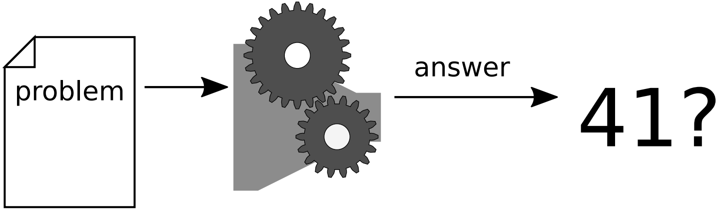
Certifying Parity Reasoning Efficiently Using Pseudo-Boolean Proofs

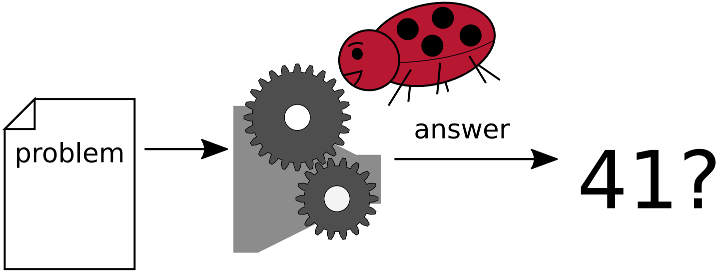
Stephan Gocht, Jakob Nordström

February 2021

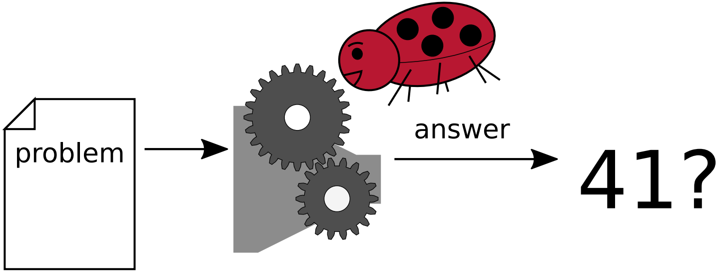
Detecting Bugs with Certifying Algorithms



Detecting Bugs with Certifying Algorithms

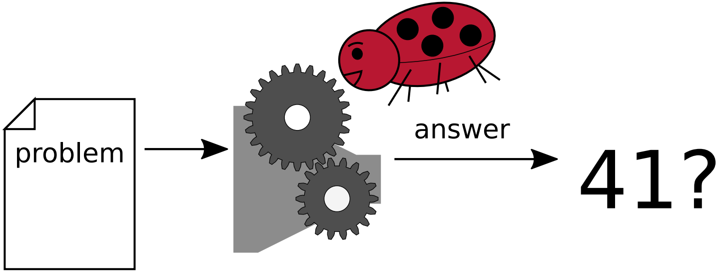


Detecting Bugs with Certifying Algorithms



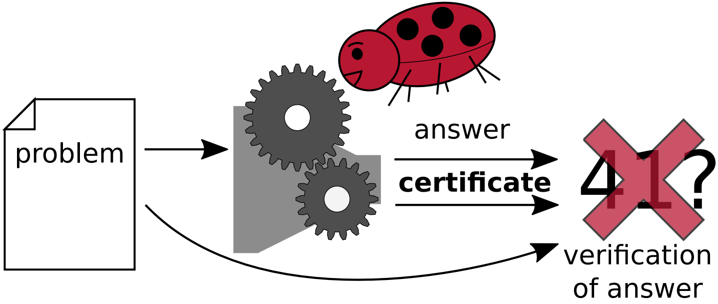
- formally verify solver?
  usually not feasible / too costly

Detecting Bugs with Certifying Algorithms



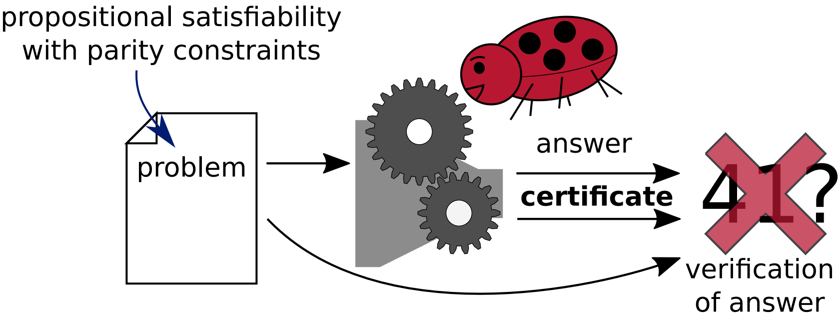
- formally verify solver?
  usually not feasible / too costly
- instead: formally verify answer!

Detecting Bugs with Certifying Algorithms



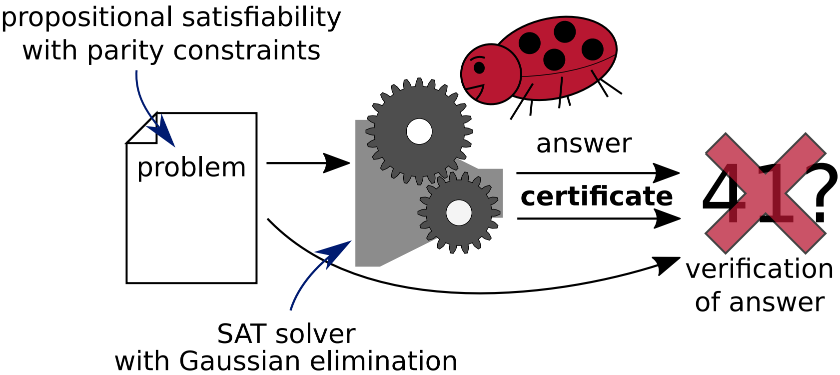
- formally verify solver?
  usually not feasible / too costly
- instead: formally verify answer!

Detecting Bugs with Certifying Algorithms



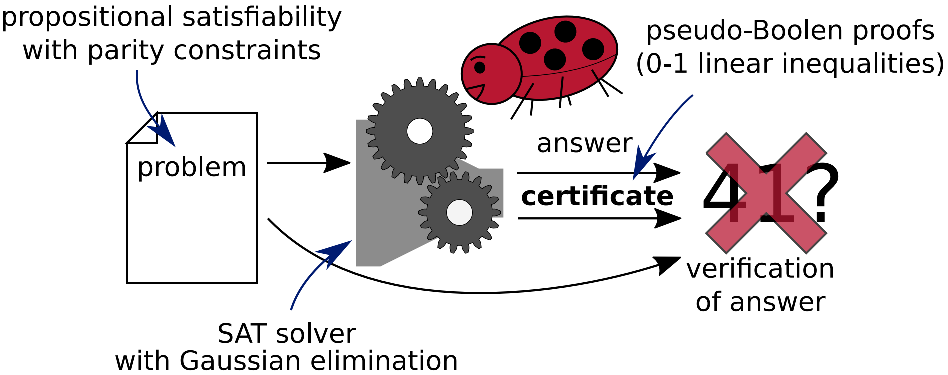
- formally verify solver?
  usually not feasible / too costly
- instead: formally verify answer!

Detecting Bugs with Certifying Algorithms



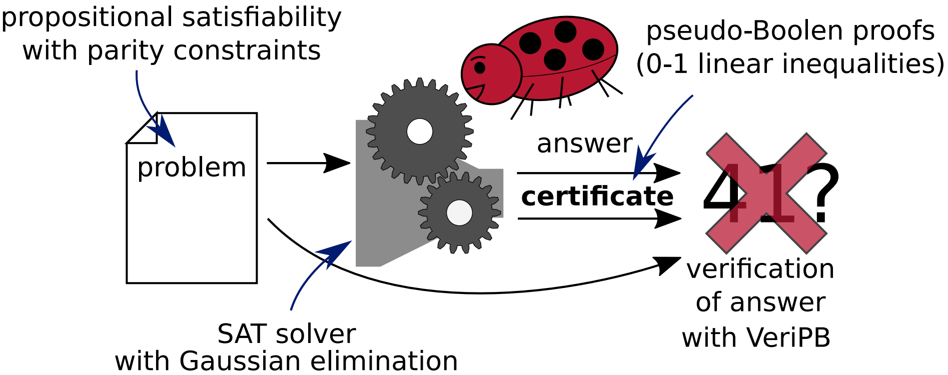
- formally verify solver?
  usually not feasible / too costly
- instead: formally verify answer!

Detecting Bugs with Certifying Algorithms



- formally verify solver?
  usually not feasible / too costly
- instead: formally verify answer!

Detecting Bugs with Certifying Algorithms



- formally verify solver?
  usually not feasible / too costly
- instead: formally verify answer!

SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability testing of propositional formulas

- SAT competition requires solver to produce certificate (aka proof logging)

SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability testing of propositional formulas
- SAT competition requires solver to produce certificate (aka proof logging)
- Proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CFMSSK17], LRAT [CFHH$^+$17]; DRAT [WHH14] has become standard.

SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability testing of propositional formulas
- SAT competition requires solver to produce certificate (aka proof logging)
- Proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CFMSSK17], LRAT [CFHH+17]; DRAT [WHH14] has become standard.
- certificates can help to
  - prove correctness of answer
  - detect and fix bugs, even when solver produced correct answer
  - audit answer later on
  - explain what solver is doing

...Except for SAT Solving Techniques That Can't Be Certified

- too much overhead / too complicated proof logging for
  - **Parity reasoning** (as in CryptoMiniSat [Cry] and Lingeling [Lin])
  - Counting arguments (as in Lingeling)
  - Symmetry breaking (as in BreakID [Bre])

  $\Rightarrow$ no available implementations for proof logging

- Not using these techniques $\Rightarrow$ exponential loss in reasoning power / performance

…Except for SAT Solving Techniques That Can't Be Certified

- too much overhead / too complicated proof logging for
  - **Parity reasoning** (as in CryptoMiniSat [Cry] and Lingeling [Lin])
  - Counting arguments (as in Lingeling)
  - Symmetry breaking (as in BreakID [Bre])

  $\Rightarrow$ no available implementations for proof logging

- Not using these techniques $\Rightarrow$ exponential loss in reasoning power / performance

- How about practical proof logging for stronger solving paradigms?
  - MaxSAT solving
  - constraint programming (CP)
  - mixed integer programming (MIP)
  - algebraic reasoning / Gröbner basis computations
  - pseudo-Boolean satisfiablity and optimization

New Proof Systems on the Rise

many new proof systems with implemented proof checkers:

- propagation redundancy (PR) [HKB17a]
- practical polynomial calculus (PAC) [RBK18, KFB20]
- propagation redundancy for BDDs [BB21]
- Max-SAT resolution [PCH21]
- **pseudo-Boolean proofs** [EGMN20, GN21]

SAT + Parity Reasoning

basic algorithm:

- search + smart look ahead + learning from failure (CDCL)

SAT + Parity Reasoning

basic algorithm:

- search + smart look ahead + learning from failure (CDCL)
- Gaussian elimination on XORs [SNC09, HJ12] to detect
  - propagation (forced values)
  - contradiction (no solution)

SAT + Parity Reasoning

basic algorithm:

- search + smart look ahead + learning from failure (CDCL)
- Gaussian elimination on XORs [SNC09, HJ12] to detect
  - propagation (forced values)
  - contradiction (no solution)

applications:

- solving cryptographic problems
- approximate counting
- circuit verification

$$x_1 + x_2 + x_3 \geq 1$$
$$x_1 + \bar{x}_2 + \bar{x}_3 \geq 1$$
$$\bar{x}_1 + x_2 + \bar{x}_3 \geq 1$$
$$\bar{x}_1 + \bar{x}_2 + x_3 \geq 1$$
$$\bar{x}_2 + x_3 \geq 1$$
$$x_2 + \bar{x}_3 \geq 1$$

- Boolean variable x with domain 0 (false) or 1 (true)
- Literal: x or its negation $\bar{x} = 1 - x$
- Pseudo-Boolean constraint:
  linear (in-)equality over literals
- Clause: at-least-one constraint
- Parity / XOR: equality modulo 2
  notation: $x_1 \oplus x_2 \oplus x_3 = 1$
- Assignment: function mapping variables to $\{0, 1\}$
- VeriPB Proof Format (PBP):
  - based on pseudo-Boolean constraints
  - has operations to reason with PB constraints

Goal: find assignment satisfying all constraints

$$x_1 + x_2 + x_3 \geq 1$$
$$x_1 + \bar{x}_2 + \bar{x}_3 \geq 1$$
$$\bar{x}_1 + x_2 + \bar{x}_3 \geq 1$$
$$\bar{x}_1 + \bar{x}_2 + x_3 \geq 1$$

$$\bar{x}_2 + x_3 \geq 1$$
$$x_2 + \bar{x}_3 \geq 1$$

Claim:

- only satisfied if $x_1 = 1$

- Boolean variable x with domain 0 (false) or 1 (true)
- Literal: x or its negation $\bar{x} = 1 - x$
- Pseudo-Boolean constraint: linear (in-)equality over literals
- Clause: at-least-one constraint
- Parity / XOR: equality modulo 2
  notation: $x_1 \oplus x_2 \oplus x_3 = 1$
- Assignment: function mapping variables to $\{0, 1\}$
- VeriPB Proof Format (PBP):
  - based on pseudo-Boolean constraints
  - has operations to reason with PB constraints

Goal: find assignment satisfying all constraints

$$\left.\begin{aligned} x_1 + x_2 + x_3 &\geq 1 \\ x_1 + \bar{x}_2 + \bar{x}_3 &\geq 1 \\ \bar{x}_1 + x_2 + \bar{x}_3 &\geq 1 \\ \bar{x}_1 + \bar{x}_2 + x_3 &\geq 1 \end{aligned}\right\} \quad \text{clausal encoding of } x_1 \oplus x_2 \oplus x_3 = 1$$

$$\left.\begin{aligned} \bar{x}_2 + x_3 &\geq 1 \\ x_2 + \bar{x}_3 &\geq 1 \end{aligned}\right\} \quad x_2 \oplus x_3 = 0$$

Claim:

- only satisfied if $x_1 = 1$

$$\left.\begin{aligned} x_1 + x_2 + x_3 &\geq 1 \\ x_1 + \bar{x}_2 + \bar{x}_3 &\geq 1 \\ \bar{x}_1 + x_2 + \bar{x}_3 &\geq 1 \\ \bar{x}_1 + \bar{x}_2 + x_3 &\geq 1 \end{aligned}\right\} \quad \text{clausal encoding of } x_1 \oplus x_2 \oplus x_3 = 1$$

$$\left.\begin{aligned} \bar{x}_2 + x_3 &\geq 1 \\ x_2 + \bar{x}_3 &\geq 1 \end{aligned}\right\} \quad x_2 \oplus x_3 = 0$$

Claim:

- only satisfied if $x_1 = 1$

How can we formalize this?

Example

Step 1: Translate XORs

$$\left.\begin{aligned} x_1 + x_2 + x_3 &\geq 1 \\ x_1 + \bar{x}_2 + \bar{x}_3 &\geq 1 \\ \bar{x}_1 + x_2 + \bar{x}_3 &\geq 1 \\ \bar{x}_1 + \bar{x}_2 + x_3 &\geq 1 \end{aligned}\right\} \quad \text{clausal encoding of } x_1 \oplus x_2 \oplus x_3 = 1$$

$$\left.\begin{aligned} \bar{x}_2 + x_3 &\geq 1 \\ x_2 + \bar{x}_3 &\geq 1 \end{aligned}\right\} \quad x_2 \oplus x_3 = 0$$

Example

Step 1: Translate XORs

$$\left.\begin{array}{r} x_1 + x_2 + x_3 \geq 1 \\ x_1 + \bar{x}_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + x_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + \bar{x}_2 + x_3 \geq 1 \end{array}\right\} \quad \text{clausal encoding of } x_1 \oplus x_2 \oplus x_3 = 1$$

$$\left.\begin{array}{r} \bar{x}_2 + x_3 \geq 1 \\ x_2 + \bar{x}_3 \geq 1 \end{array}\right\} \quad x_2 \oplus x_3 = 0$$

Step 2: XOR reasoning (via Gaussian elimination)

add both XORs

$$x_1 = 1$$

Example

Step 1: Translate XORs

$$\left.\begin{array}{r} x_1 + x_2 + x_3 \geq 1 \\ x_1 + \bar{x}_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + x_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + \bar{x}_2 + x_3 \geq 1 \end{array}\right\} \quad \text{clausal encoding of} \quad x_1 \oplus x_2 \oplus x_3 = 1$$

$$\left.\begin{array}{r} \bar{x}_2 + x_3 \geq 1 \\ x_2 + \bar{x}_3 \geq 1 \end{array}\right\} \quad x_2 \oplus x_3 = 0$$

Step 2: XOR reasoning (via Gaussian elimination)

add both XORs

$$x_1 = 1$$

Step 3: Reason clause generation

$$x_1 \geq 1$$

Example

Step 1: Translate XORs

$$\left.\begin{array}{r} x_1 + x_2 + x_3 \geq 1 \\ x_1 + \bar{x}_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + x_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + \bar{x}_2 + x_3 \geq 1 \end{array}\right\} \text{ clausal encoding of } x_1 \oplus x_2 \oplus x_3 = 1 \Rightarrow \text{ convert to pseudo-Boolean constraint } x_1 + x_2 + x_3 = 1 + 2y_1$$

$$\left.\begin{array}{r} \bar{x}_2 + x_3 \geq 1 \\ x_2 + \bar{x}_3 \geq 1 \end{array}\right\} \quad x_2 \oplus x_3 = 0 \Rightarrow x_2 + x_3 = 0 + 2y_2$$

Step 2: XOR reasoning (via Gaussian elimination)

add both XORs

$$x_1 = 1$$

Step 3: Reason clause generation

$$x_1 \geq 1$$

Example

Step 1: Translate XORs

$$\left.\begin{array}{r} x_1 + x_2 + x_3 \geq 1 \\ x_1 + \bar{x}_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + x_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + \bar{x}_2 + x_3 \geq 1 \end{array}\right\} \quad \begin{array}{r} \text{clausal encoding of} \\ x_1 \oplus x_2 \oplus x_3 = 1 \end{array} \quad \Rightarrow \quad \begin{array}{r} \text{convert to pseudo-Boolean constraint} \\ x_1 + x_2 + x_3 = 1 + 2y_1 \end{array}$$

$$\left.\begin{array}{r} \bar{x}_2 + x_3 \geq 1 \\ x_2 + \bar{x}_3 \geq 1 \end{array}\right\} \quad x_2 \oplus x_3 = 0 \quad \Rightarrow \quad x_2 + x_3 = 0 + 2y_2$$

Step 2: XOR reasoning (via Gaussian elimination)

$$\begin{array}{r} \text{add both XORs} \\ x_1 = 1 \end{array} \qquad \begin{array}{r} \text{add both pseudo-Boolean constraints} \\ x_1 + 2x_2 + 2x_3 = 1 + 2y_1 + 2y_2 \end{array}$$

Step 3: Reason clause generation

$$x_1 \geq 1$$

Example

Step 1: Translate XORs

$$\left.\begin{array}{l} x_1 + x_2 + x_3 \geq 1 \\ x_1 + \bar{x}_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + x_2 + \bar{x}_3 \geq 1 \\ \bar{x}_1 + \bar{x}_2 + x_3 \geq 1 \end{array}\right\} \quad \begin{array}{r} \text{clausal encoding of} \\ x_1 \oplus x_2 \oplus x_3 = 1 \end{array} \quad \Rightarrow \quad \begin{array}{r} \text{convert to pseudo-Boolean constraint} \\ x_1 + x_2 + x_3 = 1 + 2y_1 \end{array}$$

$$\left.\begin{array}{l} \bar{x}_2 + x_3 \geq 1 \\ x_2 + \bar{x}_3 \geq 1 \end{array}\right\} \quad x_2 \oplus x_3 = 0 \quad \Rightarrow \quad x_2 + x_3 = 0 + 2y_2$$

Step 2: XOR reasoning (via Gaussian elimination)

add both XORs	add both pseudo-Boolean constraints
$x_1 = 1$	$x_1 + 2x_2 + 2x_3 = 1 + 2y_1 + 2y_2$

Step 3: Reason clause generation

$$x_1 \geq 1 \qquad\qquad x_1 \geq 1$$

Example

Step 1: Translate XORs

$$\left.\begin{aligned} x_1 + x_2 + x_3 &\geq 1 \\ x_1 + \bar{x}_2 + \bar{x}_3 &\geq 1 \\ \bar{x}_1 + x_2 + \bar{x}_3 &\geq 1 \\ \bar{x}_1 + \bar{x}_2 + x_3 &\geq 1 \end{aligned}\right\}$$

clausal encoding of $x_1 \oplus x_2 \oplus x_3 = 1$ $\Rightarrow$ convert to pseudo-Boolean constraint $x_1 + x_2 + x_3 = 1 + 2y_1$

$$\left.\begin{aligned} \bar{x}_2 + x_3 &\geq 1 \\ x_2 + \bar{x}_3 &\geq 1 \end{aligned}\right\}$$

$x_2 \oplus x_3 = 0$ $\Rightarrow$ $x_2 + x_3 = 0 + 2y_2$

Step 2: XOR reasoning (via Gaussian elimination)

add both XORs	add both pseudo-Boolean constraints
$x_1 = 1$	$x_1 + 2x_2 + 2x_3 = 1 + 2y_1 + 2y_2$

Step 3: Reason clause generation

$x_1 \geq 1$ $\qquad$ $x_1 \geq 1$

All steps easily expressible in VeriPB!

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∼ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∼ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure
- ✓ use gaussian elimination to detect propagation

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∽ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure
- ✓ use gaussian elimination to detect propagation
- ✓ generate reason clause

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∼ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure
- ✓ use gaussian elimination to detect propagation
- ✓ generate reason clause
- ∼ detect CNF encoding of XORs with bloom filters [SM19]

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∽ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure
- ✓ use gaussian elimination to detect propagation
- ✓ generate reason clause
- ∽ detect CNF encoding of XORs with bloom filters [SM19]
- ✓ blast and recover XORs for inprocessing [SM19]

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∼ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure
- ✓ use gaussian elimination to detect propagation
- ✓ generate reason clause
- ∼ detect CNF encoding of XORs with bloom filters [SM19]
- ✓ blast and recover XORs for inprocessing [SM19]
- ∼ use watched literals structure [HJ12]

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∽ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure
- ✓ use gaussian elimination to detect propagation
- ✓ generate reason clause
- ∽ detect CNF encoding of XORs with bloom filters [SM19]
- ✓ blast and recover XORs for inprocessing [SM19]
- ∽ use watched literals structure [HJ12]
- ∽ use sophisticated bit parallelism [HJ12, SGM20]

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∼ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure
- ✓ use gaussian elimination to detect propagation
- ✓ generate reason clause
- ∼ detect CNF encoding of XORs with bloom filters [SM19]
- ✓ blast and recover XORs for inprocessing [SM19]
- ∼ use watched literals structure [HJ12]
- ∼ use sophisticated bit parallelism [HJ12, SGM20]
- ∼ generate reason clause only lazyly [SGM20]

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∼ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure
- ✓ use gaussian elimination to detect propagation
- ✓ generate reason clause
- ∼ detect CNF encoding of XORs with bloom filters [SM19]
- ✓ blast and recover XORs for inprocessing [SM19]
- ∼ use watched literals structure [HJ12]
- ∼ use sophisticated bit parallelism [HJ12, SGM20]
- ∼ generate reason clause only lazyly [SGM20]

Note:

- proof logging verifies that propagations are correct
- no guarantee that all propagations detected

Techniques for Efficient Parity Propagation

legend: ✓ can be verified, ∼ does not need to be verified, ✗ can not be verified

- ✓ replace CNF encoding with internal XOR datastructure
- ✓ use gaussian elimination to detect propagation
- ✓ generate reason clause
- ∼ detect CNF encoding of XORs with bloom filters [SM19]
- ✓ blast and recover XORs for inprocessing? [SM19]
- ∼ use watched literals structure [HJ12]
- ∼ use sophisticated bit parallelism [HJ12, SGM20]
- ∼ generate reason clause only lazyly [SGM20]

Note:

- proof logging verifies that propagations are correct
- no guarantee that all propagations detected
- SAT inprocessing requires to generalize DRAT (next slides)

More Notation

- (partial) substitution $\omega = \{ y_1 \mapsto 0 \}$
  function that maps variables to literals or $\{ 0, 1 \}$
- variable substitution

$$(x_1 + x_2 + x_3 \geq 2y_1)_{\restriction\omega} = x_1 + x_2 + x_3 \geq 0$$

- $F \models F'$: satisfying assignment to F is also satisfying assignment to F'

Substitution Redundancy Rule (Generalizing DRAT)

Substitution Redundancy (generalizing [HKB17b, BT19] to pseudo-Boolean)

Can add constraint C to formula F if and only if there is a *witnessing* partial substitution ω such that

$$F \wedge \neg C \models (F \wedge C)_{\restriction\omega}$$

- C need not be implied, but F satisfiable if and only if $F \wedge C$ satisfiable

Substitution Redundancy Rule (Generalizing DRAT)

Substitution Redundancy (generalizing [HKB17b, BT19] to pseudo-Boolean)

Can add constraint C to formula F if and only if there is a *witnessing* partial substitution ω such that

$$F \land \neg C \models (F \land C)\restriction_{\omega}$$

- C need not be implied, but F satisfiable if and only if $F \land C$ satisfiable
- as stated, to good to be true: can derive contradiction in one step

Substitution Redundancy Rule (Generalizing DRAT)

Substitution Redundancy (generalizing [HKB17b, BT19] to pseudo-Boolean)

Can add constraint C to formula F if and only if there is a *witnessing* partial substitution ω such that

$$F \wedge \neg C \models (F \wedge C)\restriction_\omega$$

- C need not be implied, but F satisfiable if and only if $F \wedge C$ satisfiable
- as stated, to good to be true: can derive contradiction in one step
- make efficiently verifiable by insisting implication easy to check

Substitution Redundancy Rule (Generalizing DRAT)

Substitution Redundancy (generalizing [HKB17b, BT19] to pseudo-Boolean)

Can add constraint C to formula F if and only if there is a *witnessing* partial substitution ω such that

$$F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$$

- C need not be implied, but F satisfiable if and only if $F \wedge C$ satisfiable
- as stated, to good to be true: can derive contradiction in one step
- make efficiently verifiable by insisting implication easy to check
- generalizes DRAT [HKB17b]
- $\Rightarrow$ all SAT pre- and inprocessing techniques covered

Example Substitution Redundancy

For fresh variable y_1 (not appearing in F), want to add...

$$C : \quad x_1 + x_2 + x_3 \geq 2y_1$$

Example Substitution Redundancy

For fresh variable y_1 (not appearing in F), want to add...

$$C: \quad x_1 + x_2 + x_3 \geq 2y_1$$

Choose witness $\omega = \{ y_1 \mapsto 0 \}$
Check condition $F \land \neg C \models (F \land C)_{\restriction\omega}$, i.e.,

$$F \land (x_1 + x_2 + x_3 < 2y_1) \models F \land (x_1 + x_2 + x_3 \geq 2y_1)_{\restriction\omega}$$

Example Substitution Redundancy

For fresh variable y_1 (not appearing in F), want to add...

$$C: \quad x_1 + x_2 + x_3 \geq 2y_1$$

Choose witness $\omega = \{ y_1 \mapsto 0 \}$
Check condition $F \wedge \neg C \models (F \wedge C)_{\restriction\omega}$, i.e.,

$$F \wedge (x_1 + x_2 + x_3 < 2y_1) \models F \wedge (x_1 + x_2 + x_3 \geq 0)$$

Example Substitution Redundancy

For fresh variable y_1 (not appearing in F), want to add...

$$C: \quad x_1 + x_2 + x_3 \geq 2y_1$$

Choose witness $\omega = \{ y_1 \mapsto 0 \}$
Check condition $F \wedge \neg C \models (F \wedge C)_{\restriction\omega}$, i.e.,

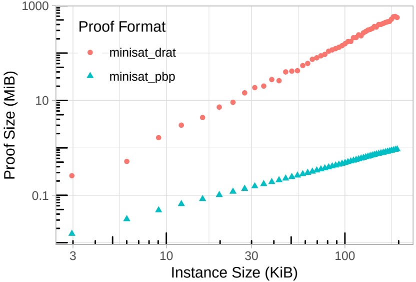
$$F \wedge (x_1 + x_2 + x_3 < 2y_1) \models F \wedge (x_1 + x_2 + x_3 \geq 0)$$

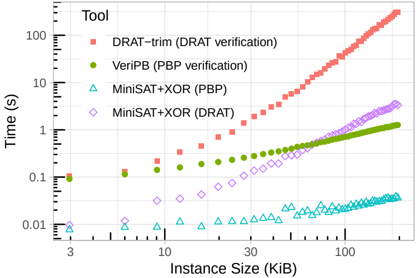
concrete proof format:

```
red  1 x1 +1 x2 +1 x3 -2 y1 >=  0 ; y1 -> 0
```

Experiments

- Implemented "plug and play" XorEngine with proof logging[1] in MiniSAT[2]
- Evaluated on crafted benchmarks (Tseitin-Formulas)
  represent worst case with single large XOR matrix
- DRAT proof for comparison [PR16]





[1] https://gitlab.com/MIAOresearch/xorengine
[2] https://gitlab.com/MIAOresearch/minisat_with_xorengine

Conclusion

- proof logging is well-established standard for SAT solving
- so far, prohibitively expensive for some techniques (XOR reasoning, counting arguments, symmetry breaking)

3

Conclusion

- proof logging is well-established standard for SAT solving
- so far, prohibitively expensive for some techniques
  (XOR reasoning, counting arguments, symmetry breaking)

Our work: Proof logging for SAT solving and XOR reasoning with VeriPB[3]

- simple to implement + efficient proof checking

[3] `https://gitlab.com/MIAOresearch/VeriPB`

Conclusion

- proof logging is well-established standard for SAT solving
- so far, prohibitively expensive for some techniques (XOR reasoning, counting arguments, symmetry breaking)

Our work: Proof logging for SAT solving and XOR reasoning with VeriPB[3]

- simple to implement + efficient proof checking

Future work:

- capture more types of reasoning within SAT solvers
  - counting arguments (should be straightforward)
  - symmetry breaking
- provide efficient proof logging also for other paradigms (MaxSAT, pseudo-Boolean optimization, MIP)
- new expressive proof formats and verifiers for competitions (why not with VeriPB ;-))

[3] `https://gitlab.com/MIAOresearch/VeriPB`

References I

[BB21] Lee A. Barnett and Armin Biere.
Non-clausal redundancy properties.
In *Proceedings of the 28th International Conference on Automated Deduction (CADE-28)*, page to appear, 2021.

[Bie06] Armin Biere.
Tracecheck.
`http://fmv.jku.at/tracecheck/`, 2006.

[Bre] BreakID.
`https://bitbucket.org/krr/breakid/src/master/`.

[BT19] Sam Buss and Neil Thapen.
DRAT proofs, propagation redundancy, and extended resolution.
In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2019.

[CFHH+17] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp.
Efficient certified rat verification.
In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 220–236, Cham, 2017. Springer International Publishing.

References II

[CFMSSK17] Luís Cruz-Filipe, Joao Marques-Silva, and Peter Schneider-Kamp.
Efficient certified resolution proof checking.
In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 118–135. Springer Berlin Heidelberg, 2017.

[Cry] CryptoMiniSat.
`https://github.com/msoos/cryptominisat/`.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström.
Justifying all differences using pseudo-boolean reasoning.
In *Proceedings of the 34th AAAI Conference on Artificial Intelligence. To appear*, 2020.

[GN03] Evguenii I. Goldberg and Yakov Novikov.
Verification of proofs of unsatisfiability for CNF formulas.
In *Design, Automation and Test in Europe Conference (DATE)*, pages 10886–10891. IEEE Computer Society, 2003.

[GN21] Stephan Gocht and Jakob Nordström.
Certifying parity reasoning efficiently using pseudo-Boolean proofs.
In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 2021.
To appear.

References III

[HJ12] Cheng-Shen Han and Jie-Hong Roland Jiang.
When boolean satisfiability meets Gaussian elimination in a Simplex way.
In *Computer Aided Verification, CAV*, pages 410–426, 2012.

[HKB17a] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere.
Short proofs without new variables.
In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2017.

[HKB17b] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere.
Short proofs without new variables.
In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 130–147. Springer, August 2017.

[KFB20] Daniela Kaufmann, Mathias Fleury, and Armin Biere.
The proof checkers pacheck and pastÃ¨que for the practical algebraic calculus.
In Ofer Strichman and Alexander Ivrii, editors, *Formal Methods in Computer-Aided Design, FMCAD 2020.*, volume 1, pages 264–269. TU Vienna Academic Press, 2020.

[Lin] Lingeling, Plingeling and Treengeling.
`http://fmv.jku.at/lingeling/`.

References IV

[PCH21] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet.
A proof builder for max-sat.
In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 488–498, Cham, 2021. Springer International Publishing.

[PR16] Tobias Philipp and Adrian Rebola-Pardo.
DRAT proofs for XOR reasoning.
In *Logics in Artificial Intelligence (JELIA 2016).*, volume 10021 of *Lecture Notes in Computer Science.*, pages 415–429, 2016.

[RBK18] Daniela Ritirc, Armin Biere, and Manuel Kauers.
A practical polynomial calculus for arithmetic circuit verification.
In Anna M. Bigatti and Martin Brain, editors, *3rd International Workshop on Satisfiability Checking and Symbolic Computation (SC2'18)*, pages 61–76. CEUR-WS, 2018.

[SGM20] Mate Soos, Stephan Gocht, and Kuldeep S. Meel.
Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling.
In *Computer Aided Verification, CAV 2020*, volume 12224 of *Lecture Notes in Computer Science*, pages 463–484. Springer, 2020.

References V

[SM19] Mate Soos and Kuldeep S. Meel.
BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting.
In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1592–1599, 2019.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia.
Extending SAT Solvers to Cryptographic Problems.
In *Proc. of SAT*, 2009.

[WHH14] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr.
DRAT-trim: Efficient checking and trimming using expressive clausal proofs.
In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, July 2014.