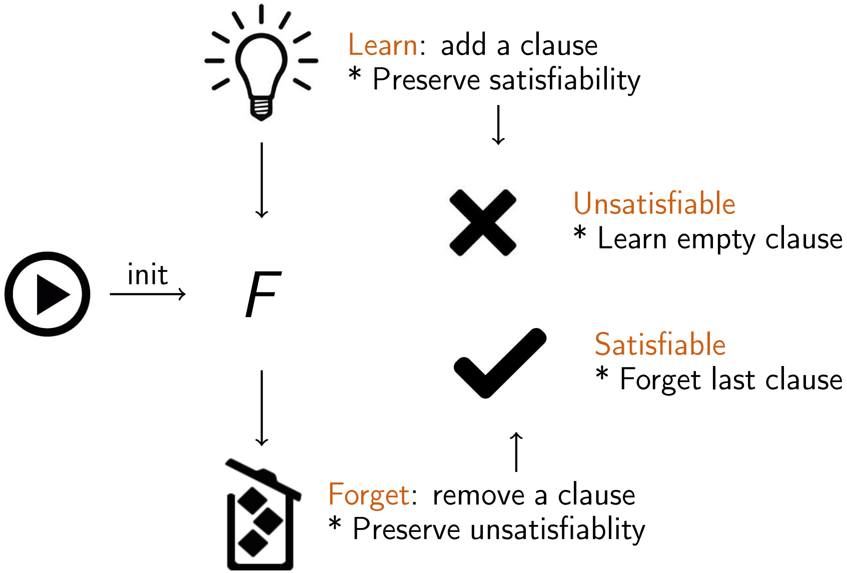
Checking: Backwards + Core-first + Deletion



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

init

F

Satisfiable
* Forget last clause

Forget: remove a clause
* Preserve unsatisfiablity

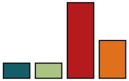
Ideal Properties of a Proof System for SAT Solvers

Easy to Emit

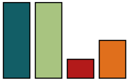
Compact

Checked Efficiently

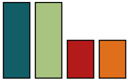
Expressive



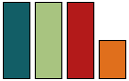
Resolution Proofs
Zhang and Malik, 2003
Van Gelder, 2008; Biere, 2008



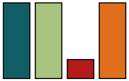
Clausal Proofs
Goldberg and Novikov, 2003
Van Gelder, 2008



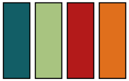
Clausal proofs + clause deletion
Heule, Hunt, Jr., and Wetzler [STVR 2014]



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

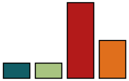
Easy to Emit

Compact

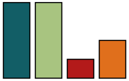
Checked Efficiently

Expressive

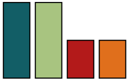
Verified



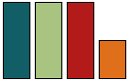
Resolution Proofs
Zhang and Malik, 2003
Van Gelder, 2008; Biere, 2008



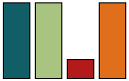
Clausal Proofs
Goldberg and Novikov, 2003
Van Gelder, 2008



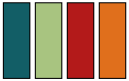
Clausal proofs + clause deletion
Heule, Hunt, Jr., and Wetzler [STVR 2014]



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]

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`

```
p cnf 3 6
-2  3 0
 1  3 0
-1  2 0
-1 -2 0
 1 -2 0
 2 -3 0
```

Most proof formats use a similar syntax.

Proof Formats: TraceCheck Overview

TraceCheck is the most popular resolution-style format.

$$E := (\bar{b} \vee c) \wedge (a \vee c) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee \bar{b}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c})$$

TraceCheck is readable and resolution chains make it relatively compact

$$\begin{aligned}
\langle \text{trace} \rangle &= \{\langle \text{clause} \rangle\} \\
\langle \text{clause} \rangle &= \langle \text{pos} \rangle \langle \text{literals} \rangle \langle \text{antecedents} \rangle \\
\langle \text{literals} \rangle &= \text{“}*\text{”} \mid \{\langle \text{lit} \rangle\}\text{“0”} \\
\langle \text{antecedents} \rangle &= \{\langle \text{pos} \rangle\}\text{“0”} \\
\langle \text{lit} \rangle &= \langle \text{pos} \rangle \mid \langle \text{neg} \rangle \\
\langle \text{pos} \rangle &= \text{“1”} \mid \text{“2”} \mid \cdots \mid \langle \text{max}-\text{idx} \rangle \\
\langle \text{neg} \rangle &= \text{“}-\text{”}\langle \text{pos} \rangle
\end{aligned}$$

```
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
```

Proof Formats: TraceCheck Examples

TraceCheck is the most popular resolution-style format.

$$E := (\bar{b} \vee c) \wedge (a \vee c) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee \bar{b}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c})$$

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**:

- $(\bar{b}) := (\bar{a} \vee \bar{b}) \diamond (a \vee \bar{b})$

Clause **8** is the resolvent **1**, **2** and **3**:

- $(c) := (\bar{b} \vee c) \diamond (\bar{a} \vee b) \diamond (a \vee c)$
- NB: the antecedents are swapped!

Clause **9** is the resolvent **6**, **7** and **8**:

- $\epsilon := (b \vee \bar{c}) \diamond (\bar{b}) \diamond (c)$

```
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
```

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

```
8  3  0 1 2 3 0        7 -2  0 4 5 0
7 -2  0 4 5 0     ≡    8  3  0 1 2 3 0
```

- The antecedents of a clause can be in arbitrary order

```
7 -2  0 5 4 0          7 -2  0 4 5 0
8  3  0 3 1 2 0   ≡    8  3  0 1 2 3 0
```

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

```
7 * 5 4 0              7 -2  0 4 5 0
8 * 3 1 2 0       ≡    8  3  0 1 2 3 0
```

Demo: Clausal Proof to TraceCheck

Proof Formats: Reverse Unit Propagation (RUP)

Unit Propagation

Given an assignment φ, extend it by making unit clauses true — until fixpoint or a clause becomes false

Reverse Unit Propagation (RUP)

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

A clause sequence $C_1, \ldots, C_m$ is a RUP proof for formula F

- $F \wedge C_1 \wedge \cdots \wedge C_{i-1} \wedge \bar{C}_i \vdash_1 \epsilon$
- $C_m = \epsilon$

Proof Formats: RUP, DRUP, RAT, and DRAT

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

$$E := (\bar{b} \vee c) \wedge (a \vee c) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee \bar{b}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c})$$

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

$$\begin{array}{rcl}
\langle\text{proof}\rangle & = & \{\langle\text{lemma}\rangle\} \\
\langle\text{lemma}\rangle & = & \langle\text{delete}\rangle\{\langle\text{lit}\rangle\}\,\text{“0”} \\
\langle\text{delete}\rangle & = & \text{“”} \mid \text{“d”} \\
\langle\text{lit}\rangle & = & \langle\text{pos}\rangle \mid \langle\text{neg}\rangle \\
\langle\text{pos}\rangle & = & \text{“1”} \mid \text{“2”} \mid \cdots \mid \langle\text{max}-\text{idx}\rangle \\
\langle\text{neg}\rangle & = & \text{“}-\text{”}\langle\text{pos}\rangle
\end{array}$$

```
  −2   0
   3   0
   0
```

$$E \wedge (b) \vdash_1 \epsilon$$
$$E \wedge (\bar{b}) \wedge (\bar{c}) \vdash_1 \epsilon$$
$$E \wedge (\bar{b}) \wedge (c) \vdash_1 \epsilon$$

Proof Formats: Open Issues and Challenges

How get useful information from a proof?

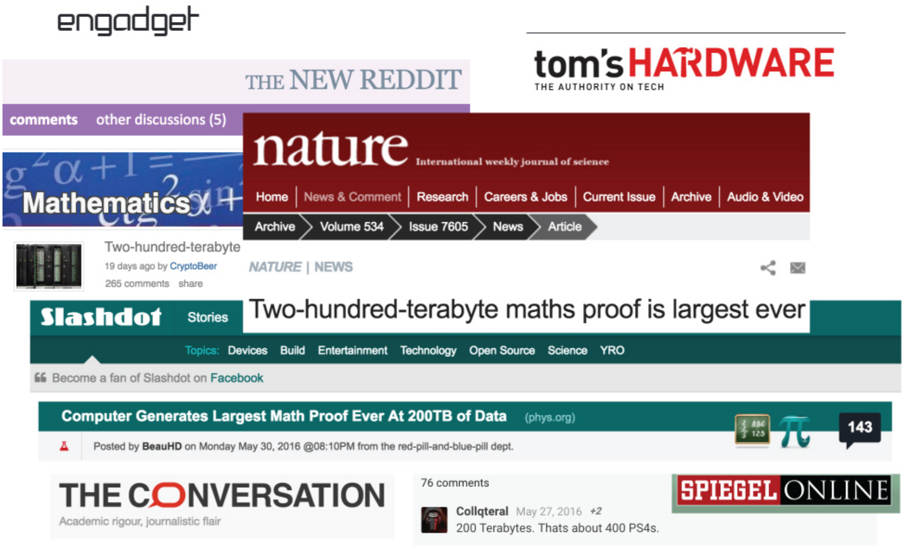
- Clausal or variable core
- Resolution proof from a clausal proof
- Interpolant
- Proof minimization
- Inside the SAT solver or using an external tool?
- What would be a good API to manipulate proofs?

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


engadget

tom's HARDWARE

THE AUTHORITY ON TECH

THE NEW REDDIT

comments other discussions (5)

Mathematics

Two-hundred-terabyte

19 days ago by CryptoBeer

265 comments share

nature International weekly journal of science

Home | News & Comment | Research | Careers & Jobs | Current Issue | Archive | Audio & Video

Archive > Volume 534 > Issue 7605 > News > Article

NATURE | NEWS

Two-hundred-terabyte maths proof is largest ever

Slashdot Stories

Topics: Devices Build Entertainment Technology Open Source Science YRO

Become a fan of Slashdot on Facebook

Computer Generates Largest Math Proof Ever At 200TB of Data (phys.org) 143

Posted by **BeauHD** on Monday May 30, 2016 @08:10PM from the red-pill-and-blue-pill dept.

THE CONVERSATION

Academic rigour, journalistic flair

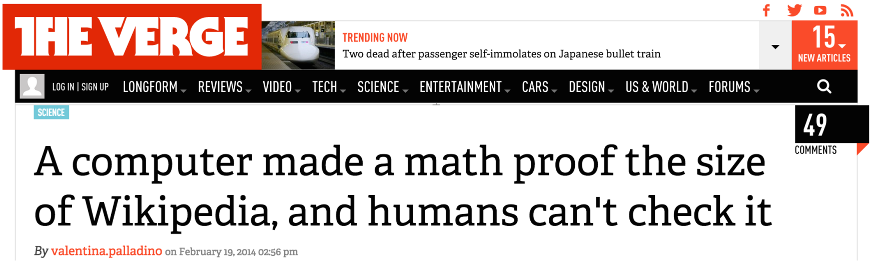
76 comments

Collqteral May 27, 2016 +2

200 Terabytes. Thats about 400 PS4s.

SPIEGEL ONLINE

Applications: Erdős Discrepancy Conjecture


THE VERGE

TRENDING NOW

Two dead after passenger self-immolates on Japanese bullet train

15 NEW ARTICLES

LOG IN | SIGN UP LONGFORM REVIEWS VIDEO TECH SCIENCE ENTERTAINMENT CARS DESIGN US & WORLD FORUMS

SCIENCE

49 COMMENTS

A computer made a math proof the size of Wikipedia, and humans can't check it

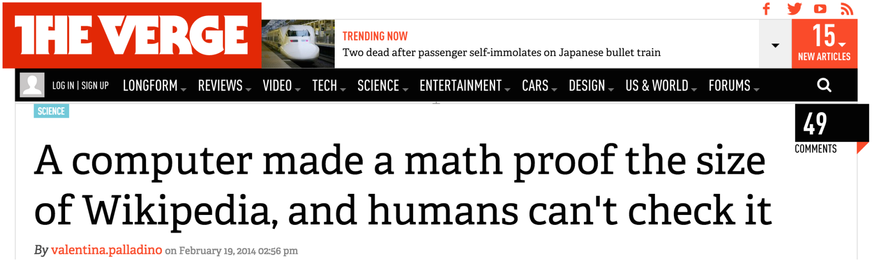
By valentina.palladino on February 19, 2014 02:56 pm

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 ($x_i \in \{-1, +1\}$):

$$\left|\sum_{i=1}^{k} x_{id}\right| \le 2$$

Applications: Erdős Discrepancy Conjecture


THE VERGE

TRENDING NOW

Two dead after passenger self-immolates on Japanese bullet train

15 NEW ARTICLES

LOG IN | SIGN UP LONGFORM REVIEWS VIDEO TECH SCIENCE ENTERTAINMENT CARS DESIGN US & WORLD FORUMS

SCIENCE

49 COMMENTS

A computer made a math proof the size of Wikipedia, and humans can't check it

By valentina.palladino on February 19, 2014 02:56 pm

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).

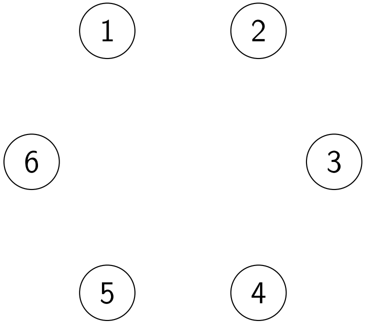
Example run of DRAT-trim on Erdős Discrepancy Proof

```
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?

$$\begin{aligned} & R(3) = 6 \\ & R(4) = 18 \\ 43 \leq\ & R(5) \leq 49 \end{aligned}$$



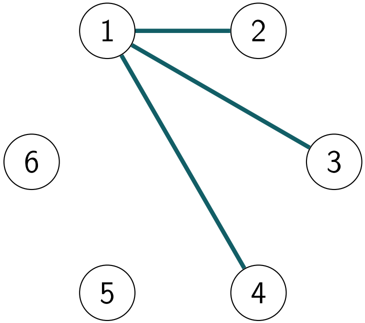
SAT solvers can determine that $R(4) = 18$ in 1 second using symmetry breaking; w/o symmetry breaking it requires weeks.

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?

$$\begin{aligned} R(3) &= 6 \\ R(4) &= 18 \\ 43 \leq \quad R(5) &\leq 49 \end{aligned}$$



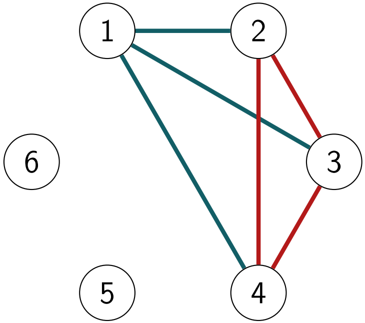
SAT solvers can determine that $R(4) = 18$ in 1 second using symmetry breaking; w/o symmetry breaking it requires weeks.

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?

$$\begin{aligned} R(3) &= 6 \\ R(4) &= 18 \\ 43 \leq \quad R(5) &\leq 49 \end{aligned}$$



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!