Efficient and Verified Checking of Unsatisfiability Proofs

Marijn J.H. Heule
Warren A. Hunt Jr.
Nathan Wetzler

The University of Texas at Austin

May 9, 2013

The Satisfiability Approach

The Satisfiability (SAT) approach:

- **Encode** a problem into propositional logic;
- **Solve** the resulting formula using a SAT solver;
- **Decode** the solution to obtain the answer.

Success of the SAT approach:

- SAT solvers have become very efficient search engines.
- Several problems can be solved faster by SAT solvers than any other existing technique.
- SAT solvers are used in lots of industries.

Bugs in SAT solvers?

Literature:

- Many SAT, SMT, and QBF solvers were found buggy [BrummayerBiere '09, BrummayerLonsingBiere '10]

Bugs in SAT solvers?

Literature:

- Many SAT, SMT, and QBF solvers were found buggy [BrummayerBiere '09, BrummayerLonsingBiere '10]

Even winners of competitions show "discrepancies":

- Hardware Model Checking Competition 2011 and 2012

Bugs in SAT solvers?

Literature:

- Many SAT, SMT, and QBF solvers were found buggy [BrummayerBiere '09, BrummayerLonsingBiere '10]

Even winners of competitions show "discrepancies":

- Hardware Model Checking Competition 2011 and 2012

Some bugs cannot be detected using existing methods

- Blocked clause addition is "experimentally correct"

Bugs in SAT solvers?

Literature:

- Many SAT, SMT, and QBF solvers were found buggy [BrummayerBiere '09, BrummayerLonsingBiere '10]

Even winners of competitions show "discrepancies":

- Hardware Model Checking Competition 2011 and 2012

Some bugs cannot be detected using existing methods

- Blocked clause addition is "experimentally correct"

Bugs in solvers can be introduced by the compiler

- gcc 4.5.x (-O2) incorrectly compiles some solvers

Focus and Accomplishments

We want to increase confidence
in state-of-the-art satisfiability solvers:

- We developed a fast UNSAT proof checker in C.
  The checking costs are reduced significantly
  and comparable to solving times.
- We developed a more general proof format
  can supports all existing SAT solving techniques.
- We mechanically verified an UNSAT proof checker
  using the ACL2 Theorem Prover.

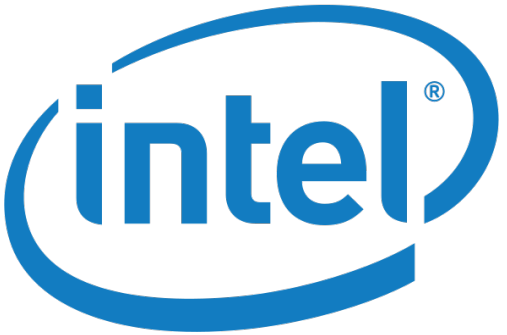
Outline

Outline

Applications of Satisfiability (1)

Applications:

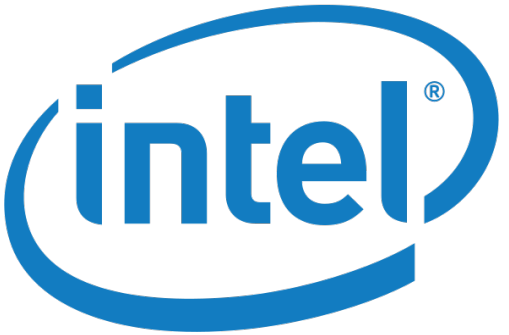
- Software Verification
- Hardware Verification
- Planning
- Scheduling
- Optimal Control
- Protocol Design
- Routing
- Combinatorial problems
- Equivalence Checking



Applications of Satisfiability (2)

Applications:

- Software Verification
- Hardware Verification



What is the typical answer?

`unsatisfiable`

What does it mean?

`no bugs`

Running example

Is the formula below satisfiable or unsatisfiable?

$$
\begin{array}{l}
(\ A \vee\ \ B \vee -C) \wedge \\
(-A \vee -B \vee\ \ C) \wedge \\
(\ B \vee\ \ C \vee -D) \wedge \\
(-B \vee -C \vee\ \ D) \wedge \\
(\ A \vee\ \ C \vee\ \ D) \wedge \\
(-A \vee -C \vee -D) \wedge \\
(\ A \vee -B \vee -D)
\end{array}
$$

Black box solver returns a model

Input: Boolean formula

↙

Black box solver

↘

Output: model (A ∧ -B ∧ C ∧ -D)

Checking models is easy

Checking model (A ∧ -B ∧ C ∧ -D):

Each clause must contain one literal from the solution

$$
\begin{aligned}
&(A \vee \ \ B \vee -C) \wedge \\
&(-A \vee -B \vee \ \ C) \wedge \\
&(B \vee \ \ C \vee -D) \wedge \\
&(-B \vee -C \vee \ \ D) \wedge \\
&(A \vee \ \ C \vee \ \ D) \wedge \\
&(-A \vee -C \vee -D) \wedge \\
&(A \vee -B \vee -D)
\end{aligned}
$$

Outline

Black box solver claims there is no model

Boolean formula

↙

Black box solver

↘

Output: `unsatisfiable`. What now?!?

Black box solver claims there is no model

Boolean formula

↙

Black box solver

↘

Output: `unsatisfiable`. What now?!?

Core SAT solving techniques:
Addition and removal of
redundant clauses

Black box solver claims there is no model

Boolean formula

↙

Black box solver

↘

Core SAT solving techniques:
Addition and removal of
redundant clauses

Construct proof of added clauses
[Goldberg & Novikov 2002]

Output: `unsatisfiable` and a sequence of added clauses

Redundant clauses

A clause is redundant w.r.t. a formula if adding it to the formula preserves satisfiability.

- For unsatisfiable formulas, all clauses can be added, including the empty clause ( ).

Redundant clauses

A clause is redundant w.r.t. a formula if adding it to the formula preserves satisfiability.

- For unsatisfiable formulas, all clauses can be added, including the empty clause ( ).

A clause is redundant w.r.t. a formula if removing it from the formula preserves unsatisfiability.

- For satisfiable formulas, all clauses can be removed.

Redundant clauses

A clause is redundant w.r.t. a formula if adding it to the formula preserves satisfiability.

- For unsatisfiable formulas, all clauses can be added, including the empty clause ( ).

A clause is redundant w.r.t. a formula if removing it from the formula preserves unsatisfiability.

- For satisfiable formulas, all clauses can be removed.

Challenge regarding redundant clauses:

- How to check redundancy in polynomial time?

Verify added clauses by proof checker

Boolean formula

Black box solver

Proof checker

Output: `unsatisfiable` and a sequence of added clauses

Extended example contains no models

The formula with (-A ∨ B ∨ D) is unsatisfiable

$$
\begin{array}{l}
(A \vee B \vee -C) \wedge \\
(-A \vee -B \vee C) \wedge \\
(B \vee C \vee -D) \wedge \\
(-B \vee -C \vee D) \wedge \\
(A \vee C \vee D) \wedge \\
(-A \vee -C \vee -D) \wedge \\
(A \vee -B \vee -D) \wedge \\
(-A \vee B \vee D)
\end{array}
$$

Reverse Unit Propagation (RUP) proof of example

Boolean formula:

$$
\begin{aligned}
&(\ A \vee\ \ B \vee -C) \wedge \\
&(-A \vee -B \vee\ \ C) \wedge \\
&(\ B \vee\ \ C \vee -D) \wedge \\
&(-B \vee -C \vee\ \ D) \wedge \\
&(\ A \vee\ \ C \vee\ \ D) \wedge \\
&(-A \vee -C \vee -D) \wedge \\
&(\ A \vee -B \vee -D) \wedge \\
&(-A \vee\ \ B \vee\ \ D)
\end{aligned}
$$

RUP proof:

$$
\begin{aligned}
&(-A \vee\ \ B) \\
&(-A) \\
&(\ B) \\
&(\ \ )
\end{aligned}
$$

Reverse Unit Propagation (RUP) proof of example

Boolean formula:

$$
\begin{aligned}
&(A \vee B \vee \neg C) \wedge \\
&(\neg A \vee \neg B \vee C) \wedge \\
&(B \vee C \vee \neg D) \wedge \\
&(\neg B \vee \neg C \vee D) \wedge \\
&(A \vee C \vee D) \wedge \\
&(\neg A \vee \neg C \vee \neg D) \wedge \\
&(A \vee \neg B \vee \neg D) \wedge \\
&(\neg A \vee B \vee D) \wedge \\
&(\neg A \vee B)
\end{aligned}
$$

RUP proof:

$$
\begin{aligned}
&(\neg A) \\
&(B) \\
&( )
\end{aligned}
$$

Reverse Unit Propagation (RUP) proof of example

Boolean formula:

$$
\begin{aligned}
&(A \vee B \vee \neg C) \wedge \\
&(\neg A \vee \neg B \vee C) \wedge \\
&(B \vee C \vee \neg D) \wedge \\
&(\neg B \vee \neg C \vee D) \wedge \\
&(A \vee C \vee D) \wedge \\
&(\neg A \vee \neg C \vee \neg D) \wedge \\
&(A \vee \neg B \vee \neg D) \wedge \\
&(\neg A \vee B \vee D) \wedge \\
&(\neg A \vee B)
\end{aligned}
$$

RUP proof:

$$
\begin{aligned}
&(\neg A) \\
&(B) \\
&( )
\end{aligned}
$$

Outline

First enhancement: Add clause deletion information

Input: Boolean formula

↘

Proof checker

Main disadvantage

RUP checking is very costly on large proofs (Slower than solving)

↗

Output: sequence of added clauses

First enhancement: Add clause deletion information

Input: Boolean formula

↘

Proof checker

Main disadvantage

RUP checking is very costly on large proofs (Slower than solving)

Our proposed fix

Add deletion info

↗

Output: sequence of added *and deleted clauses*

Clause deletion information from solver to checker

Input: Boolean formula

Black box solver

`unsatisfiable`?

Proof checker

Output: sequence of added *and deleted* clauses

Reverse Unit Propagation (RUP) proofs with deletion

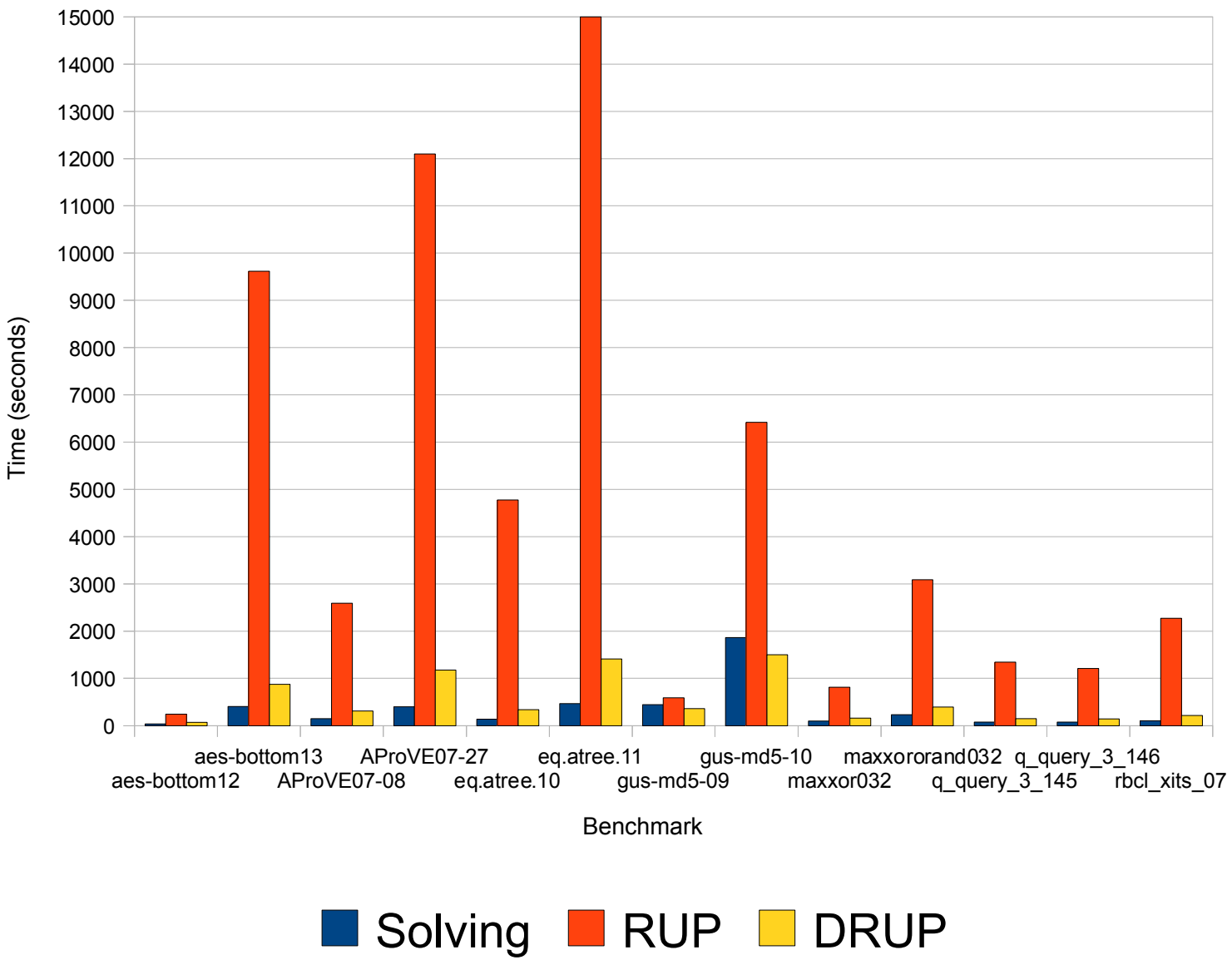
Boolean formula:

$$
\begin{array}{l}
(\ A \vee\ \ B \vee -C) \wedge \\
(-A \vee -B \vee\ \ C) \wedge \\
(\ B \vee\ \ C \vee -D) \wedge \\
(-B \vee -C \vee\ \ D) \wedge \\
(\ A \vee\ \ C \vee\ \ D) \wedge \\
(-A \vee -C \vee -D) \wedge \\
(\ A \vee -B \vee -D) \wedge \\
(-A \vee\ \ B \vee\ \ D)
\end{array}
$$

DRUP proof:

$$
\begin{array}{l}
\ \ (-A \vee\ \ B) \\
\mathtt{d}\ (-A \vee\ \ B \vee\ \ D) \\
\ \ (-A) \\
\mathtt{d}\ (-A \vee\ \ B) \\
\mathtt{d}\ (-A \vee -B \vee\ \ C) \\
\mathtt{d}\ (-A \vee -C \vee -D) \\
\ \ (\ B) \\
\ \ (\ \ )
\end{array}
$$

Reduced checking costs using DRUP



Outline

Second enhancement: Generalized redundancy

Input: Boolean formula

↘

Proof checker

Another problem

Not all techniques are covered by RUP (No full verification)

↗

Output: sequence of added RUP clauses

Second enhancement: Generalized redundancy

Input: Boolean formula

↘

Proof checker

Another problem

Not all techniques are covered by RUP (No full verification)

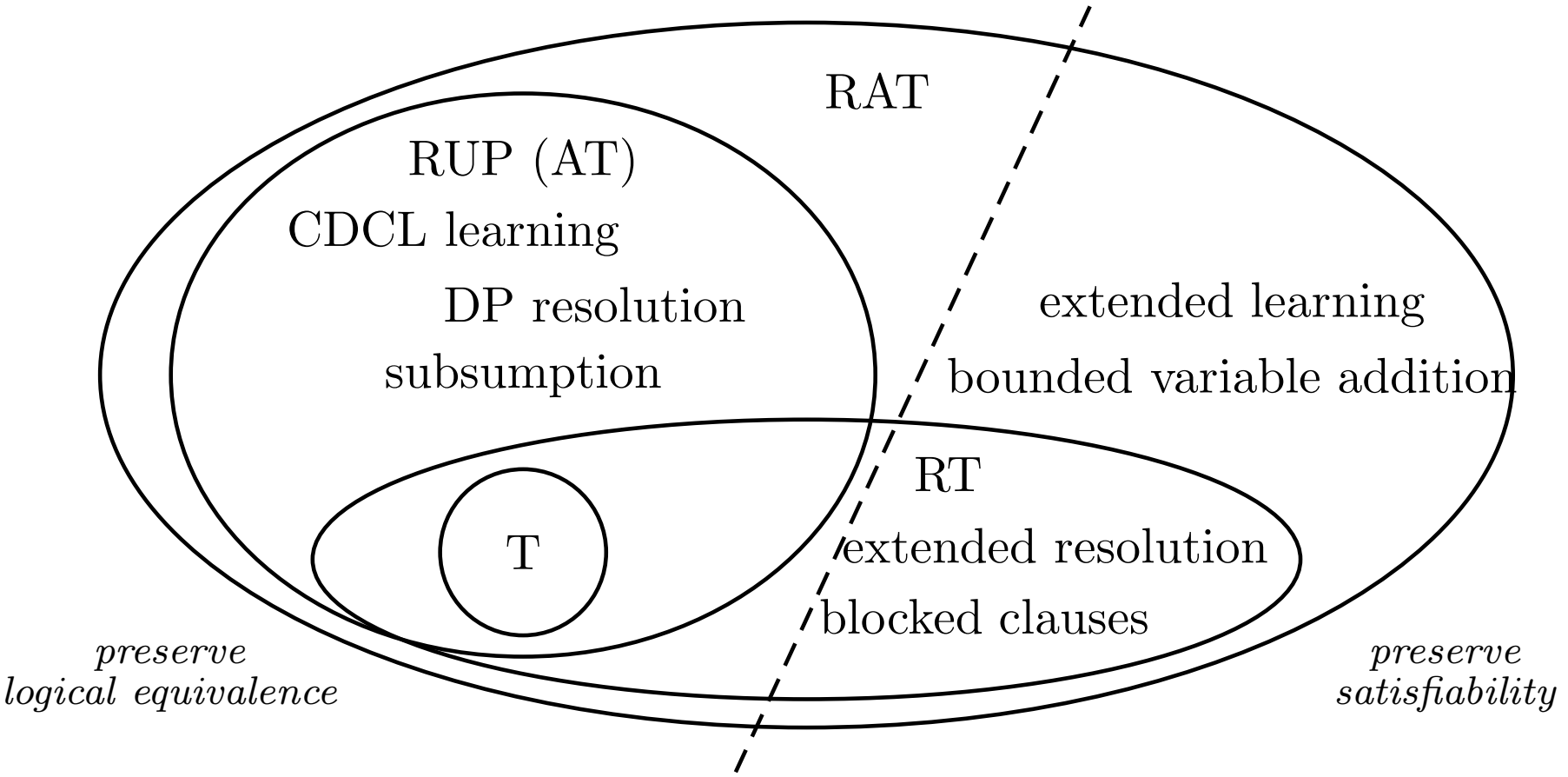
Our proposed fix

Emit clauses with the RAT redundancy

↗

Output: sequence of added RAT clauses

Relationship between Redundancy Properties



All known techniques can be expressed using RAT

Only RAT can check some techniques

Bounded Variable Addition (BVA) is an effective preprocessing technique on several families of benchmarks.

BVA cannot be checked using RUP

benchmark	without BVA	with BVA	speed-up
PH_{10}	7.71	1.25	**6.17**
PH_{11}	84.42	12.34	**6.84**
PH_{12}	494.29	8.45	**58.49**
rbcl_07	52.92	2.88	**18.38**
rbcl_08	1,763.36	10.72	**164.49**
rbcl_09	$> 40,000.00$	129.20	$>$ **309.59**

Outline

All problems solved?

Input: Boolean formula

Black box solver

`unsatisfiable`?

Proof checker

Output: sequence of added and deleted RAT clauses

All problems solved? NO!

Input: Boolean formula

↘

Proof checker

Created problem

The new checker has become complex

Trust checker? NO!

↗

Output: sequence of added and deleted RAT clauses

All problems solved? NO!

Input: Boolean formula

↘

Proof checker

Created problem

The new checker
has become complex
Trust checker? NO!

Our proposed fix

Make compact proof
for a simple checker

↗

Output: sequence of added and deleted RAT clauses

Add a proof filter and verify checker

Input: Boolean formula

Black box solver

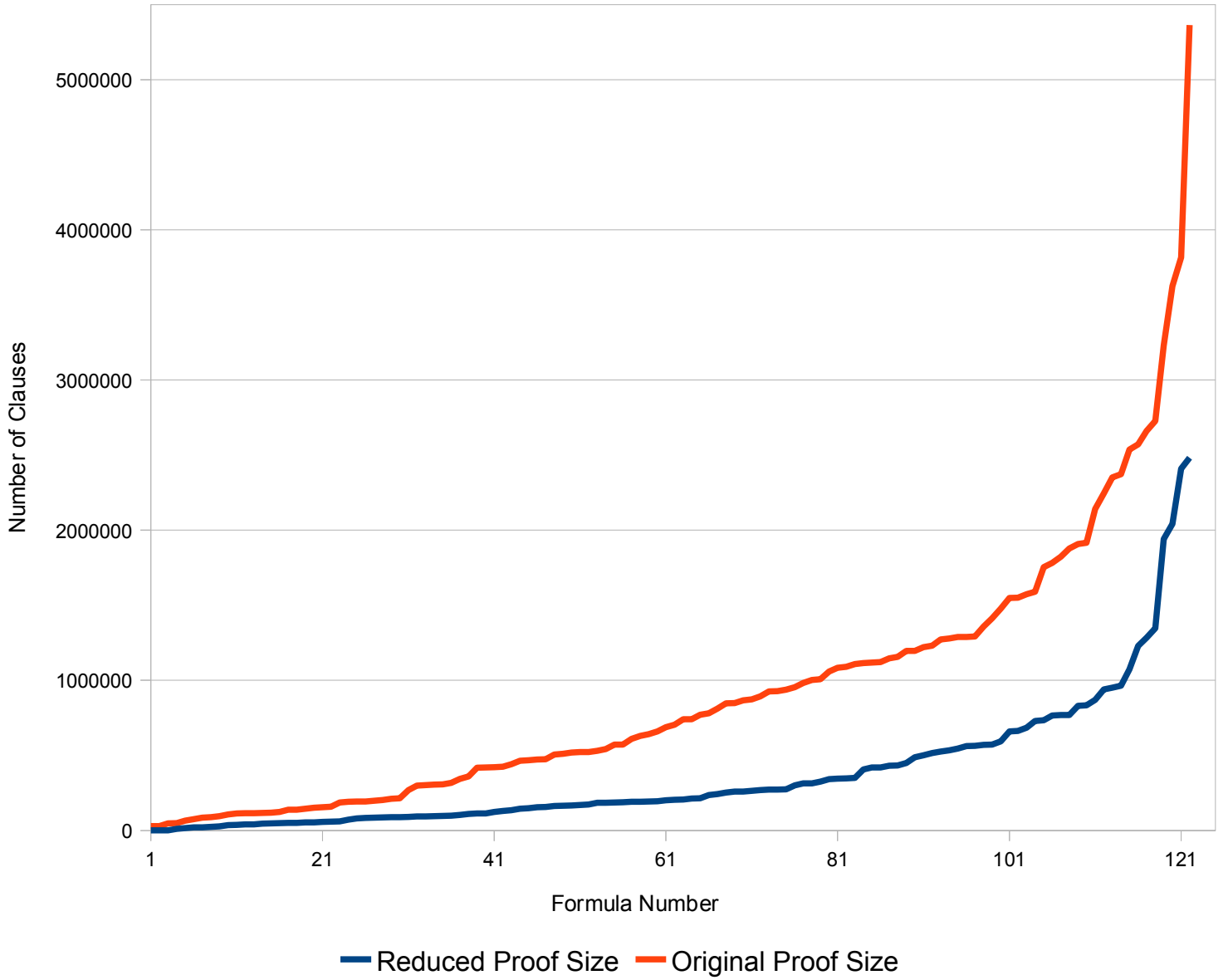
`unsatisfiable`?

Proof filter

Verified checker

Output: sequence of added and deleted RAT clauses

Reduced Size of the Unsatisfiability Proof



Outline

What can go wrong in practice?

From UNSAT to "SAT":

- cause: removal of a non-redundant clause
- rare: removal is generally restricted to added clauses
- cheap to detect: verify the "solution"

What can go wrong in practice?

From UNSAT to "SAT":

- cause: removal of a non-redundant clause
- rare: removal is generally restricted to added clauses
- cheap to detect: verify the "solution"

From SAT to "UNSAT":

- cause: addition of a non-redundant clause
- frequently: "optimization" makes a clause too short
- hard to detect: many possible causes
  - most SAT formulas have millions of solutions
  - most slightly shorter clauses are also redundant
  - expensive to verify all added clauses

Relationship to Our X86 Efforts

- Our symbolic simulation approach, which uses BDDs, does not scale well for larger examples. Centaur has had the same problem.
- Satisfiability (SAT) solvers can scale better than BDDs and can be integrated into symbolic simulation, but we have to *trust* that the solvers are correct.
- We want to develop fast, verified satisfiability proof checkers. This will enable larger x86 code simulation proofs by symbolic simulation.

Recent papers on this subject

- Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler
  Bridging the Gap Between Easy Generation and Efficient Verification of Unsatisfiability Proofs
  Submitted to Software Testing, Verification and Reliability: Special Issue on Tests and Proofs in December 2012.
- Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler
  Verifying Refutations with Extended Resolution
  Accepted for Conference on Automated Deduction 2013.
- Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr.
  Mechanical Verification of SAT Refutations with Extended Resolution
  Accepted for Conference on Interactive Theorem Proving 2013.

Efficient and Verified Checking of Unsatisfiability Proofs

Marijn J.H. Heule
Warren A. Hunt Jr.
Nathan Wetzler

The University of Texas at Austin

May 9, 2013