

Verifying Automated Reasoning Results

Marijn J.H. Heule

**Carnegie
Mellon
University**

`http://www.cs.cmu.edu/~mheule/15816-f19/`
`https://github.com/marijnheule/proof-demo`
Automated Reasoning and Satisfiability, October 10, 2019

Outline

Introduction

Proof Checking

Proof Systems and Formats

Certified Checking

Media and Applications

Conclusions

Introduction

Proof Checking

Proof Systems and Formats

Certified Checking

Media and Applications

Conclusions

Automated Reasoning Has Many Applications



formal verification



security



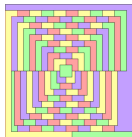
bioinformatics



planning and
scheduling



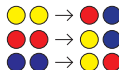
train safety



automated
theorem proving



exploit
generation



term rewriting
termination

encode



SAT/SMT solver

decode



Certifying Satisfiability and Unsatisfiability

- Certifying **satisfiability** of a formula is easy:

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$

Certifying Satisfiability and Unsatisfiability

- Certifying **satisfiability** of a formula is easy:

- Just consider a **satisfying assignment**: $x\bar{y}z$

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$

- We can easily check that the assignment is satisfying:
Just check for every clause if it has a satisfied literal!

Certifying Satisfiability and Unsatisfiability

■ Certifying **satisfiability** of a formula is easy:

- Just consider a **satisfying assignment**: $x\bar{y}z$

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{y} \vee \bar{z})$$

- We can easily check that the assignment is satisfying:
Just check for every clause if it has a satisfied literal!

■ Certifying **unsatisfiability** is not so easy:

- If a formula has n variables, there are 2^n possible assignments.
- ➡ Checking whether **every** assignment falsifies the formula is **costly**.
- More compact certificates of unsatisfiability are desirable.
 - ➡ **Proofs**

What Is a Proof in SAT?

- In general, a **proof** is a **string** that **certifies the unsatisfiability** of a formula.
 - Proofs are **efficiently** (usually **polynomial-time**) **checkable**...

What Is a Proof in SAT?

- In general, a **proof** is a **string** that **certifies the unsatisfiability** of a formula.
 - Proofs are **efficiently** (usually **polynomial-time**) **checkable**...
... but can be of exponential size with respect to a formula.

What Is a Proof in SAT?

- In general, a **proof** is a **string** that **certifies the unsatisfiability** of a formula.
 - Proofs are **efficiently** (usually **polynomial-time**) **checkable**...
... but can be of exponential size with respect to a formula.

- **Example:** Resolution proofs

- A **resolution proof** is a sequence C_1, \dots, C_m of clauses.
- Every clause is either contained in the formula or derived from two earlier clauses via the **resolution rule**:

$$\frac{C \vee x \quad \bar{x} \vee D}{C \vee D}$$

- C_m is the **empty clause** (containing no literals), denoted by \perp .
- There exists a resolution proof for every unsatisfiable formula.

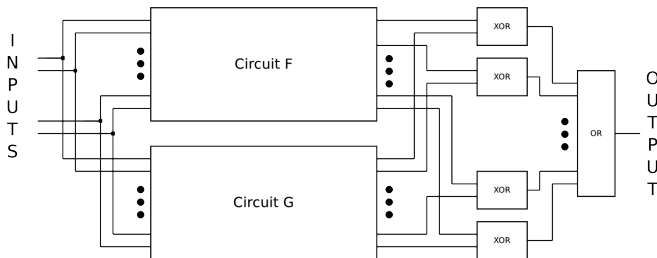
Motivation for Validating Proofs of Unsatisfiability

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.

Combinatorial Equivalence Checking

Chip makers use SAT to check the **correctness** of their designs. Equivalence checking involves comparing a specification with an implementation or an optimized with a non-optimized circuit.



Demo: Validating Results

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

Introduction

Proof Checking

Proof Systems and Formats

Certified Checking

Media and Applications

Conclusions

Resolution Rule and Resolution Chains

Resolution Rule

$$\frac{C \vee x \quad \bar{x} \vee D}{C \vee D}$$

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

Resolution Rule and Resolution Chains

Resolution Rule

$$\frac{C \vee x \quad \bar{x} \vee D}{C \vee D}$$

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

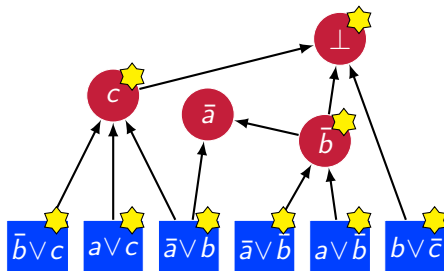
Example

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

Resolution Proofs versus Clausal Proofs

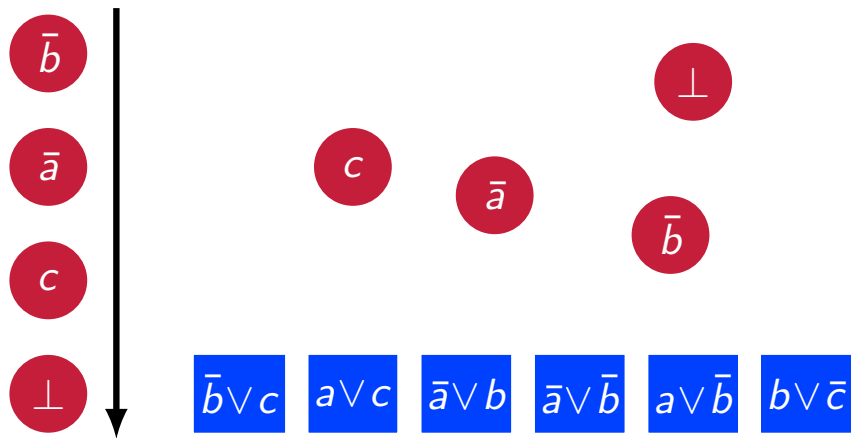
Consider $F := (\bar{b} \vee c) \wedge (a \vee c) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee \bar{b}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c})$

A resolution graph of F is:

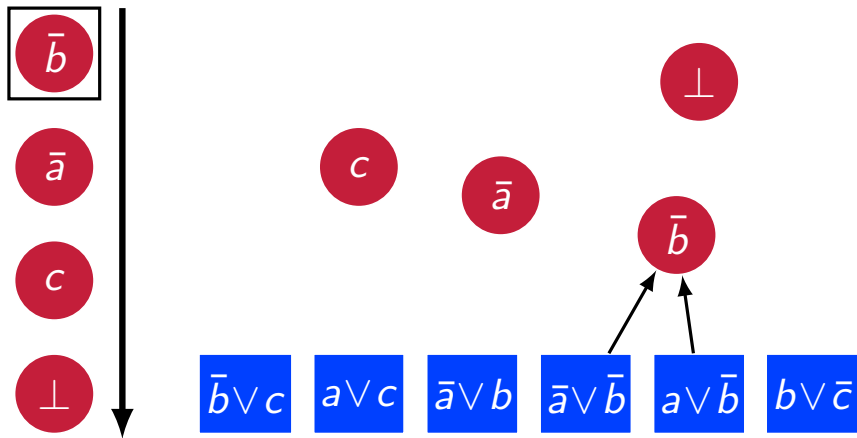


- A **resolution proof** consists of all nodes and edges of the resolution graph
- Graphs from SAT solvers have ~ 400 incoming edges per node
 - Resolution proof logging can heavily increase memory usage ($\times 100$)
- A **clausal proof** is a list of all nodes sorted by topological order
- Clausal proofs are easy to emit and relatively small
 - Clausal proof checking requires to reconstruct the edges (costly)

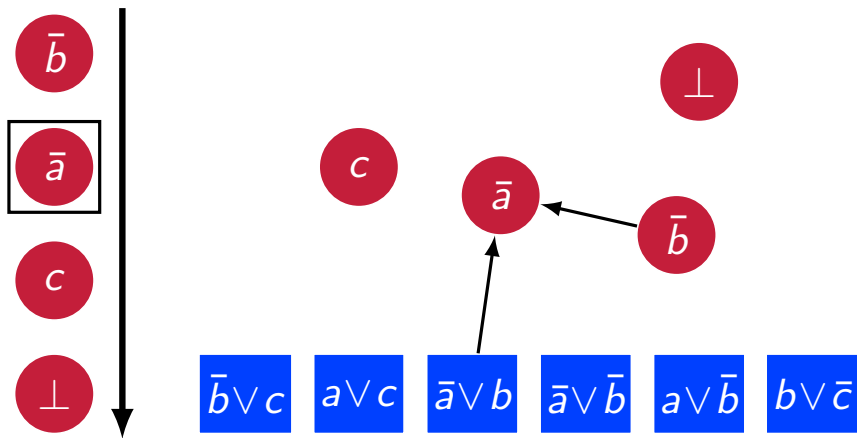
Clausal Proof: Checker has to reconstruct resolution edges



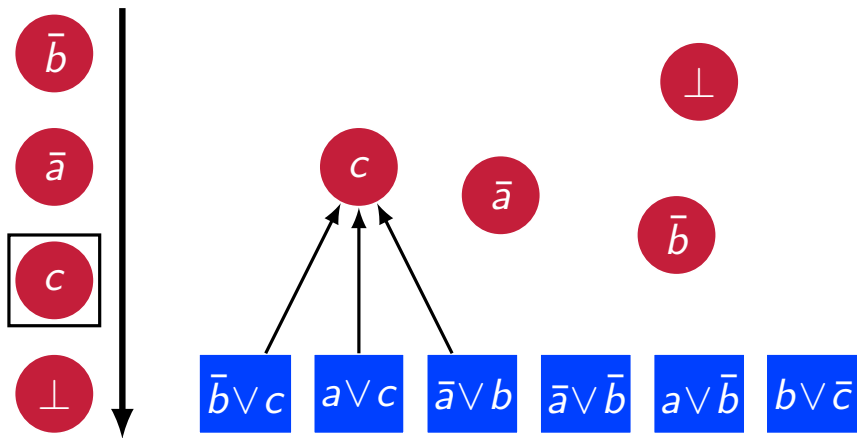
Clausal Proof: Checker has to reconstruct resolution edges



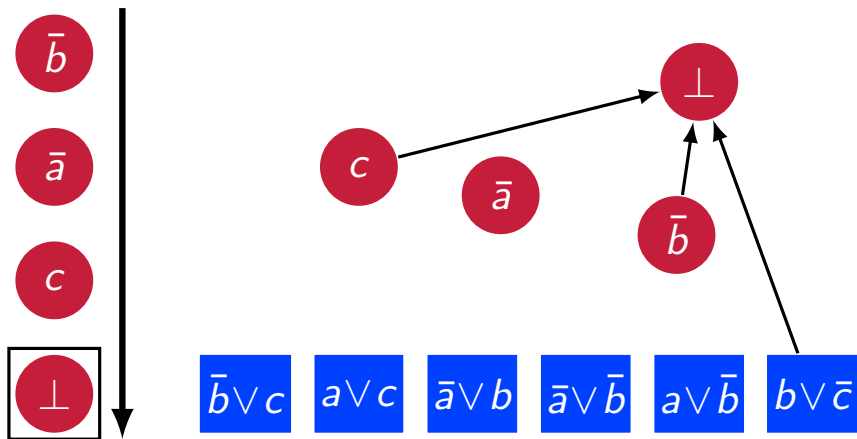
Clausal Proof: Checker has to reconstruct resolution edges



Clausal Proof: Checker has to reconstruct resolution edges



Clausal Proof: Checker has to reconstruct resolution edges

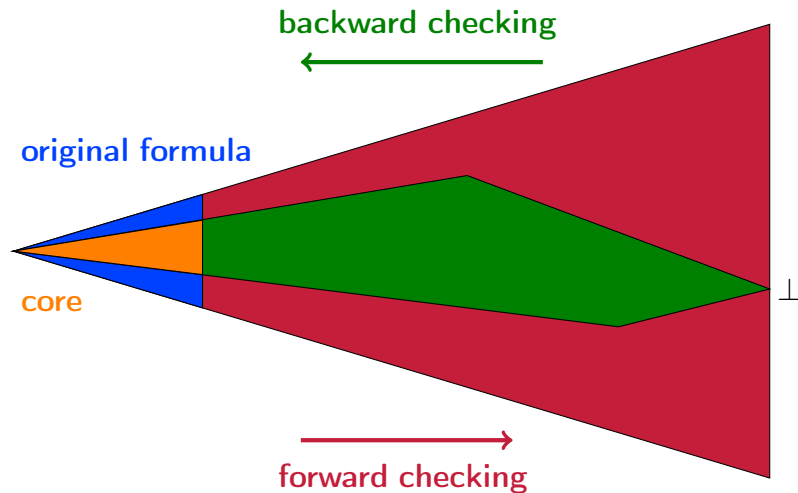


Reverse Unit Propagation

How to find/reconstruct the edges efficiently?

- **Unit propagation** (UP) satisfies unit clauses by assigning their literal to true (until fixpoint or a conflict).
- Given an assignment α , $F|_{\alpha}$ denotes a formula F without the clauses satisfied by α and without the literals falsified by α .
- Let F be a formula, C a clause, and α the smallest assignment that falsifies C . C is **implied by F via UP** (denoted by $F \vdash_1 C$) if UP on $F|_{\alpha}$ results in a conflict.
- $F \vdash_1 C$ is also known as **Reverse Unit Propagation** (RUP).
- **Learned clauses** in CDCL solvers are RUP clauses.
- RUP typically summarizes **dozens to hundreds of resolution steps**.

Forward vs Backward Proof Checking



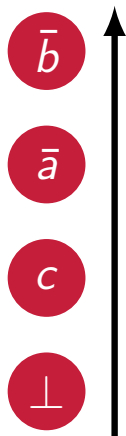
Improvement I: Backwards Checking

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.



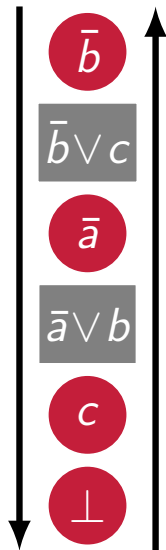
Improvement II: Clause Deletion

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.



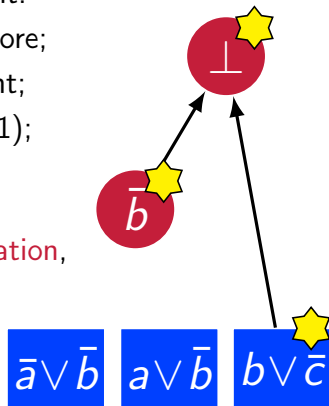
Improvement III: Core-first Unit Propagation

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.

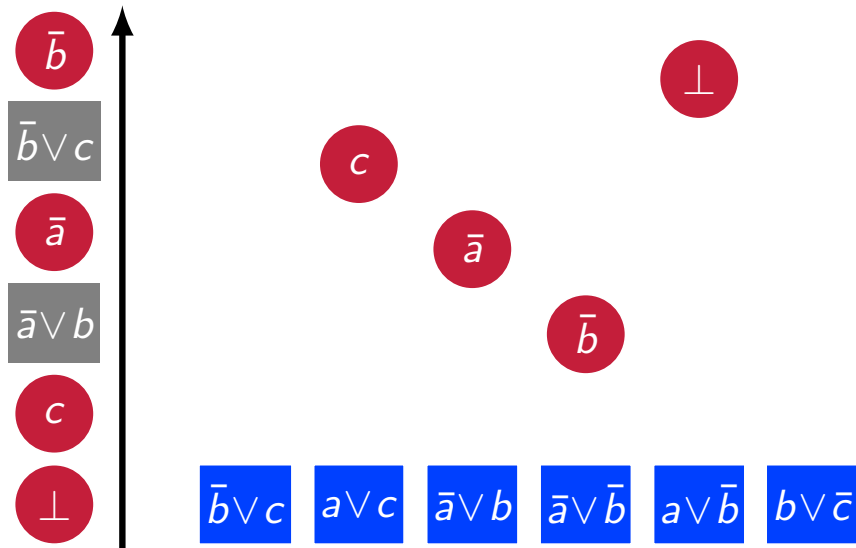
The variant, called **Core-first Unit Propagation**, can reduce checking costs considerably.

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



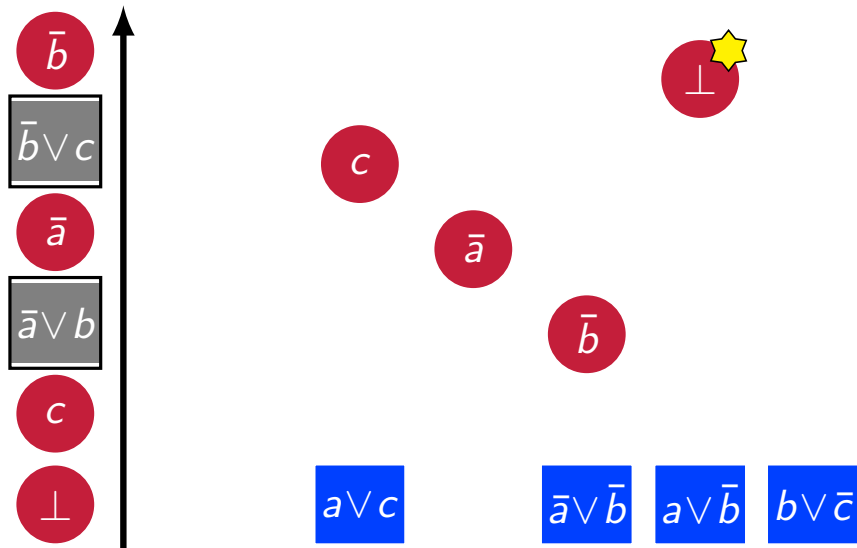
Also, the resulting core and proof are smaller

Checking: Backwards + Core-first + Deletion



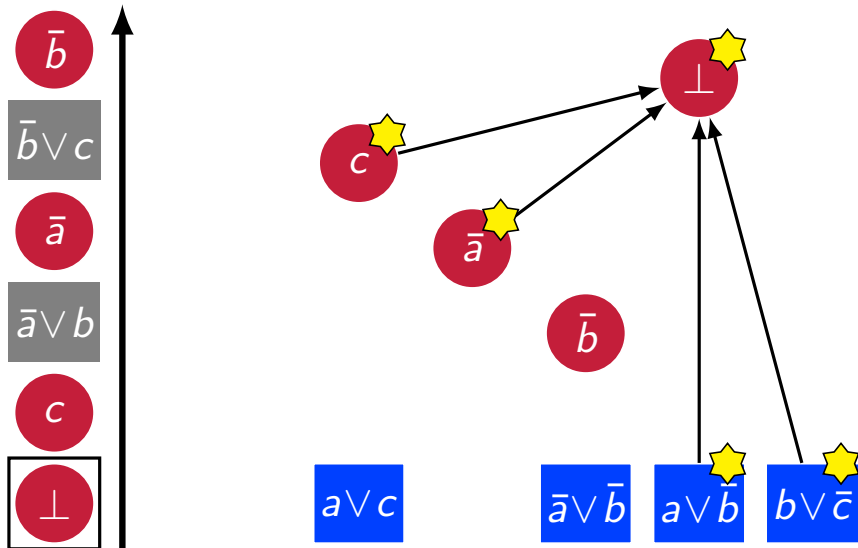
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



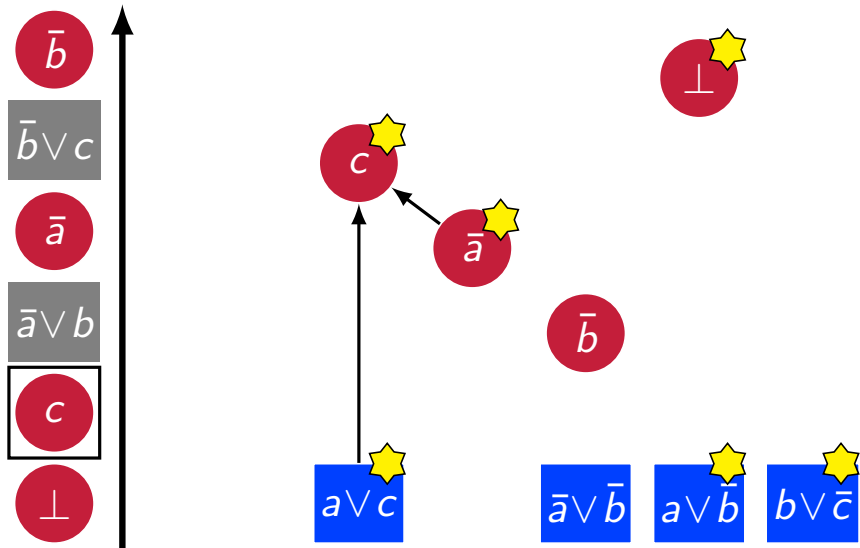
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



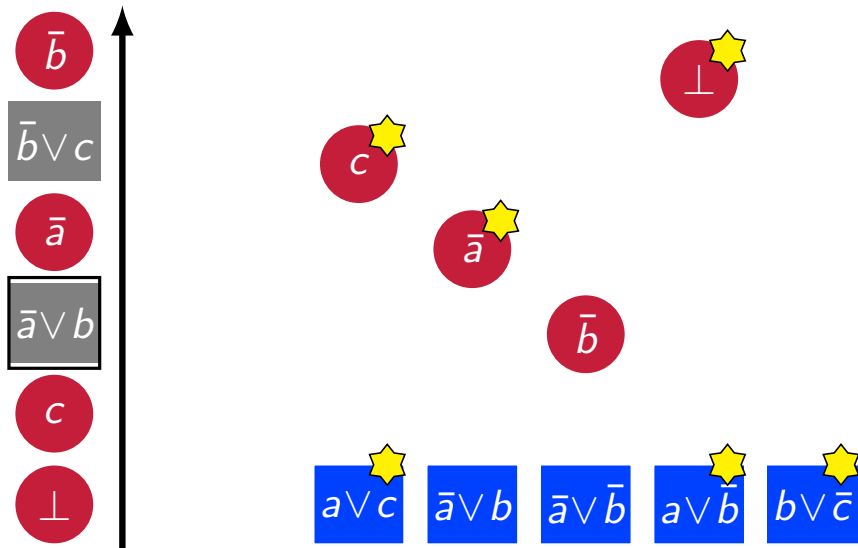
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



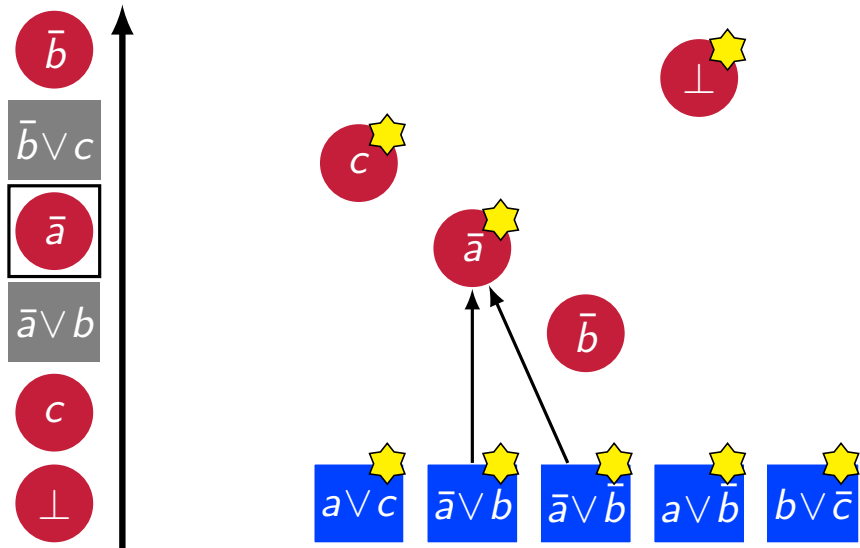
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



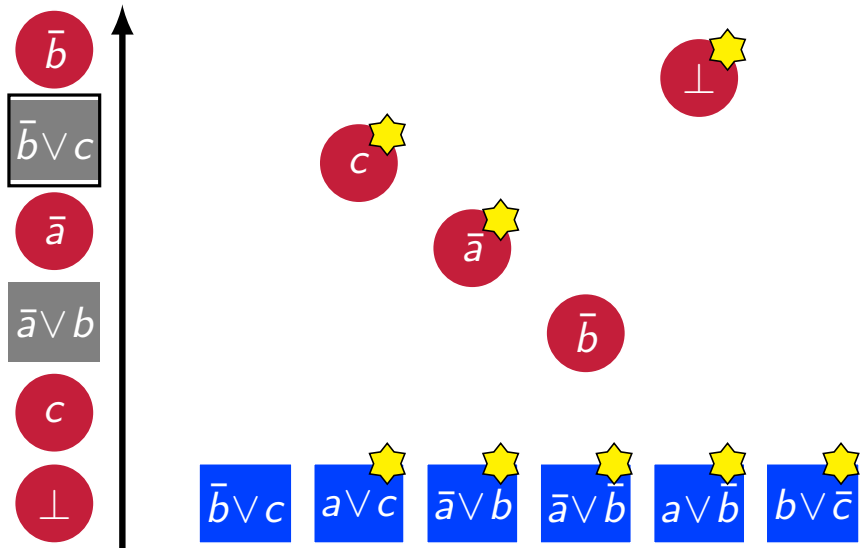
Core-first unit propagation results in smaller cores and proofs

Checking: Backwards + Core-first + Deletion



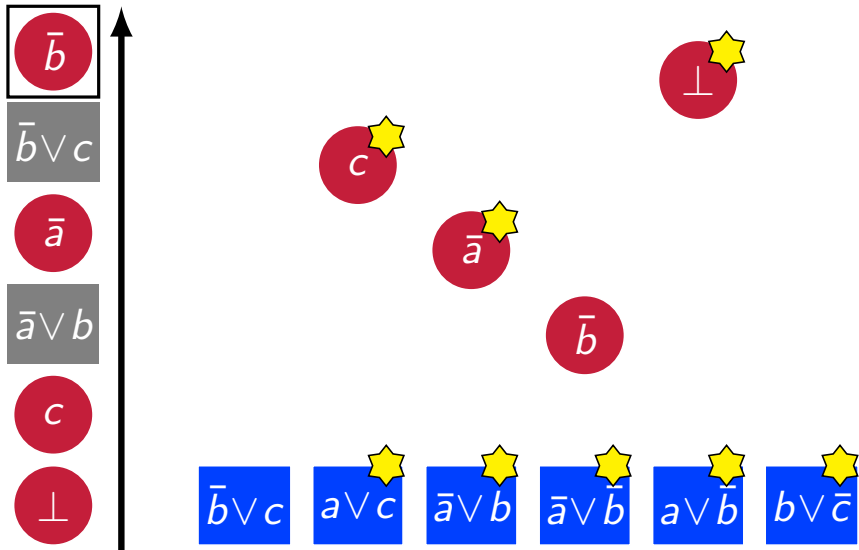
Core-first unit propagation results in smaller cores and proofs

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.

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**.

Popular **example** of a clausal proof system: **DRAT**

- DRAT allows the addition of **RATs** (defined below) 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 (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**.

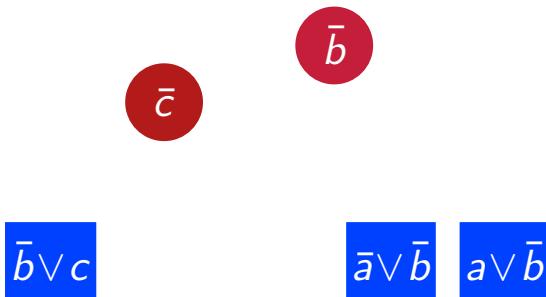
Popular **example** of a clausal proof system: **DRAT**

- DRAT allows the addition of **RATs** (defined below) 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)

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

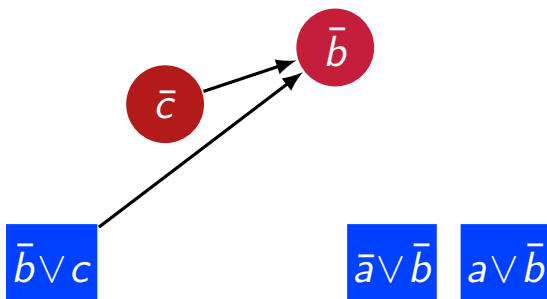
DRAT Example

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



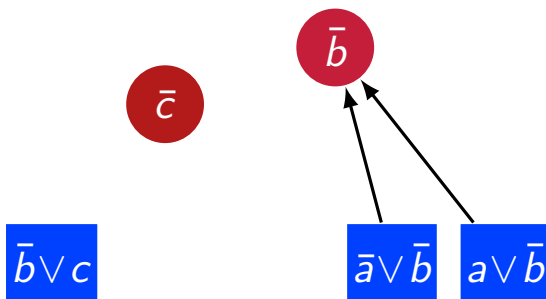
DRAT Example

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



DRAT Example

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



Demo: DRAT step

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

Introduction

Proof Checking

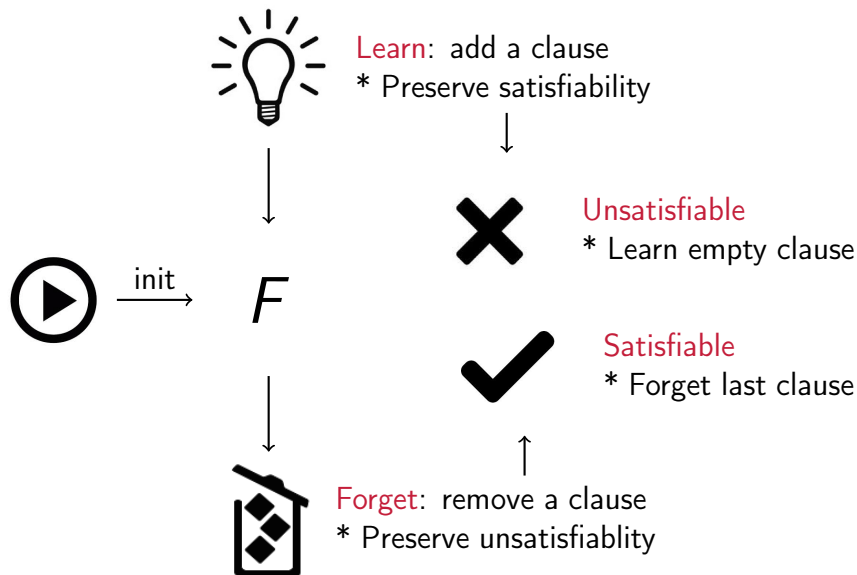
Proof Systems and Formats

Certified Checking

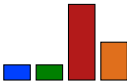
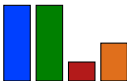
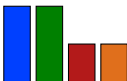
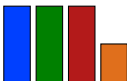
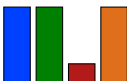

Media and Applications

Conclusions

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

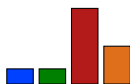


Ideal Properties of a Proof System for SAT Solvers

		Resolution Proofs Zhang and Malik, 2003 Van Gelder, 2008; Biere, 2008
Easy to Emit		Clausal Proofs Goldberg and Novikov, 2003 Van Gelder, 2008
Compact		Clausal proofs + deletion Heule, Hunt, Jr., Wetzler [STVR'14]
Checked Efficiently		Optimized clausal proof checker Heule, Hunt, Jr., and Wetzler [FMCAD'13]
Expressive		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

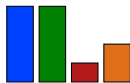


Resolution Proofs

Zhang and Malik, 2003

Van Gelder, 2008; Biere, 2008

Compact

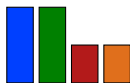


Clausal Proofs

Goldberg and Novikov, 2003

Van Gelder, 2008

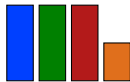
Checked Efficiently



Clausal proofs + deletion

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

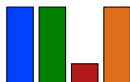
Expressive



Optimized clausal proof checker

Heule, Hunt, Jr., and Wetzler [FMCAD'13]

Verified



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

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

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

$\langle \text{trace} \rangle = \{ \langle \text{clause} \rangle \}$
 $\langle \text{clause} \rangle = \langle \text{pos} \rangle \langle \text{literals} \rangle \langle \text{clsidx} \rangle$
 $\langle \text{literals} \rangle = " * " \mid \{ \langle \text{lit} \rangle \} "0"$
 $\langle \text{clsidx} \rangle = \{ \langle \text{pos} \rangle \} "0"$
 $\langle \text{lit} \rangle = \langle \text{pos} \rangle \mid \langle \text{neg} \rangle$
 $\langle \text{pos} \rangle = "1" \mid "2" \mid \dots \mid \langle \text{maxidx} \rangle$
 $\langle \text{neg} \rangle = " - " \langle \text{pos} \rangle$

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.

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

TraceCheck is readable and resolution chains make it relatively compact

The clauses **1** to **6** are **input clauses**

Clause **7** is the resolvent of **4** and **5**:

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

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

■ $\perp := (\bar{b}) \diamond (c) \diamond (b \vee \bar{c})$

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	

 \equiv

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

 \equiv

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	

 \equiv

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.

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

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

$\langle \text{proof} \rangle = \{ \langle \text{lemma} \rangle \}$
 $\langle \text{lemma} \rangle = \langle \text{delete} \rangle \{ \langle \text{lit} \rangle \} \text{"0"}$
 $\langle \text{delete} \rangle = \text{""} \mid \text{"d"}$
 $\langle \text{lit} \rangle = \langle \text{pos} \rangle \mid \langle \text{neg} \rangle$
 $\langle \text{pos} \rangle = \text{"1"} \mid \text{"2"} \mid \dots \mid \langle \text{maxidx} \rangle$
 $\langle \text{neg} \rangle = \text{"-"} \langle \text{pos} \rangle$

-2	0
3	0
0	

$$\begin{aligned} E \wedge (b) &\vdash_1 \perp \\ E \wedge (\bar{b}) \wedge (\bar{c}) &\vdash_1 \perp \\ E \wedge (\bar{b}) \wedge (c) &\vdash_1 \perp \end{aligned}$$

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 $\text{lit}_1 \dots \text{lit}_{k-1}$ 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

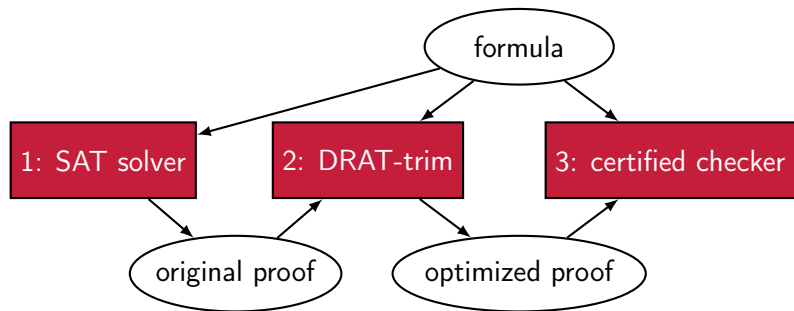
Proof Systems and Formats

Certified Checking

Media and 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: ACL2-Based, SAT Proof Checker

We developed a mechanically verified, ACL2-based, proof checker for proofs of unsatisfiability.

Given files containing:

- the initial conjecture, as a set of clauses, and
 - an ordered list of proof steps ending with the empty clause,
- our mechanically verified, SAT proof checker attempts to confirm the veracity of each proof step.

Parsing is hard, while writing is easy.

- after verification, we emit a conjecture that can be compared to the initial conjecture.
- a common tool, such as `diff`, can do the comparison.

Certified Checking: Proof Claims

Basic Soundness.

```
(implies (and (formula-p formula)
              (refutation-p proof formula))
         (not (satisfiable formula))))
```

Soundness Plus Formula Confirmation.

```
(let ((formula
      (mv-nth 1 (proved-formula cnf-file clrat-file
                               chunk-size debug
                               nil ; incomplete-okp
                               ctx state))))
      (implies formula
                (not (satisfiable formula)))))
```

; Print proved formula, to diff against input formula

Certified Checking: Eliminate Complexity

Certified proof checking challenges:

- backward checking is complex and heavy on memory;
- unit propagation is expensive.

We eliminate both challenges by modifying the proof:

- an efficient unverified tool removes the redundancy, making **forward checking** as fast as backward checking;
- searching for units is replaced by **hints** to locate units;
- the modified proofs are not much larger;
- we do not need to trust the unverified tool.

Certified Checking: LRAT format

The LRAT format is syntactically similar to TraceCheck, however:

- The formula is **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
----------	----	---	-----------	----------	----------	----------



Certified Checking: LRAT format

The LRAT format is syntactically similar to TraceCheck, however:

- The formula is **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:

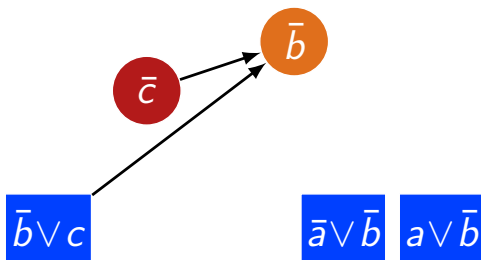
```
p cnf 3 3
-2 3 0
-1 -2 0
1 -2 0
```

DRAT:

```
-3 0
```

LRAT:

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



Certified Checking: LRAT format

The LRAT format is syntactically similar to TraceCheck, however:

- The formula is **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:

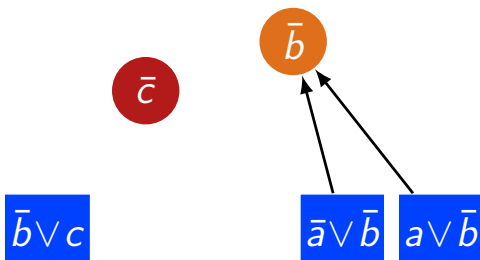
```
p cnf 3 3
-2 3 0
-1 -2 0
1 -2 0
```

DRAT:

```
-3 0
```

LRAT:

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



Introduction

Proof Checking

Proof Systems and Formats

Certified Checking

Media and Applications

Conclusions

Media: The Largest Math Proof Ever

engadget

THE NEW REDDIT

comments other discussions (5)

Mathematics

nature

International weekly journal of science

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

Archive > Volume 534 > Issue 7605 > News > Article

Two-hundred-terabyte

19 days ago by [CryptoBeer](#)

265 comments share

NATURE | NEWS



Slashdot

Stories

Two-hundred-terabyte maths proof is largest ever

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 [\(phys.org\)](#)



Posted by [BeauHD](#) on Monday May 30, 2016 @08:10PM from the red-pill-and-blue-pill dept.



143

THE CONVERSATION

Academic rigour, journalistic flair

76 comments

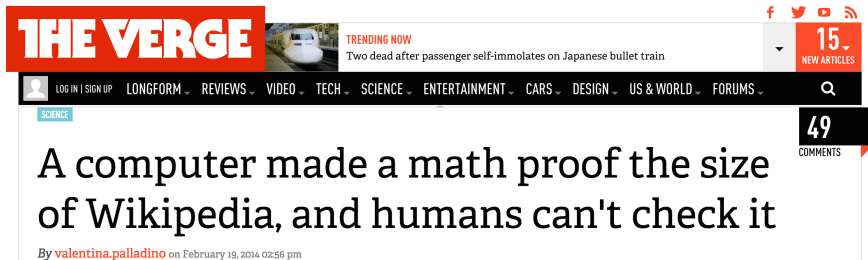


[Collqteral](#) May 27, 2016 +2

200 Terabytes. Thats about 400 PS4s.

SPIEGEL ONLINE

Applications: Erdős Discrepancy Conjecture

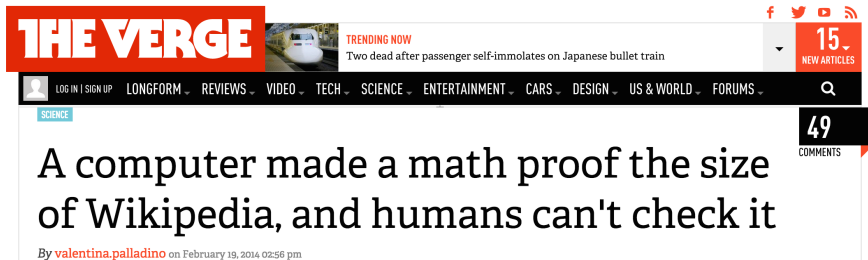


Erdős Discrepancy Conjecture was recently solved using SAT.

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 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
```

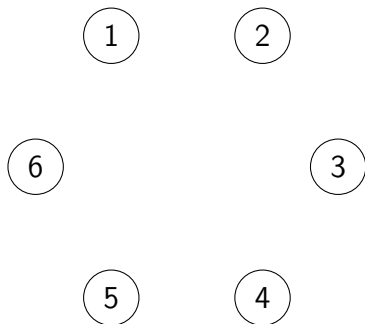
Applications: Ramsey Numbers

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]

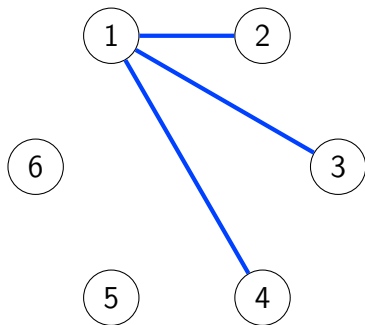
Applications: Ramsey Numbers

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]

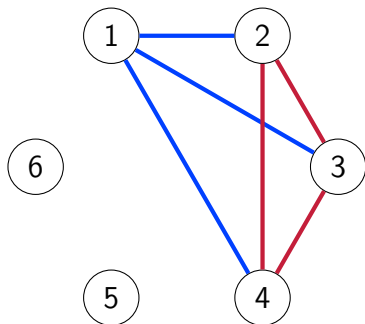
Applications: Ramsey Numbers

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]

Demo: Certifying DRAT Proofs

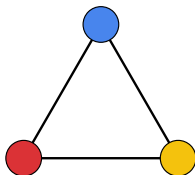
```
git clone https://github.com/marijnheule/proof-demo
```

Chromatic Number of the Plane

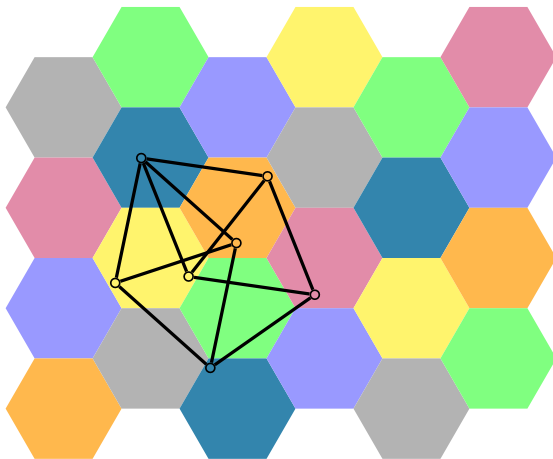
The Hadwiger-Nelson problem:

How many colors are required to color the plane such that each pair of points that are exactly 1 apart are colored differently?

The answer must be three or more because three points can be mutually 1 apart—and thus must be colored differently.



Bounds since the 1950s

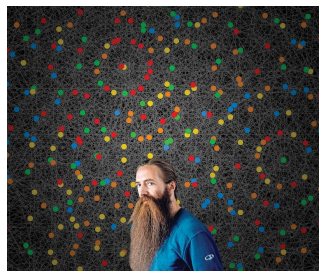


- The Moser Spindle graph shows the lower bound of 4
- A coloring of the plane showing the upper bound of 7

First progress in decades

Recently enormous progress:

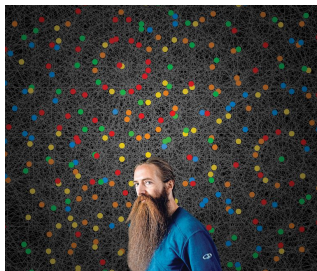
- Lower bound of 5 [DeGrey '18] based on a 1581-vertex graph
- This breakthrough started a polymath project
- Improved bounds of the fractional chromatic number of the plane



First progress in decades

Recently enormous progress:

- Lower bound of 5 [DeGrey '18] based on a 1581-vertex graph
- This breakthrough started a polymath project
- Improved bounds of the fractional chromatic number of the plane



Quanta magazine | Physics Mathematics

業餘數學家為一道填色難題帶來突破！
2018/4/26 • TNL • 四色定理、填色難題、數學

Раскраска для математиков
Как покрасить плоскость?

WIRED

Marijn Heule, a computer scientist at the University of Texas, Austin, found one with just 874 vertices. Yesterday he lowered this number to 826 vertices.

We found smaller graphs with SAT:

- 874 vertices on April 14, 2018
- 803 vertices on April 30, 2018
- 610 vertices on May 14, 2018

Propositional Proofs for Graph Validation and Shrinking

Checking that a unit-distance graph has chromatic number 5:

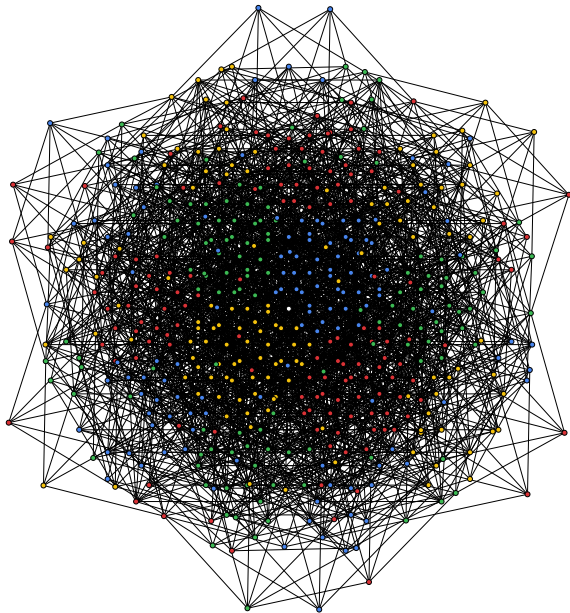
- Show that there exists a 5-coloring
- While there is no 4-coloring (formula is UNSAT)
- Unsatisfiable core represents a subgraph

SAT solvers find **short proofs** of unsatisfiability for these formulas

Proof minimization techniques allow further reduction

Combining the techniques allows finding **much smaller graphs**

Proof Minimization: 529 Vertices [Heule 2019]



Introduction

Proof Checking

Proof Systems and Formats

Certified Checking

Media and 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?