Verifying Automated Reasoning Results

Ruben Martins

Carnegie Mellon University

http://www.cs.cmu.edu/~rubenm/15816-f25/

https://github.com/marijnheule/proof-demo

Automated Reasoning and Satisfiability, October 1, 2025

Outline

Introduction

Proof Checking

Proof Systems and Formats

Certified Checking

Applications

Conclusions

Introduction

Proof Checking

Proof Systems and Formats

Certified Checking

Applications

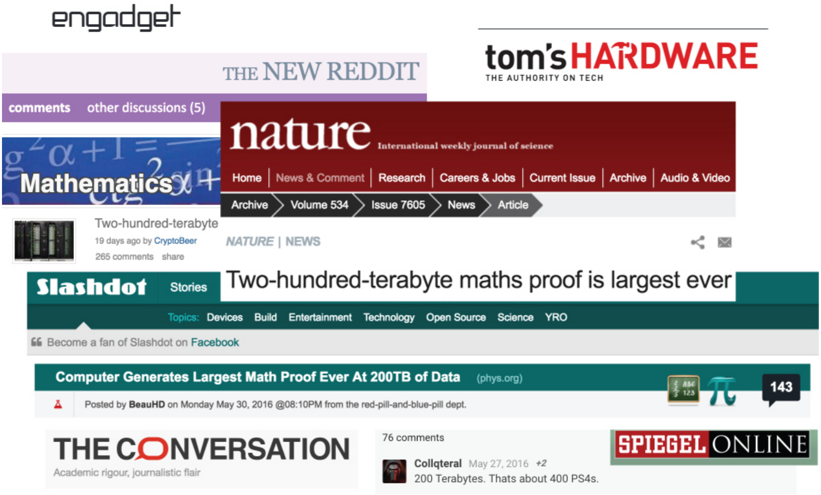
Conclusions

Motivation for Validating Proofs of Unsatisfiability

SAT solvers may have errors and only return yes/no.

- Documented bugs in SAT, SMT, and QSAT solvers; [Brummayer and Biere, 2009; Brummayer et al., 2010]
- Competition winners have contradictory results (HWMCC winners from 2011 and 2012)
- Implementation errors often imply conceptual errors;
- Proofs now mandatory for the annual SAT Competitions;
- Mathematical results require a stronger justification than a simple yes/no by a solver. UNSAT must be verifiable.

"The Largest Math Proof Ever"



Demo: Validating Results

```
git clone https://github.com/marijnheule/proof-demo
```

Introduction

Proof Checking

Proof Systems and Formats

Certified Checking

Applications

Conclusions

Resolution Rule and Resolution Chains

Resolution Rule

$$\frac{C \vee x \qquad \overline{x} \vee D}{C \vee D}$$

- Or equivalently: $C \vee D := (C \vee x) \bowtie (\overline{x} \vee D)$
- Many SAT techniques can be simulated by resolution.

Resolution Rule and Resolution Chains

Resolution Rule

$$\frac{C \vee x \qquad \overline{x} \vee D}{C \vee D}$$

- Or equivalently: $C \vee D := (C \vee x) \bowtie (\overline{x} \vee D)$
- 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.

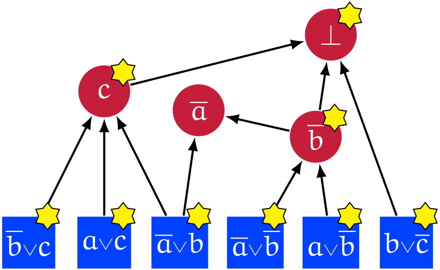
Example

- $(c) := (\overline{a} \vee \overline{b} \vee c) \bowtie (\overline{a} \vee b) \bowtie (a \vee c)$
- $(\overline{a} \vee c) := (\overline{a} \vee b) \bowtie (a \vee c) \bowtie (\overline{a} \vee \overline{b} \vee c)$
- The order of the clauses in the chain matter

Resolution Proofs versus Clausal Proofs

Consider $\Gamma := (\overline{b} \vee c) \wedge (a \vee c) \wedge (\overline{a} \vee b) \wedge (\overline{a} \vee \overline{b}) \wedge (a \vee \overline{b}) \wedge (b \vee \overline{c})$

A resolution graph of Γ is:



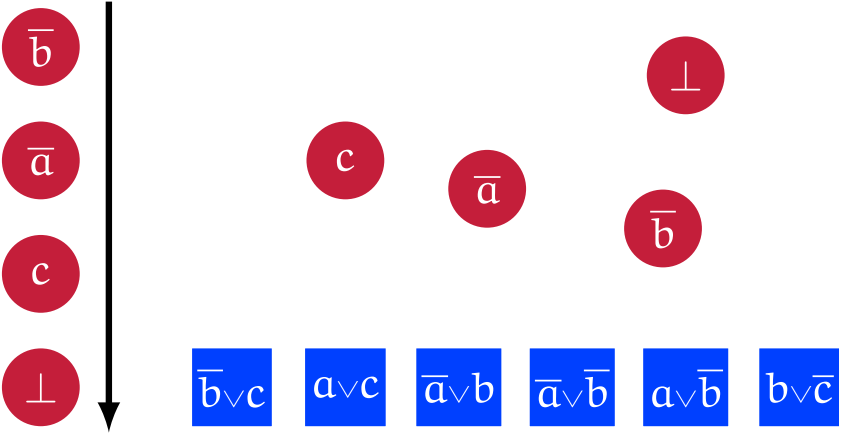
A resolution proof consists of all nodes and edges of the resolution graph

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

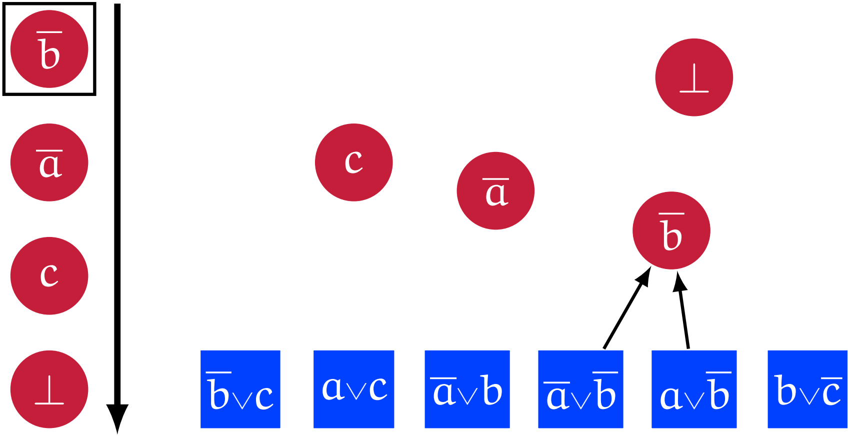
A clausal proof is a list of all nodes sorted by topological order

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

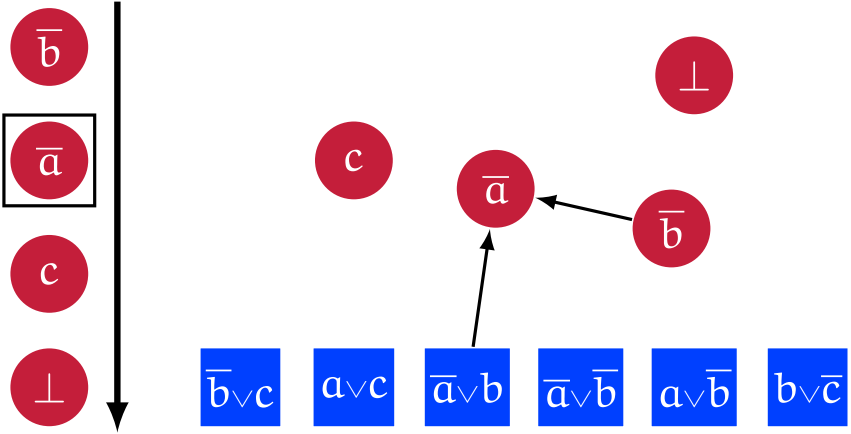
Clausal Proof: Checker has to reconstruct resolution edges



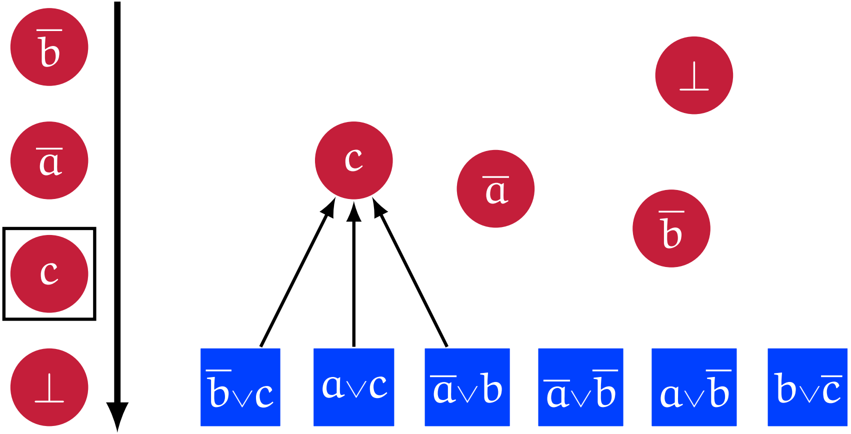
Clausal Proof: Checker has to reconstruct resolution edges



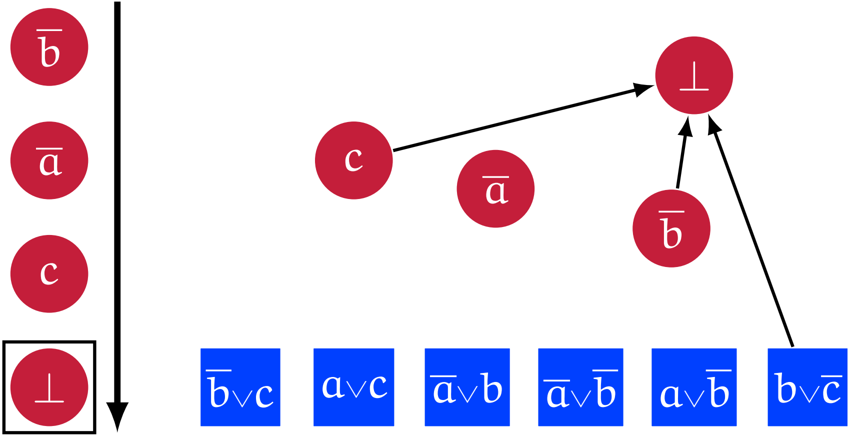
Clausal Proof: Checker has to reconstruct resolution edges



Clausal Proof: Checker has to reconstruct resolution edges



Clausal Proof: Checker has to reconstruct resolution edges



Reconstruct Edges Efficiently: Reverse Unit Propagation

- Unit propagation (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \land \neg C$ results in a conflict.

Example

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

Reconstruct Edges Efficiently: Reverse Unit Propagation

- Unit propagation (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \land \neg C$ results in a conflict.

Example

$$\begin{aligned}\Gamma = {} & (a \lor b \lor \bar{c}) \land (\bar{a} \lor \bar{b} \lor c) \land (b \lor c \lor \bar{d}) \land (\bar{b} \lor \bar{c} \lor d) \land \\ & (a \lor c \lor d) \land (\bar{a} \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor b \lor d) \land (a \lor \bar{b} \lor \bar{d})\end{aligned}$$

clause	$(a \lor b)$
units	$\bar{a} \land \bar{b}$

Reconstruct Edges Efficiently: Reverse Unit Propagation

- Unit propagation (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \wedge \neg C$ results in a conflict.

Example

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

clause	$(a \vee b)$	$(a \vee b \vee \bar{c})$
units	$\bar{a} \wedge \bar{b}$	$\bar{c}$

Reconstruct Edges Efficiently: Reverse Unit Propagation

- Unit propagation (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \land \neg C$ results in a conflict.

Example

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

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

Reconstruct Edges Efficiently: Reverse Unit Propagation

- Unit propagation (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \land \neg C$ results in a conflict.

Example

$$\Gamma = (a \lor b \lor \bar{c}) \land (\bar{a} \lor \bar{b} \lor c) \land (b \lor c \lor \bar{d}) \land (\bar{b} \lor \bar{c} \lor d) \land (a \lor c \lor d) \land (\bar{a} \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor b \lor d) \land (a \lor \bar{b} \lor \bar{d})$$

clause	$(a \lor b)$	$(a \lor b \lor \bar{c})$	$(b \lor c \lor \bar{d})$	$(a \lor c \lor d)$
units	$\bar{a} \land \bar{b}$	$\bar{c}$	$\bar{d}$	$\bot$

Reconstruct Edges Efficiently: Reverse Unit Propagation

- Unit propagation (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Let F be a formula. A clause C is implied by F via UP (denoted by $F \vdash_1 C$) if UP on $F \land \neg C$ results in a conflict.

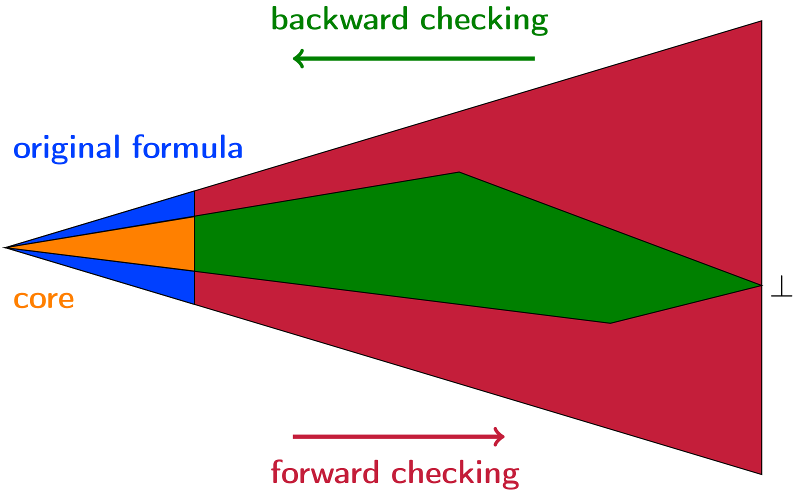
Example

$$\Gamma = (a \lor b \lor \bar{c}) \land (\bar{a} \lor \bar{b} \lor c) \land (b \lor c \lor \bar{d}) \land (\bar{b} \lor \bar{c} \lor d) \land (a \lor c \lor d) \land (\bar{a} \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor b \lor d) \land (a \lor \bar{b} \lor \bar{d})$$

clause	$(a \lor b)$	$(a \lor b \lor \bar{c})$	$(b \lor c \lor \bar{d})$	$(a \lor c \lor d)$
units	$\bar{a} \land \bar{b}$	$\bar{c}$	$\bar{d}$	$\bot$

$$\dfrac{\dfrac{(a \lor c \lor d) \qquad (b \lor c \lor \bar{d})}{(a \lor b \lor c)} \qquad (a \lor b \lor \bar{c})}{(a \lor b)}$$

Forward vs Backward Proof Checking



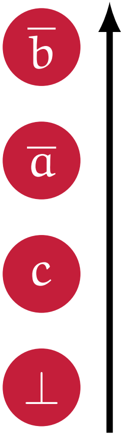
Improvement I: Backwards Checking

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.



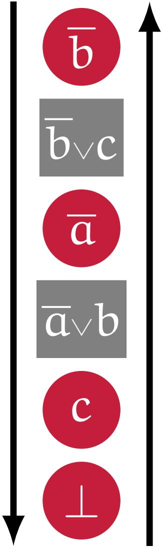
Improvement II: Clause Deletion

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.



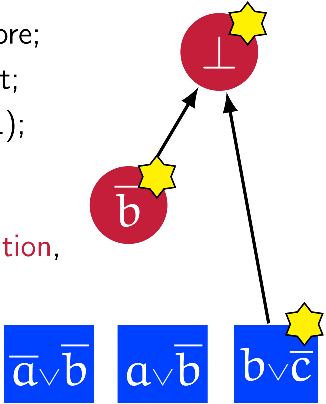
Improvement III: Core-first Unit Propagation

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.

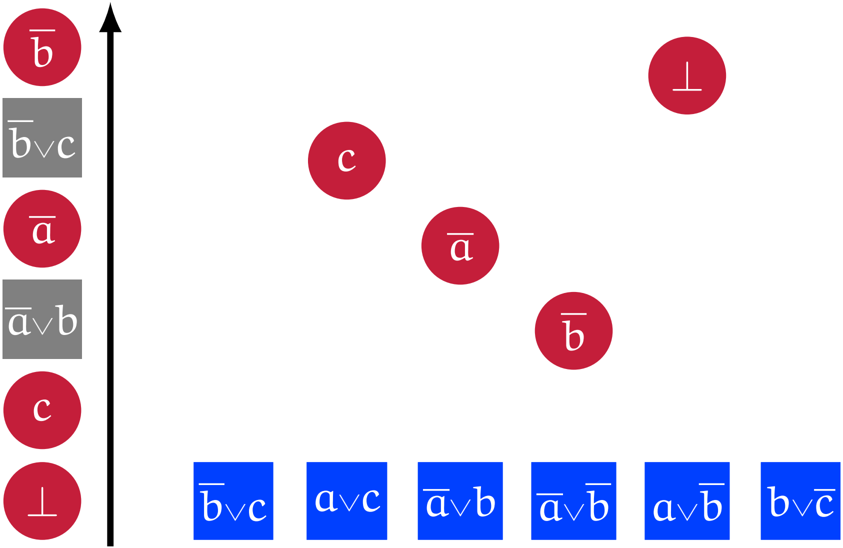
The variant, called Core-first Unit Propagation, can reduce checking costs considerably.

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



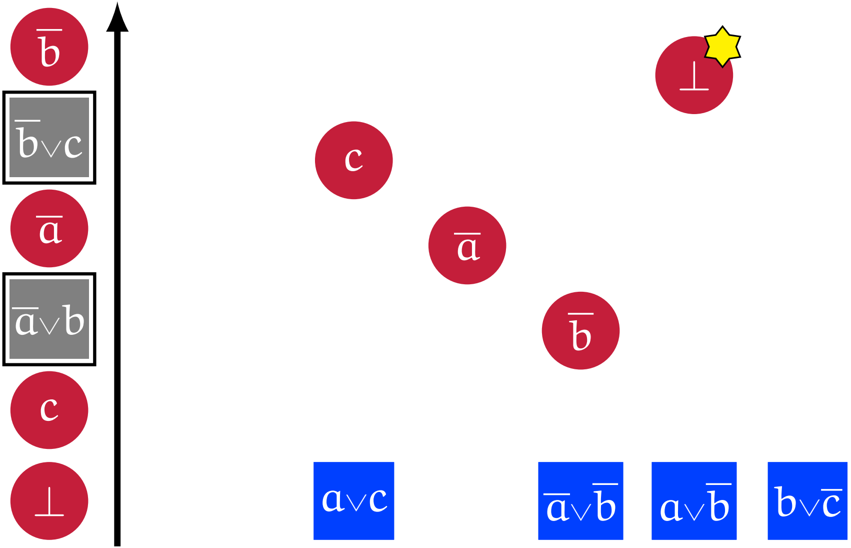
Also, the resulting core and proof are smaller

Checking: Backwards + Core-first + Deletion



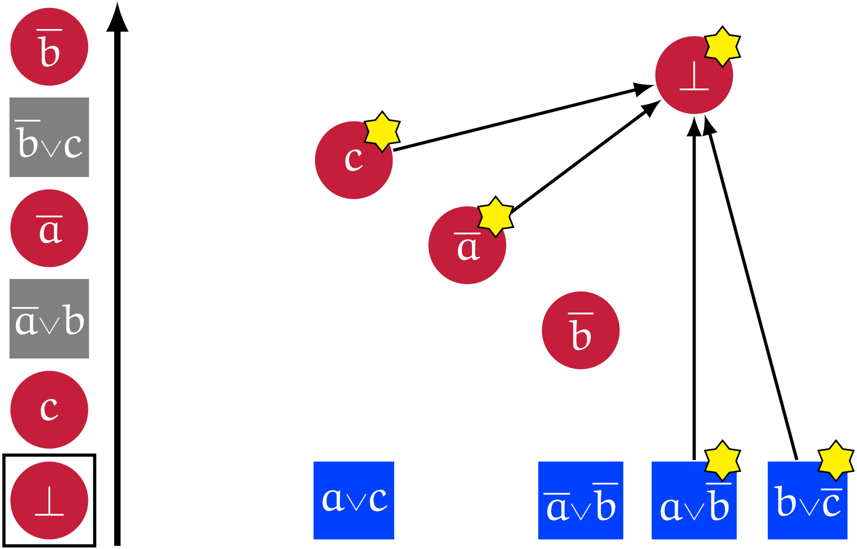
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



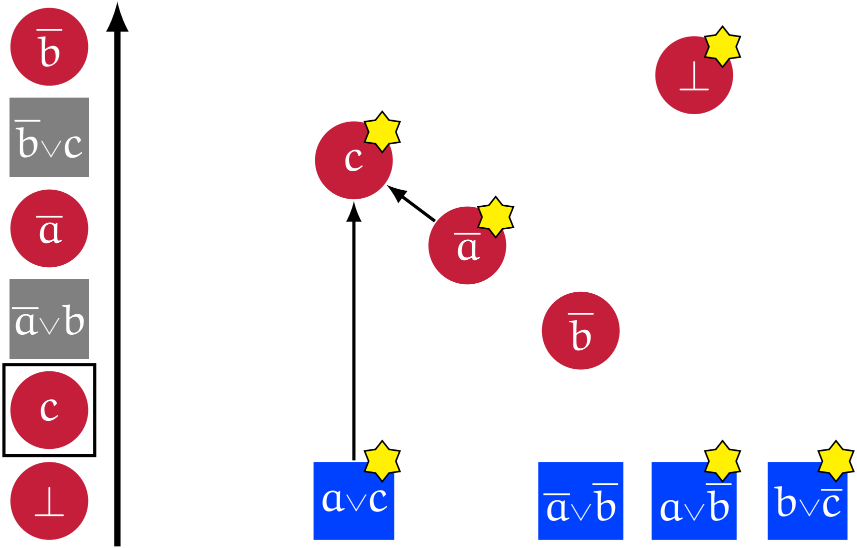
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



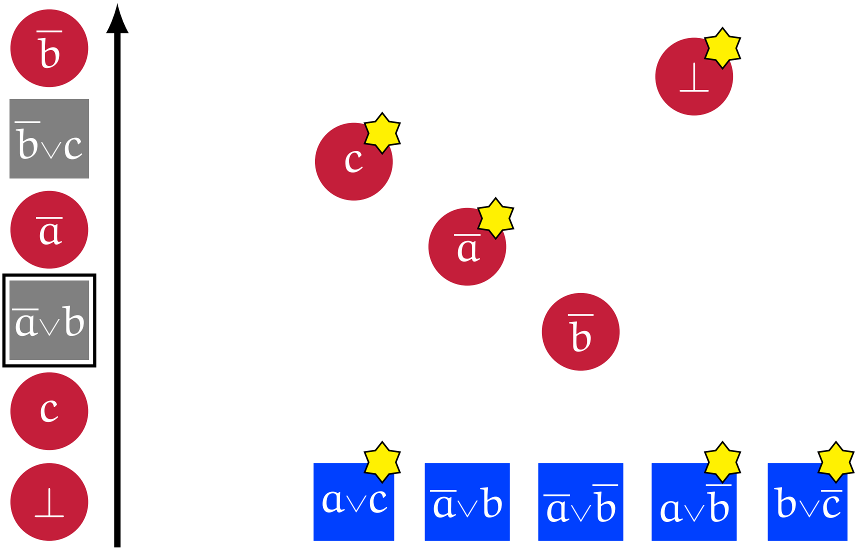
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



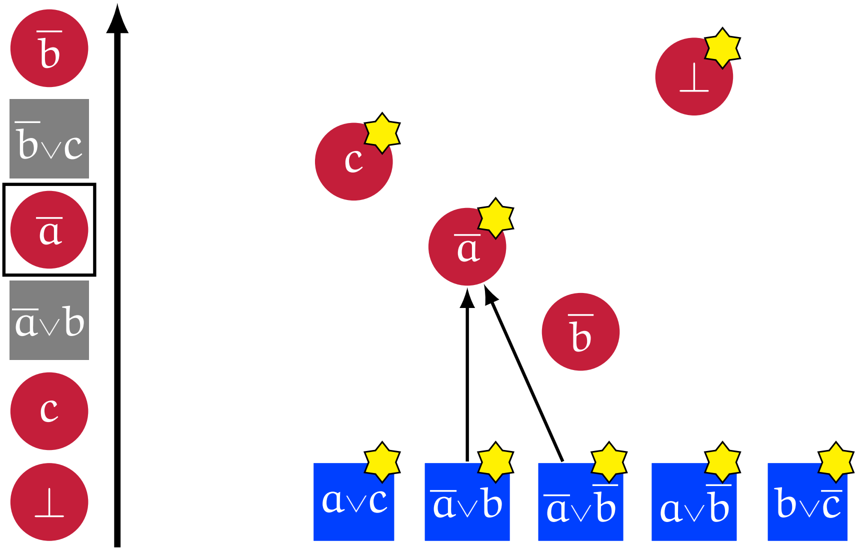
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



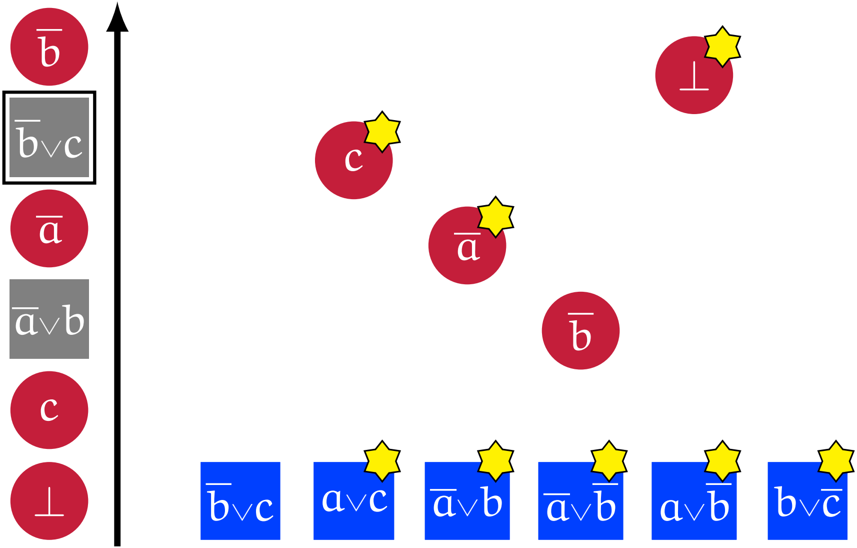
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



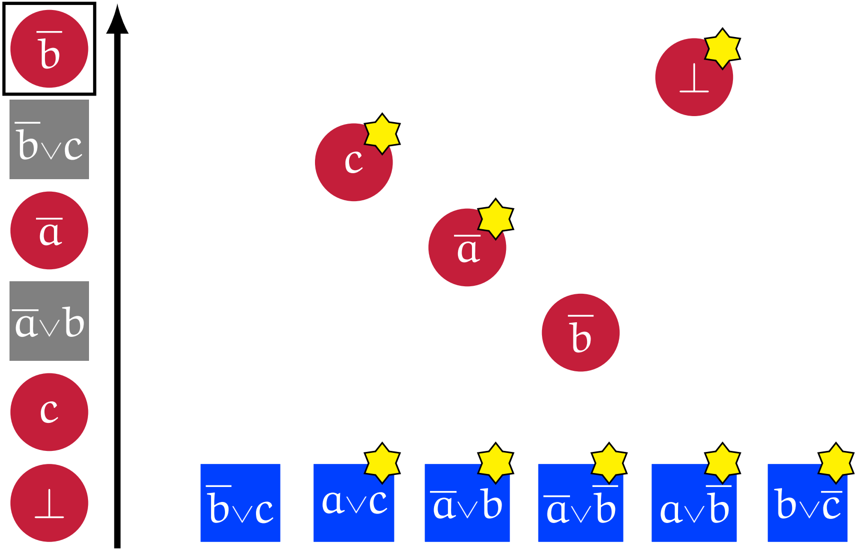
Core-first unit propagation results in smaller cores and proofs

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

DRAT (Deletion Resolution Asymmetric Tautology)

Drawbacks of resolution:

- For many seemingly simple formulas, there are only resolution proofs of exponential size.
- State-of-the-art solving techniques are not succinctly expressible.

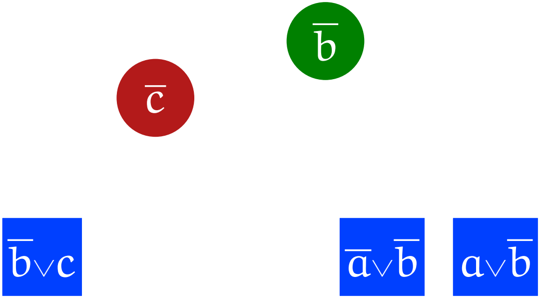
A clause $(C \vee x)$ is a resolution asymmetric tautology (RAT) on x w.r.t. a CNF formula Γ if for every clause $(D \vee \overline{x}) \in \Gamma$, the resolvent $C \vee D$ is implied by Γ via unit-propagation, i.e., $\Gamma \vdash_1 C \vee D$.

Popular example of a clausal proof system: DRAT

- DRAT allows the addition of RATs to a formula.
  - RATs are not necessarily implied by the formula.
  - But RATs are redundant: their addition preserves satisfiability.
  - Clause deletion may introduce clause addition options (interference)

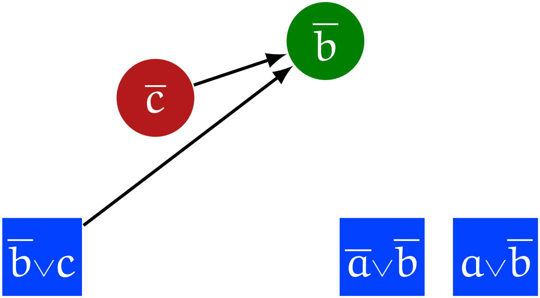
DRAT Example

A clause $(C \vee x)$ is a resolution asymmetric tautology (RAT) on x w.r.t. a CNF formula Γ if for every clause $(D \vee \overline{x}) \in \Gamma$, the resolvent $C \vee D$ is implied by Γ via unit-propagation, i.e., $\Gamma \vdash_1 C \vee D$.



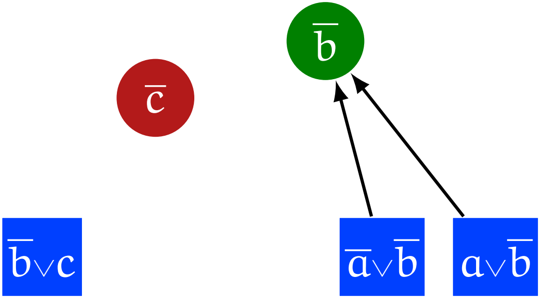
DRAT Example

A clause $(C \vee x)$ is a resolution asymmetric tautology (RAT) on x w.r.t. a CNF formula Γ if for every clause $(D \vee \overline{x}) \in \Gamma$, the resolvent $C \vee D$ is implied by Γ via unit-propagation, i.e., $\Gamma \vdash_1 C \vee D$.



DRAT Example

A clause $(C \vee x)$ is a resolution asymmetric tautology (RAT) on x w.r.t. a CNF formula Γ if for every clause $(D \vee \overline{x}) \in \Gamma$, the resolvent $C \vee D$ is implied by Γ via unit-propagation, i.e., $\Gamma \vdash_1 C \vee D$.



Demo: DRAT step

```
git clone https://github.com/marijnheule/proof-demo
```

Introduction

Proof Checking

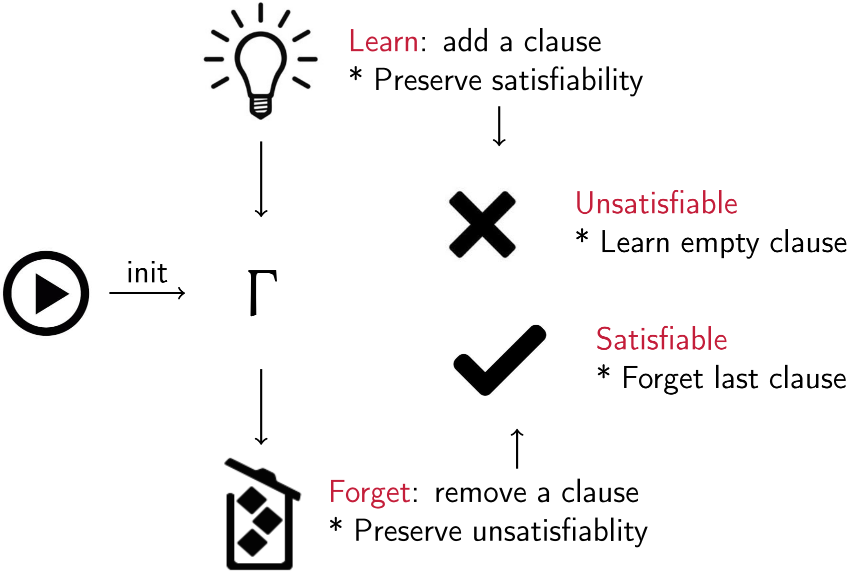
Proof Systems and Formats

Certified Checking

Applications

Conclusions

Clausal Proof System [Järvisalo, Heule, and Biere 2012]



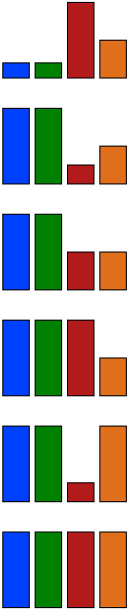
Ideal Properties of a Proof System for SAT Solvers

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 + deletion
Heule, Hunt, Jr., Wetzler [STVR'14]

Optimized clausal proof checker
Heule, Hunt, Jr., and Wetzler [FMCAD'13]

Clausal RAT proofs
Heule, Hunt, Jr., Wetzler [CADE'13]

DRAT proofs (RAT + deletion)
Wetzler, Heule, Hunt, Jr. [SAT'14]

Ideal Properties of a Proof System for SAT Solvers

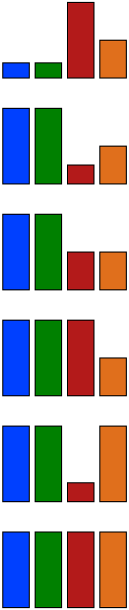
Easy to Emit

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 + deletion
Heule, Hunt, Jr., Wetzler [STVR'14]

Optimized clausal proof checker
Heule, Hunt, Jr., and Wetzler [FMCAD'13]

Clausal RAT proofs
Heule, Hunt, Jr., Wetzler [CADE'13]

DRAT proofs (RAT + deletion)
Wetzler, Heule, Hunt, Jr. [SAT'14]

Proof Formats: The Input Format DIMACS

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

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.

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

TraceCheck is readable and resolution chains make it relatively compact

$$\begin{aligned}
\langle \text{trace} \rangle &= \{\langle \text{clause} \rangle\} \\
\langle \text{clause} \rangle &= \langle \text{pos} \rangle \langle \text{literals} \rangle \langle \text{clsidx} \rangle \\
\langle \text{literals} \rangle &= \text{"}{*}\text{"} \mid \{\langle \text{lit} \rangle\}\ \text{"0"} \\
\langle \text{clsidx} \rangle &= \{\langle \text{pos} \rangle\}\ \text{"0"} \\
\langle \text{lit} \rangle &= \langle \text{pos} \rangle \mid \langle \text{neg} \rangle \\
\langle \text{pos} \rangle &= \text{"1"} \mid \text{"2"} \mid \cdots \mid \langle \text{maxidx} \rangle \\
\langle \text{neg} \rangle &= \text{"}-\text{"} \langle \text{pos} \rangle
\end{aligned}$$

```
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  7 8 6 0
```

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 of 4 and 5:

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

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

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

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

- $\bot := (\overline{b}) \bowtie (c) \bowtie (b \lor \overline{c})$

```
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  7 8 6 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
8  3  0 3 1 2 0
```

≡

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

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

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

≡

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

Proof Formats: Clausal Proofs

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

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

RUP is much more compact than LRAT because it lacks hints

- formula not included as well

$$\begin{aligned}
\langle\text{proof}\rangle &= \{\langle\text{lemma}\rangle\} \\
\langle\text{lemma}\rangle &= \langle\text{delete}\rangle\{\langle\text{lit}\rangle\}\ \text{"0"} \\
\langle\text{delete}\rangle &= \text{""} \mid \text{"d"} \\
\langle\text{lit}\rangle &= \langle\text{pos}\rangle \mid \langle\text{neg}\rangle \\
\langle\text{pos}\rangle &= \text{"1"} \mid \text{"2"} \mid \cdots \mid \langle\text{maxidx}\rangle \\
\langle\text{neg}\rangle &= \text{"}-\text{"}\langle\text{pos}\rangle
\end{aligned}$$

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

$$E \wedge (b) \vdash_1 \bot$$
$$E \wedge (\overline{b}) \wedge (\overline{c}) \vdash_1 \bot$$
$$E \wedge (\overline{b}) \wedge (c) \vdash_1 \bot$$

Proof Formats: Binary Formats

There are various cheap compression techniques to shrink proofs:

- Use 4 bytes per literal instead storing the ascii characters
- Sort literals in clauses and store the delta between literals
- Use a variable byte encoding for literals

encoding	**example** (prefix pivot $\text{lit}_1 \ldots \text{lit}_{k-1}$ end)	**#bytes**
ascii	`d 6278 -3425 -42311 9173 22754 0\n`	33
sascii	`d 6278 -3425 9173 22754 -42311 0\n`	33
4byte	`64 0c310000 c31a0000 8f4a0100 aa470000 c4b10000 00000000`	25
s4byte	`64 0c310000 c31a0000 aa470000 c4b10000 8f4a0100 00000000`	25
ds4byte	`64 0c310000 c31a0000 e82c0000 1a6a0000 cb980000 00000000`	25
vbyte	`64 8c62c335 8f9505aa 8f01c4e3 0200`	15
svbyte	`64 8c62c335 aa8f01c4 e3028f95 0500`	15
dsvbyte	`64 8c62c335 e8599ad4 01cbb102 00`	14

Proof Formats: Beyond Checking

Clausal Proof checkers can produce many additional results:

- Clausal core, e.g. useful for MUS computation, MaxSAT
  DRAT-trim option: `-c CORE`
- Extract a resolution proof, e.g. useful for interpolation
  DRAT-trim option: `-r RESPROOF`
- Proof minimization: removing redundant lemmas and literals
  DRAT-trim option: `-l OPTPROOF`

Demo: Proof Mining

```
git clone https://github.com/marijnheule/proof-demo
```

Introduction

Proof Checking

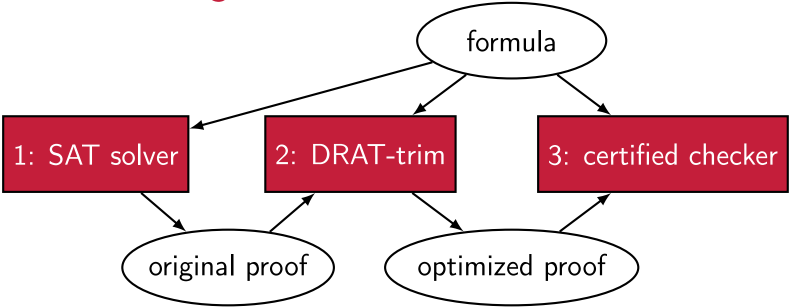
Proof Systems and Formats

Certified Checking

Applications

Conclusions

Certified Checking: Tool Chain



The proof of the Pythagorean Triples problem is almost 200 terabytes (DRAT) and has been validated in 16,000 CPU hours.

This proof has been certified using formally-verified checkers.

Certified Checking: LRAT format

The LRAT format is syntactically similar to TraceCheck, however:

- The formula in **not** included in the proof
- Clause deletion support: ⟨pos⟩“ d "⟨clsidx⟩
- Can express a RAT step: use negative cls to denote resolvent

DIMACS:

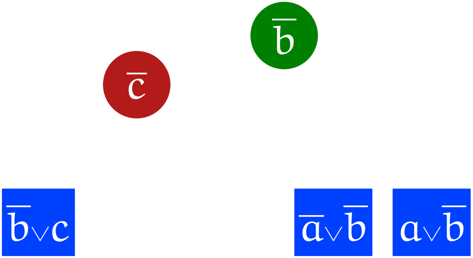
```
p cnf 3 3
-2  3 0
-1 -2 0
 1 -2 0
```

DRAT:

```
-3 0
```

LRAT:

```
4 -3 0 -1 2 3 0
```



Certified Checking: LRAT format

The LRAT format is syntactically similar to TraceCheck, however:

- The formula in **not** included in the proof
- Clause deletion support: $\langle \text{pos} \rangle$“ `d` "$\langle \text{clsidx} \rangle$
- Can express a RAT step: use negative cls to denote resolvent

DIMACS:

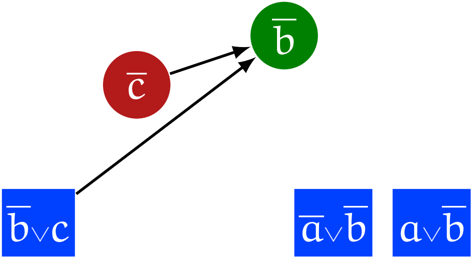
```
p cnf 3 3
-2  3 0
-1 -2 0
 1 -2 0
```

DRAT:

```
-3 0
```

LRAT:

```
4 -3 0 -1 2 3 0
```



rubenm@cmu.edu

Certified Checking: LRAT format

The LRAT format is syntactically similar to TraceCheck, however:

- The formula in **not** included in the proof
- Clause deletion support: ⟨pos⟩“ d "⟨clsidx⟩
- Can express a RAT step: use negative cls to denote resolvent

DIMACS:

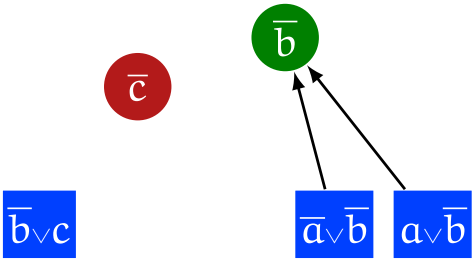
```
p cnf 3 3
-2  3 0
-1 -2 0
 1 -2 0
```

DRAT:

```
-3 0
```

LRAT:

```
4 -3 0 -1 2 3 0
```



Introduction

Proof Checking

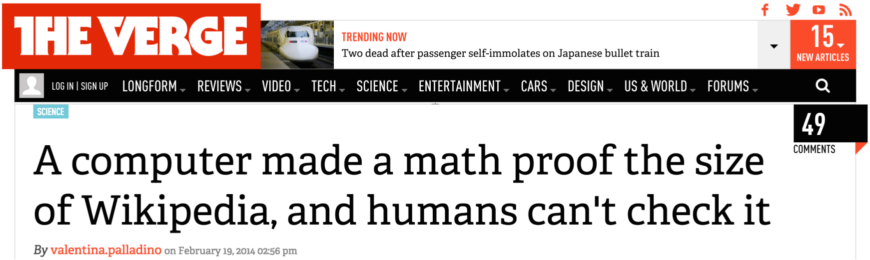
Proof Systems and Formats

Certified Checking

Applications

Conclusions

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

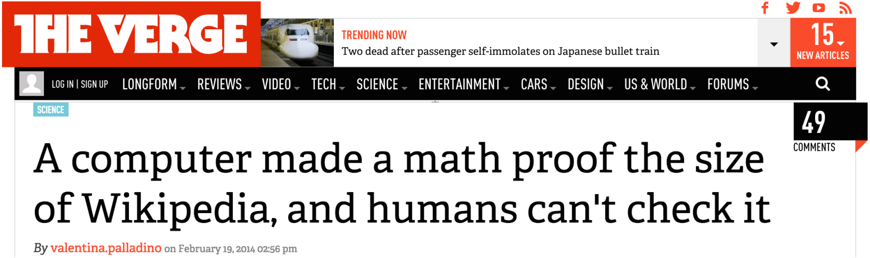
By valentina.palladino on February 19, 2014 02:56 pm

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

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

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 the tool DRAT-trim [SAT14]

Applications: SAT Competitions

DRAT proof logging supported by all the top-tier solvers:

- e.g. Lingeling, MiniSAT, Glucose, and CryptoMiniSAT
- Proof logging is mandatory since SAT Competition 2013
- Formally-verified checking since SAT Competition 2017

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
```

Introduction

Proof Checking

Proof Systems and Formats

Certified Checking

Applications

Conclusions

Many options in DRAT-trim

```
usage: drat-trim [INPUT] [<PROOF>] [<option> ...]
  -h            print this command line option summary
  -c CORE       prints the unsatisfiable core to CORE
  -a ACTIVE     prints the active clauses to ACTIVE
  -l DRAT       prints the core lemmas to DRAT
  -L LRAT       prints the core lemmas to LRAT
  -r TRACE      prints resolution graph to TRACE
  -t <lim>      time limit in seconds (default 20000)
  -u            default unit propagation (no core)
  -f            forward mode for UNSAT
  -v            more verbose output
  -b            show progress bar
  -O            optimize proof till fixpoint
  -C            compress core lemmas (emit binary proof)
  -i            force binary proof parse mode
  -w            suppress warning messages
  -W            exit after first warning
  -p            run in plain mode (no deletion)
```

Conclusions

Verification of proofs of unsatisfiability is now mature:

- Practically all state-of-the-art SAT solvers support it;
- There exist formally-verified checkers in ACL2, Coq, Isabelle;
- Proofs exist of recently solved long-standing open problems;
- The SAT Competitions now require proof emission;
- The overhead of certification is reasonable.

Challenges:

- How to reduce the size of proofs on disk and in memory?
- What information can be mined from proofs?
- How to effectively deal with Gaussian elimination, cardinality resolution, and pseudo-Boolean reasoning?