A Unified Proof System for Discrete Combinatorial Problems

Jakob Nordström

University of Copenhagen and Lund University

Dagstuhl Seminar 23471
“The Next Generation of Deduction Systems:
From Composition to Compositionality”
November 24, 2023

Based on joint work with Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Andy Oertel, and Yong Kiam Tan

A Unified Proof System for Discrete Combinatorial Problems?

Jakob Nordström

University of Copenhagen and Lund University

Dagstuhl Seminar 23471
"The Next Generation of Deduction Systems: From Composition to Compositionality"
November 24, 2023

Based on joint work with Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Andy Oertel, and Yong Kiam Tan

The Success of Combinatorial Solving (and the Dirty Little Secret)

- Astounding progress last couple of decades on combinatorial solvers for, e.g.:
  - Boolean satisfiability (SAT) solving and optimization [BHvMW21]
  - Constraint programming [RvBW06]
  - Mixed integer linear programming [AW13, BR07]
  - Satisfiability modulo theories (SMT) solving [BHvMW21]
- Solvers very fast, but sometimes wrong (even best commercial ones) [BLB10, CKSW13, AGJ$^+$18, GSD19, GS19, BMN22, BBN$^+$23]
- Even get feasibility of solutions wrong (though this should be straightforward!)
- And how to check the absence of solutions?
- Or that a solution is optimal? (Even off-by-one mistakes can snowball into large errors if solver used as subroutine)

What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But inherently can only detect presence of bugs, not absence

What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But inherently can only detect presence of bugs, not absence

- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to this level of complexity

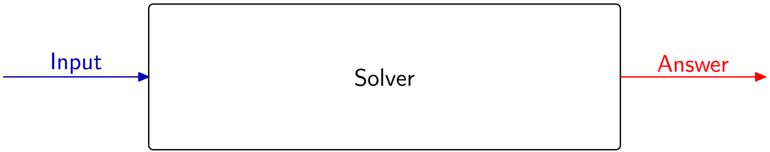
What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But inherently can only detect presence of bugs, not absence

- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to this level of complexity

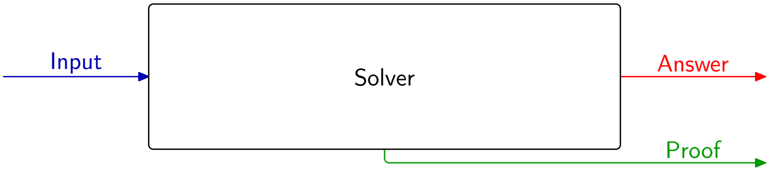
- **Proof logging**
  Make solver certifying [ABM+11, MMNS11] by adding code so that it outputs
  1. not only answer but also
  2. simple, machine-verifiable proof that answer is correct

Proof Logging with Certifying Solvers: Workflow



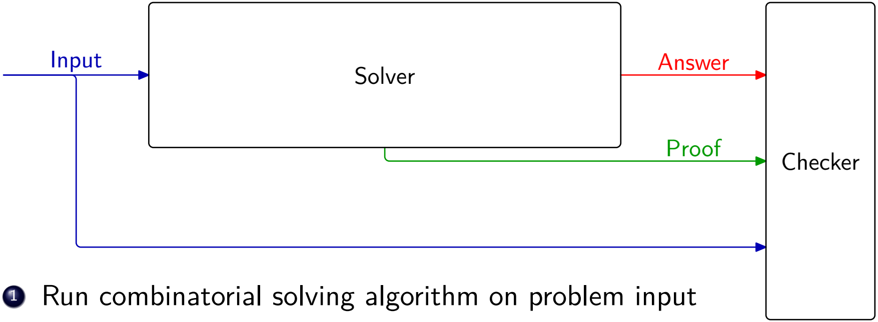
1. Run combinatorial solving algorithm on problem input

Proof Logging with Certifying Solvers: Workflow



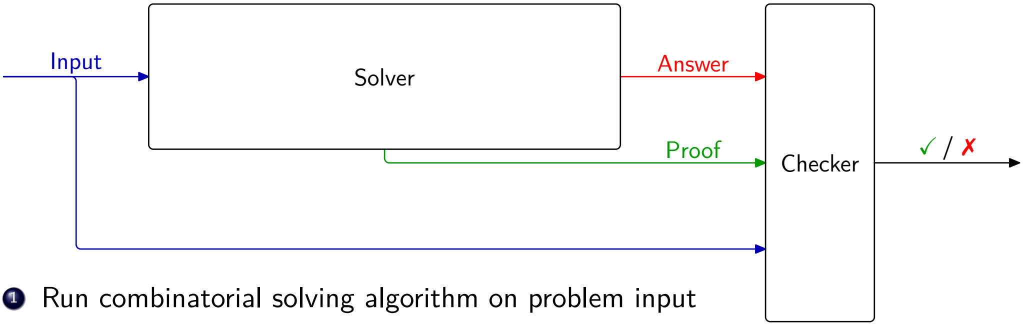
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

Proof Logging with Certifying Solvers: Workflow



1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker

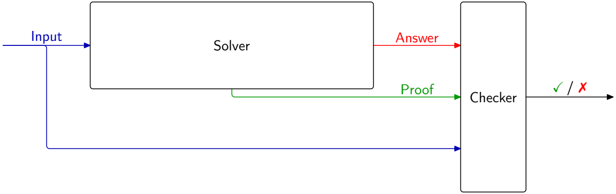
Proof Logging with Certifying Solvers: Workflow



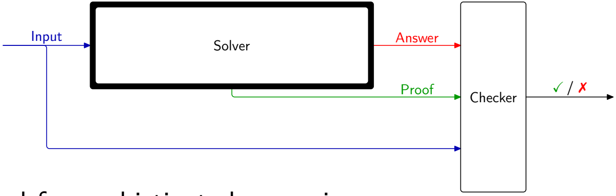
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

Proof Logging Desiderata

Proof format for certifying solver should be



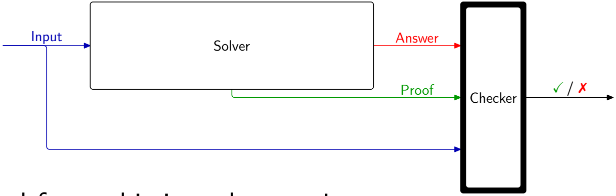
Proof Logging Desiderata



Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning

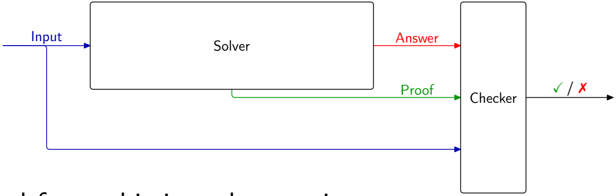
Proof Logging Desiderata



Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning
- **dead simple:** checking correctness of proofs should be (almost) trivial

Proof Logging Desiderata

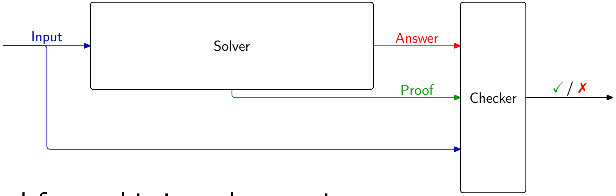


Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning
- **dead simple:** checking correctness of proofs should be (almost) trivial

Clear conflict expressivity vs. simplicity!

Proof Logging Desiderata



Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning
- **dead simple:** checking correctness of proofs should be (almost) trivial

Clear conflict expressivity vs. simplicity!

Asking for both perhaps a little bit too good to be true?

This Talk

Proof logging for combinatorial optimization is possible with single, unified method!

This Talk

Proof logging for combinatorial optimization is possible with single, unified method!

- Build on successes in SAT solving with proof formats such as DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], ...
- But represent constraints as 0–1 integer linear inequalities
- Formalize reasoning using cutting planes [CCT87] proof system
- Add well-chosen strengthening rules [Goc22, GN21, BGMN23]
- Implemented in VeriPB (`https://gitlab.com/MIAOresearch/software/VeriPB`)

This Talk

Proof logging for combinatorial optimization is possible with single, unified method!

- Build on successes in SAT solving with proof formats such as DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], ...
- But represent constraints as 0–1 integer linear inequalities
- Formalize reasoning using cutting planes [CCT87] proof system
- Add well-chosen strengthening rules [Goc22, GN21, BGMN23]
- Implemented in VeriPB (https://gitlab.com/MIAOresearch/software/VeriPB)

Purpose of this talk:

1. Marketing pitch ☺

This Talk

Proof logging for combinatorial optimization is possible with single, unified method!

- Build on successes in SAT solving with proof formats such as DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], ...
- But represent constraints as 0–1 integer linear inequalities
- Formalize reasoning using cutting planes [CCT87] proof system
- Add well-chosen strengthening rules [Goc22, GN21, BGMN23]
- Implemented in VeriPB (`https://gitlab.com/MIAOresearch/software/VeriPB`)

Purpose of this talk:

1. Marketing pitch ☺
2. Explore potential connections with more challenging settings such as SMT, first-order logic, ...

The Sales Pitch For Proof Logging

1. Certifies correctness of computed results
2. Detects errors even if due to compiler bugs, hardware failures, or cosmic rays
3. Provides debugging support during software development [EG21, GMM+20, KM21, BBN+23]
4. Facilitates performance analysis
5. Helps identify potential for further improvements
6. Enables auditability
7. Serves as stepping stone towards explainability

Design Principles for Proof Logging

Proof logging implementation

- Don't change solver
- Just add proof logging statements (plus some book-keeping) to solver code

Design Principles for Proof Logging

Proof logging implementation

- Don't change solver
- Just add proof logging statements (plus some book-keeping) to solver code

Performance goals

- Proof logging overhead small constant fraction ($\lessapprox 10\%$)
- Proof checking time within constant factor of solving time (current aim $\lessapprox \times 10$)

Design Principles for Proof Logging

Proof logging implementation

- Don't change solver
- Just add proof logging statements (plus some book-keeping) to solver code

Performance goals

- Proof logging overhead small constant fraction ($\lessapprox 10\%$)
- Proof checking time within constant factor of solving time (current aim $\lessapprox \times 10$)

Proof system

- Keep proof language maximally simple
- Reason about XOR constraints, CP propagators, symmetries, etc within language
- Combine proof logging with formally verified proof checker

Pseudo-Boolean Constraints

Proof consists of 0-1 integer linear inequalities or pseudo-Boolean constraints:

$$\sum_i a_i \ell_i \geq A$$

- $a_i, A \in \mathbb{Z}$
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)
- variables x_i take values $0 = \mathit{false}$ or $1 = \mathit{true}$

Pseudo-Boolean Constraints

Proof consists of 0-1 integer linear inequalities or pseudo-Boolean constraints:

$$\sum_i a_i \ell_i \geq A$$

- $a_i, A \in \mathbb{Z}$
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)
- variables x_i take values $0 = \mathit{false}$ or $1 = \mathit{true}$

Sometimes convenient to use normalized form [Bar95] with all a_i, A positive (without loss of generality)

Some Types of Pseudo-Boolean Constraints

1. Clauses

$$x \vee \overline{y} \vee z \quad \Leftrightarrow \quad x + \overline{y} + z \geq 1$$

2. Cardinality constraints

$$x_1 + x_2 + x_3 + x_4 \geq 2$$

3. General pseudo-Boolean constraints

$$x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

Pseudo-Boolean Proof Logging Wishlist

Paradigms

- SAT solving
- (linear) pseudo-Boolean solving
- graph solving
- constraint programming
- automated planning
- mixed integer linear programming
- SMT solving

Problem types

- decision / feasibility
- optimization
- multi-objective optimization
- projected model enumeration
- projected model counting
- preprocessing / problem reformulation

Pseudo-Boolean Proof Logging Wishlist

Paradigms

- SAT solving
- (linear) pseudo-Boolean solving
- graph solving
- constraint programming
- automated planning
- mixed integer linear programming
- SMT solving

Problem types

- decision / feasibility
- optimization
- multi-objective optimization
- projected model enumeration
- projected model counting
- preprocessing / problem reformulation

Supported in VeriPB presently

Pseudo-Boolean Proof Logging Wishlist

Paradigms

- SAT solving
- (linear) pseudo-Boolean solving
- graph solving
- constraint programming
- automated planning
- mixed integer linear programming
- SMT solving

Problem types

- decision / feasibility
- optimization
- multi-objective optimization
- projected model enumeration
- projected model counting
- preprocessing / problem reformulation

Supported in VeriPB presently, Real Soon Now™

Pseudo-Boolean Proof Logging Wishlist

Paradigms

- SAT solving
- (linear) pseudo-Boolean solving
- graph solving
- constraint programming
- automated planning
- mixed integer linear programming
- SMT solving

Problem types

- decision / feasibility
- optimization
- multi-objective optimization
- projected model enumeration
- projected model counting
- preprocessing / problem reformulation

Supported in VeriPB presently, Real Soon Now™, or hopefully in future extensions

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0-1 integer linear program

- just do proof logging [basically: add print statements to solver code]

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0-1 integer linear program

- just do proof logging [basically: add print statements to solver code]

Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0-1 integer linear program

- just do proof logging [basically: add print statements to solver code]

Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

Goldilocks compromise between expressivity and simplicity:

1. 0-1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments
3. Efficient reification of constraints

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0-1 integer linear program

- just do proof logging [basically: add print statements to solver code]

Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

Goldilocks compromise between expressivity and simplicity:

1. 0-1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments
3. Efficient reification of constraints — example:

$$r \Rightarrow x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

$$r \Leftarrow x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0-1 integer linear program

- just do proof logging [basically: add print statements to solver code]

Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

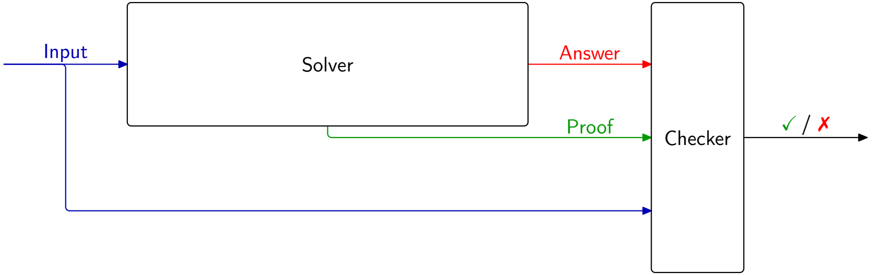
Goldilocks compromise between expressivity and simplicity:

1. 0-1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments
3. Efficient reification of constraints — example:

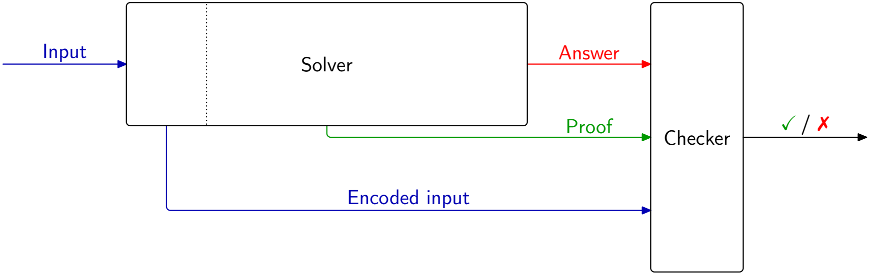
$$r \Rightarrow x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7 \qquad 7\overline{r} + x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

$$r \Leftarrow x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7 \qquad 9r + \overline{x}_1 + 2x_2 + 3\overline{x}_3 + 4x_4 + 5\overline{x}_5 \geq 9$$

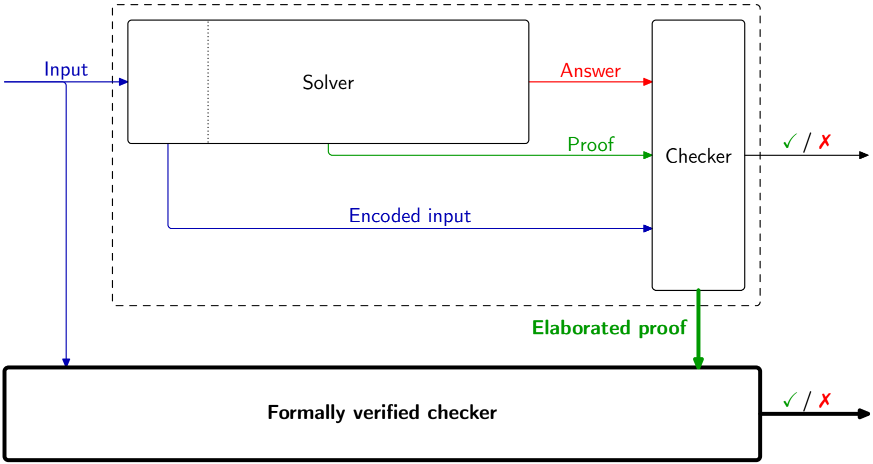
Proof Logging with Formally Verified Checking: Full Workflow



Proof Logging with Formally Verified Checking: Full Workflow



Proof Logging with Formally Verified Checking: Full Workflow



VeriPB Proof Configuration (Slightly Simplified)

Core set $\mathcal{C}$

- Contains input formula at the start
- Maintains "equivalence" with input formula

Objective $f = \sum_i w_i \ell_i + k$

- 0–1 linear function to minimize
- Or $f = 0$ for decision problem
- Keep track of best known bound; initialize to ∞

Derived set $\mathcal{D}$

- All constraints derived during search
- Also intermediate constraints used in proof logging [but not used by solver]

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input axioms From the input

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input axioms From the input

Literal axioms $$\frac{}{\ell_i \geq 0}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input axioms From the input

Literal axioms
$$\frac{}{\ell_i \geq 0}$$

Addition
$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input axioms From the input

Literal axioms

$$\frac{}{\ell_i \geq 0}$$

Addition

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$$

Multiplication for any $c \in \mathbb{N}^+$

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i c a_i \ell_i \geq cA}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input axioms	From the input
Literal axioms	$\dfrac{}{\ell_i \geq 0}$
Addition	$\dfrac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$
Multiplication for any $c \in \mathbb{N}^+$	$\dfrac{\sum_i a_i \ell_i \geq A}{\sum_i c a_i \ell_i \geq cA}$
Division for any $c \in \mathbb{N}^+$ (constraint in normalized form)	$\dfrac{\sum_i a_i \ell_i \geq A}{\sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil}$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input axioms — From the input

Literal axioms

$$\frac{}{\ell_i \geq 0}$$

Addition

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$$

Multiplication for any $c \in \mathbb{N}^+$

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i c a_i \ell_i \geq cA}$$

Division for any $c \in \mathbb{N}^+$
(constraint in normalized form)

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil}$$

Saturation
(constraint in normalized form)

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \min(a_i, A) \cdot \ell_i \geq A}$$

Cutting Planes Toy Example

$$w + 2x + y \geq 2$$

Cutting Planes Toy Example

Multiply by 2 $\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$

Cutting Planes Toy Example

Multiply by 2 $\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$ $\qquad w + 2x + 4y + 2z \geq 5$

Cutting Planes Toy Example

$$\dfrac{\text{Multiply by } 2\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}\ \text{Add}$$

Cutting Planes Toy Example

$$
\dfrac{\text{Multiply by } 2\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5 \qquad \overline{z} \geq 0}{3w + 6x + 6y + 2z \geq 9}\ \text{Add}
$$

Cutting Planes Toy Example

$$\text{Add}\ \dfrac{\text{Multiply by } 2\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Multiply by } 2$$

Cutting Planes Toy Example

$$\dfrac{\dfrac{\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}\ \text{Multiply by 2} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}\ \text{Add} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Multiply by 2}}{3w + 6x + 6y + 2z + 2\overline{z} \geq 9}\ \text{Add}$$

Cutting Planes Toy Example

$$\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Multiply by 2}\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Multiply by 2}}{3w + 6x + 6y + 2 \qquad \geq 9}$$

Cutting Planes Toy Example

$$
\text{Add}\;\dfrac{\text{Multiply by 2}\;\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{\text{Add}\;\dfrac{3w + 6x + 6y + 2z \geq 9 \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\;\text{Multiply by 2}}{3w + 6x + 6y \qquad\qquad \geq 7}}
$$

Cutting Planes Toy Example

$$\text{Divide by } 3\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Multiply by } 2\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Multiply by } 2}{3w + 6x + 6y \geq 7}}{w + 2x + 2y \geq 2\tfrac{1}{3}}$$

Cutting Planes Toy Example

$$
\dfrac{\dfrac{\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}\,\text{Multiply by } 2 \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}\,\text{Add} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\,\text{Multiply by } 2}{\dfrac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 3}\,\text{Divide by } 3}\,\text{Add}
$$

Cutting Planes Toy Example

$$
\dfrac{\dfrac{\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}\ \text{Multiply by } 2 \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}\ \text{Add} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Multiply by } 2}{\dfrac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 3}\ \text{Divide by } 3}\ \text{Add}
$$

By naming constraints by integers and literal axioms by the literal involved as

$$
\begin{aligned}
\text{Constraint 1} &\doteq 2x + y + w \geq 2 \\
\text{Constraint 2} &\doteq 2x + 4y + 2z + w \geq 5 \\
\sim\text{z} &\doteq \overline{z} \geq 0
\end{aligned}
$$

Cutting Planes Toy Example

$$
\dfrac{\dfrac{\dfrac{\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}\text{ Multiply by 2} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}\text{ Add} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\text{ Multiply by 2}}{3w + 6x + 6y \geq 7}\text{ Add}}{w + 2x + 2y \geq 3}\text{ Divide by 3}
$$

By naming constraints by integers and literal axioms by the literal involved as

$$
\begin{aligned}
\text{Constraint 1} &\doteq 2x + y + w \geq 2 \\
\text{Constraint 2} &\doteq 2x + 4y + 2z + w \geq 5 \\
\sim\text{z} &\doteq \overline{z} \geq 0
\end{aligned}
$$

such a calculation is written in the proof log in reverse Polish notation as

```
pol 1 2 * 2 + ~z 2 * + 3 d
```

Redundance-Based Strengthening

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Want to allow adding such “redundant” constraints

Redundance-Based Strengthening

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Want to allow adding such “redundant” constraints

Redundance-based strengthening ([BT19, GN21], inspired by [JHB12])

C is redundant with respect to F if and only if there is a substitution ω (mapping variables to truth values or literals), called a witness, for which

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

Redundance-Based Strengthening

C is redundant with respect to F if F and $F \land C$ are equisatisfiable
Want to allow adding such "redundant" constraints

Redundance-based strengthening ([BT19, GN21], inspired by [JHB12])

C is redundant with respect to F if and only if there is a substitution ω (mapping variables to truth values or literals), called a witness, for which

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

- Proof sketch for interesting direction: If α satisfies F but falsifies C, then $\alpha \circ \omega$ satisfies $F \land C$

Redundance-Based Strengthening

C is redundant with respect to F if F and $F \land C$ are equisatisfiable

Want to allow adding such "redundant" constraints

Redundance-based strengthening ([BT19, GN21], inspired by [JHB12])

C is redundant with respect to F if and only if there is a substitution ω (mapping variables to truth values or literals), called a witness, for which

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

- Proof sketch for interesting direction: If α satisfies F but falsifies C, then $\alpha \circ \omega$ satisfies $F \land C$
- In a proof, the implication needs to be efficiently verifiable — every $D \in (F \land C)\restriction_{\omega}$ should follow from $F \land \neg C$ either
  1. "obviously" or
  2. by explicitly presented derivation

Redundance and Dominance Rules in VeriPB (Slightly Simplified)

Redundance-based strengthening, optimization version [BGMN23]

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models (\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{f\restriction_\omega \leq f\}$$

Redundance and Dominance Rules in VeriPB (Slightly Simplified)

Redundance-based strengthening, optimization version [BGMN23]

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models (\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{f\restriction_\omega \leq f\}$$

Can be more aggressive if witness ω strictly improves solution

Redundance and Dominance Rules in VeriPB (Slightly Simplified)

Redundance-based strengthening, optimization version [BGMN23]

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models (\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{f\restriction_\omega \leq f\}$$

Can be more aggressive if witness ω strictly improves solution

Dominance-based strengthening [BGMN23]

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Redundance and Dominance Rules in VeriPB (Slightly Simplified)

Redundance-based strengthening, optimization version [BGMN23]

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models (\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{f\restriction_\omega \leq f\}$$

Can be more aggressive if witness ω strictly improves solution

Dominance-based strengthening [BGMN23]

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

- Applying ω should strictly decrease f
- If so, don't need to show that $(\mathcal{D} \cup \{C\})\restriction_\omega$ implied!

Soundness of Dominance Rule

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}{\restriction_\omega} \cup \{f{\restriction_\omega} < f\}$$

Why is this sound? Let $\mathcal{D} = \emptyset$ for simplicity

Soundness of Dominance Rule

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Why is this sound? Let $\mathcal{D} = \emptyset$ for simplicity

1. Suppose α satisfies $\mathcal{C}$ but falsifies C (i.e., satisfies $\neg C$)

Soundness of Dominance Rule

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Why is this sound? Let $\mathcal{D} = \emptyset$ for simplicity

1. Suppose α satisfies $\mathcal{C}$ but falsifies C (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $\mathcal{C}$ and $f(\alpha \circ \omega) < f(\alpha)$

Soundness of Dominance Rule

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Why is this sound? Let $\mathcal{D} = \emptyset$ for simplicity

1. Suppose α satisfies $\mathcal{C}$ but falsifies C (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $\mathcal{C}$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies C, we're done

Soundness of Dominance Rule

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Why is this sound? Let $\mathcal{D} = \emptyset$ for simplicity

1. Suppose α satisfies $\mathcal{C}$ but falsifies C (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $\mathcal{C}$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies C, we're done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $\mathcal{C}$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$

Soundness of Dominance Rule

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Why is this sound? Let $\mathcal{D} = \emptyset$ for simplicity

1. Suppose α satisfies $\mathcal{C}$ but falsifies C (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $\mathcal{C}$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies C, we're done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $\mathcal{C}$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done

Soundness of Dominance Rule

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Why is this sound? Let $\mathcal{D} = \emptyset$ for simplicity

1. Suppose α satisfies $\mathcal{C}$ but falsifies C (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $\mathcal{C}$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies C, we're done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $\mathcal{C}$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done
6. Otherwise $((\alpha \circ \omega) \circ \omega) \circ \omega$ satisfies $\mathcal{C}$ and $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f((\alpha \circ \omega) \circ \omega)$

Soundness of Dominance Rule

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}{\restriction_\omega} \cup \{f{\restriction_\omega} < f\}$$

Why is this sound? Let $\mathcal{D} = \emptyset$ for simplicity

1. Suppose α satisfies $\mathcal{C}$ but falsifies C (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $\mathcal{C}$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies C, we're done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $\mathcal{C}$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done
6. Otherwise $((\alpha \circ \omega) \circ \omega) \circ \omega$ satisfies $\mathcal{C}$ and $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f((\alpha \circ \omega) \circ \omega)$
7. ...

Soundness of Dominance Rule

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Why is this sound? Let $\mathcal{D} = \emptyset$ for simplicity

1. Suppose α satisfies $\mathcal{C}$ but falsifies C (i.e., satisfies $\neg C$)
2. Then $\alpha \circ \omega$ satisfies $\mathcal{C}$ and $f(\alpha \circ \omega) < f(\alpha)$
3. If $\alpha \circ \omega$ satisfies C, we're done
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies $\mathcal{C}$ and $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done
6. Otherwise $((\alpha \circ \omega) \circ \omega) \circ \omega$ satisfies $\mathcal{C}$ and $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f((\alpha \circ \omega) \circ \omega)$
7. ...
8. Can't go on forever, so finally reach α' satisfying $\mathcal{C} \cup \{C\}$

Soundness of Dominance Rule (Continued)

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Soundness of Dominance Rule (Continued)

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Suppose now that $\mathcal{D} \neq \emptyset$

- Same inductive proof as before, but also nested forward induction over derivation
- Or pick α satisfying $\mathcal{C} \cup \mathcal{D}$ and minimizing f and argue by contradiction

Soundness of Dominance Rule (Continued)

Dominance-based strengthening

Add constraint C to derived set $\mathcal{D}$ if exists witness substitution ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \models \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega < f\}$$

Suppose now that $\mathcal{D} \neq \emptyset$

- Same inductive proof as before, but also nested forward induction over derivation
- Or pick α satisfying $\mathcal{C} \cup \mathcal{D}$ and minimizing f and argue by contradiction

Further extensions:

- Define dominance rule with respect to order independent of objective function
- Switch between different orders in same proof
- See [BGMN23] for details

Three Pseudo-Boolean Proof Logging Vignettes

1. Advanced SAT solving techniques [GN21, BGMN23]
2. Graph solving (subgraph isomorphism) [GMN20, GMM$^+$20]
3. Constraint programming [EGMN20, GMN22, MM23]

Parity (XOR) Reasoning in SAT Solving

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Parity (XOR) Reasoning in SAT Solving

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

Parity (XOR) Reasoning in SAT Solving

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Parity (XOR) Reasoning in SAT Solving

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Exponentially hard for CDCL [Urq87]
But used in CryptoMiniSat [Cry]

Parity (XOR) Reasoning in SAT Solving

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Exponentially hard for CDCL [Urq87]
But used in CryptoMiniSat [Cry]

DRAT proof logging like [PR16] too inefficient in practice!

Parity (XOR) Reasoning in SAT Solving

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$\begin{aligned} x + y + z &= 1 \pmod 2 \\ y + z + w &= 1 \pmod 2 \end{aligned}$$

imply

$$x + w = 0 \pmod 2$$

Exponentially hard for CDCL [Urq87]
But used in CryptoMiniSat [Cry]

DRAT proof logging like [PR16] too inefficient in practice!

Could add XORs to language, but prefer to keep things super-simple

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Use redundance rule with fresh variables a, b to derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("$=$" syntactic sugar for "$\geq$" plus "$\leq$")

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Use redundance rule with fresh variables a, b to derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("$=$" syntactic sugar for "$\geq$" plus "$\leq$")
Add to get

$$x + w + 2y + 2z + 2a + 2b = 6$$

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Use redundance rule with fresh variables a, b to derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("$=$" syntactic sugar for "$\geq$" plus "$\leq$")
Add to get

$$x + w + 2y + 2z + 2a + 2b = 6$$

From this can extract

$$x + \overline{w} \geq 1$$
$$\overline{x} + w \geq 1$$

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Use redundance rule with fresh variables a, b to derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("$=$" syntactic sugar for "$\geq$" plus "$\leq$")
Add to get

$$x + w + 2y + 2z + 2a + 2b = 6$$

From this can extract

$$x + \overline{w} \geq 1$$
$$\overline{x} + w \geq 1$$

VeriPB can certify XOR reasoning [GN21]

Symmetry Breaking in SAT Solving

1. Pretend to solve optimisation problem minimizing $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$
(search for lexicographically smallest assignment satisfying formula)

Symmetry Breaking in SAT Solving

1. Pretend to solve optimisation problem minimizing $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$ (search for lexicographically smallest assignment satisfying formula)
2. Derive (for proof log only) pseudo-Boolean version of lex-leader constraint

$$f \leq f\restriction_\sigma \quad \doteq \quad \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

Symmetry Breaking in SAT Solving

1. Pretend to solve optimisation problem minimizing $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$ (search for lexicographically smallest assignment satisfying formula)
2. Derive (for proof log only) pseudo-Boolean version of lex-leader constraint

$$f \leq f\restriction_\sigma \quad \doteq \quad \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

3. Derive symmetry breaking clauses from this PB constraint:

$$y_0 \qquad\qquad \overline{y}_j \vee \overline{\sigma(x_j)} \vee x_j$$

$$\overline{y}_{j-1} \vee \overline{x}_j \vee \sigma(x_j) \qquad\qquad y_j \vee \overline{y}_{j-1} \vee \overline{x}_j$$

$$\overline{y}_j \vee y_{j-1} \qquad\qquad y_j \vee \overline{y}_{j-1} \vee \sigma(x_j)$$

Symmetry Breaking in SAT Solving

1. Pretend to solve optimisation problem minimizing $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$ (search for lexicographically smallest assignment satisfying formula)
2. Derive (for proof log only) pseudo-Boolean version of lex-leader constraint

$$f \leq f\restriction_\sigma \quad \doteq \quad \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

3. Derive symmetry breaking clauses from this PB constraint:

$$y_0 \geq 1 \qquad \overline{y}_j + \overline{\sigma(x_j)} + x_j \geq 1$$

$$\overline{y}_{j-1} + \overline{x}_j + \sigma(x_j) \geq 1 \qquad y_j + \overline{y}_{j-1} + \overline{x}_j \geq 1$$

$$\overline{y}_j + y_{j-1} \geq 1 \qquad y_j + \overline{y}_{j-1} + \sigma(x_j) \geq 1$$

Symmetry Breaking in SAT Solving

1. Pretend to solve optimisation problem minimizing $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$ (search for lexicographically smallest assignment satisfying formula)
2. Derive (for proof log only) pseudo-Boolean version of lex-leader constraint

$$f \leq f\restriction_\sigma \quad \doteq \quad \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

3. Derive symmetry breaking clauses from this PB constraint:

$$\begin{aligned} y_0 &\geq 1 & \overline{y}_j + \overline{\sigma(x_j)} + x_j &\geq 1 \\ \overline{y}_{j-1} + \overline{x}_j + \sigma(x_j) &\geq 1 & y_j + \overline{y}_{j-1} + \overline{x}_j &\geq 1 \\ \overline{y}_j + y_{j-1} &\geq 1 & y_j + \overline{y}_{j-1} + \sigma(x_j) &\geq 1 \end{aligned}$$

VeriPB can certify fully general SAT symmetry breaking [BGMN23]

The Subgraph Isomorphism Problem

Input

- Pattern graph $\mathcal{P}$ with vertices $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{u, v, w, \ldots\}$

The Subgraph Isomorphism Problem

Input

- Pattern graph $\mathcal{P}$ with vertices $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{u, v, w, \ldots\}$

Task

- Find all subgraph isomorphisms $\varphi : V(\mathcal{P}) \to V(\mathcal{T})$
- I.e., if
  1. $\varphi(a) = u$
  2. $\varphi(b) = v$
  3. $(a, b) \in E(\mathcal{P})$

  then must have $(u, v) \in E(\mathcal{T})$

Pseudo-Boolean Proof Logging for Subgraph Isomorphism Solving

All reasoning steps in Glasgow Subgraph Solver [ADH+19, GSS] can be formalized efficiently in the cutting planes proof system [GMN20]

Pseudo-Boolean Proof Logging for Subgraph Isomorphism Solving

All reasoning steps in Glasgow Subgraph Solver [ADH$^+$19, GSS] can be formalized efficiently in the cutting planes proof system [GMN20]

Means that

1. Solver can justify each step by writing local formal derivation

Pseudo-Boolean Proof Logging for Subgraph Isomorphism Solving

All reasoning steps in Glasgow Subgraph Solver [ADH+19, GSS] can be formalized efficiently in the cutting planes proof system [GMN20]

Means that

1. Solver can justify each step by writing local formal derivation
2. Local derivations can be chained into global correctness proof

Pseudo-Boolean Proof Logging for Subgraph Isomorphism Solving

All reasoning steps in Glasgow Subgraph Solver [ADH+19, GSS] can be formalized efficiently in the cutting planes proof system [GMN20]

Means that

1. Solver can justify each step by writing local formal derivation
2. Local derivations can be chained into global correctness proof
3. Proof checkable by stand-alone verifier that knows nothing about graphs

Pseudo-Boolean Proof Logging for Subgraph Isomorphism Solving

All reasoning steps in Glasgow Subgraph Solver [ADH+19, GSS] can be formalized efficiently in the cutting planes proof system [GMN20]

Means that

1. Solver can justify each step by writing local formal derivation
2. Local derivations can be chained into global correctness proof
3. Proof checkable by stand-alone verifier that knows nothing about graphs
4. Strong correctness guarantees:
   - Even for buggy solver, a correct proof is always accepted
   - Even for formally verified solver that gets whacked by cosmic radiation/hardware failure, wrong proof will always be rejected

Subgraph Isomorphism as a Pseudo-Boolean Formula

- Pattern graph $\mathcal{P}$ with $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with $V(\mathcal{T}) = \{u, v, w, \ldots\}$
- No loops (for simplicity)

Subgraph Isomorphism as a Pseudo-Boolean Formula

- Pattern graph $\mathcal{P}$ with $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with $V(\mathcal{T}) = \{u, v, w, \ldots\}$
- No loops (for simplicity)

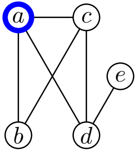
Pseudo-Boolean encoding

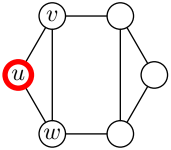
$$\sum_{v \in V(\mathcal{T})} x_{a \mapsto v} = 1 \qquad \text{[every } a \text{ maps somewhere]}$$

$$\sum_{b \in V(\mathcal{P})} \overline{x}_{b \mapsto u} \geq |V(\mathcal{P})| - 1 \qquad \text{[mapping is one-to-one]}$$

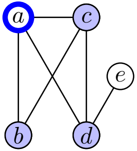
$$\overline{x}_{a \mapsto u} + \sum_{v \in N(u)} x_{b \mapsto v} \geq 1 \qquad \text{[edge } (a, b) \text{ maps to edge } (u, v)\text{]}$$

Pseudo-Boolean Proof Logging Example: Degree Preprocessing

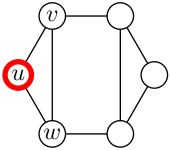




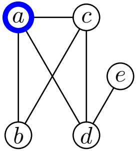
Pseudo-Boolean Proof Logging Example: Degree Preprocessing



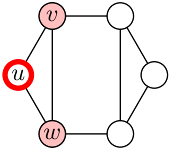
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$



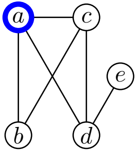
Pseudo-Boolean Proof Logging Example: Degree Preprocessing



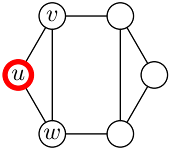
$$\begin{aligned}
\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} &\geq 4 \\
\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} &\geq 4
\end{aligned}$$



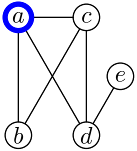
Pseudo-Boolean Proof Logging Example: Degree Preprocessing



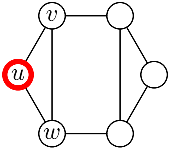
$$\begin{aligned}
\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} &\geq 1\\
\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} &\geq 1\\
\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} &\geq 1\\
\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} &\geq 4\\
\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} &\geq 4\\
x_{a\mapsto v} &\geq 0\\
x_{a\mapsto w} &\geq 0\\
x_{e\mapsto v} &\geq 0\\
x_{e\mapsto w} &\geq 0
\end{aligned}$$



Pseudo-Boolean Proof Logging Example: Degree Preprocessing

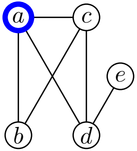


$$\begin{aligned}
\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} &\geq 4 \\
\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} &\geq 4 \\
x_{a\mapsto v} &\geq 0 \\
x_{a\mapsto w} &\geq 0 \\
x_{e\mapsto v} &\geq 0 \\
x_{e\mapsto w} &\geq 0
\end{aligned}$$

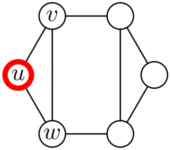


Sum up all constraints & divide by 3 to obtain

Pseudo-Boolean Proof Logging Example: Degree Preprocessing



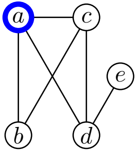
$$
\begin{aligned}
\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} &\geq 4 \\
\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} &\geq 4 \\
x_{a\mapsto v} &\geq 0 \\
x_{a\mapsto w} &\geq 0 \\
x_{e\mapsto v} &\geq 0 \\
x_{e\mapsto w} &\geq 0
\end{aligned}
$$



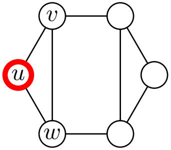
Sum up all constraints & divide by 3 to obtain

$$3\overline{x}_{a\mapsto u} + 10 \geq 11$$

Pseudo-Boolean Proof Logging Example: Degree Preprocessing



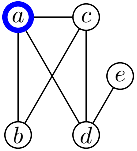
$$
\begin{aligned}
\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} &\geq 1\\
\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} &\geq 1\\
\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} &\geq 1\\
\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} &\geq 4\\
\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} &\geq 4\\
x_{a\mapsto v} &\geq 0\\
x_{a\mapsto w} &\geq 0\\
x_{e\mapsto v} &\geq 0\\
x_{e\mapsto w} &\geq 0
\end{aligned}
$$

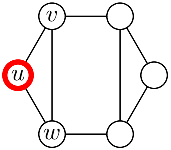


Sum up all constraints & divide by 3 to obtain

$$3\overline{x}_{a\mapsto u} \geq 1$$

Pseudo-Boolean Proof Logging Example: Degree Preprocessing





$$
\begin{aligned}
\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} &\geq 1 \\
\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} &\geq 4 \\
\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} &\geq 4 \\
x_{a\mapsto v} &\geq 0 \\
x_{a\mapsto w} &\geq 0 \\
x_{e\mapsto v} &\geq 0 \\
x_{e\mapsto w} &\geq 0
\end{aligned}
$$

Sum up all constraints & divide by 3 to obtain

$$
\begin{aligned}
3\overline{x}_{a\mapsto u} &\geq 1 \\
\overline{x}_{a\mapsto u} &\geq 1
\end{aligned}
$$

Integer Variables in Constraint Programming (1/2)

How to deal with integer variables?

Given $A \in \{-3 \dots 9\}$, the direct encoding is:

$$\begin{aligned} a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3} \\ + a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} = 1 \end{aligned}$$

Integer Variables in Constraint Programming (1/2)

How to deal with integer variables?

Given $A \in \{-3 \ldots 9\}$, the direct encoding is:

$$\begin{aligned} a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3} \\ + a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} = 1 \end{aligned}$$

This doesn't work for large domains...

Integer Variables in Constraint Programming (1/2)

How to deal with integer variables?

Given $A \in \{-3 \ldots 9\}$, the direct encoding is:

$$\begin{aligned} a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3} \\ + a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} = 1 \end{aligned}$$

This doesn't work for large domains. . .

We can instead use a binary encoding:

$$\begin{aligned} -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} &\geq -3 \qquad \text{and} \\ 16a_{\text{neg}} + -1a_{\text{b0}} + -2a_{\text{b1}} + -4a_{\text{b2}} + -8a_{\text{b3}} &\geq -9 \end{aligned}$$

Doesn't propagate much, but that isn't a problem for proof logging

Integer Variables in Constraint Programming (2/2)

We can mix binary and order encodings! Define linear inequalities encoding

$$a_{\geq 4} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4$$

$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$

$$a_{=4} \Leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}$$

Integer Variables in Constraint Programming (2/2)

We can mix binary and order encodings! Define linear inequalities encoding

$$a_{\geq 4} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4$$
$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$
$$a_{=4} \Leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}$$

When creating $a_{\geq i}$, also introduce pseudo-Boolean constraints encoding

$$a_{\geq i} \Rightarrow a_{\geq j} \quad \text{and} \quad a_{\geq h} \Rightarrow a_{\geq i}$$

for the closest values $j < i < h$ that already exist

Integer Variables in Constraint Programming (2/2)

We can mix binary and order encodings! Define linear inequalities encoding

$$a_{\geq 4} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4$$
$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$
$$a_{=4} \Leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}$$

When creating $a_{\geq i}$, also introduce pseudo-Boolean constraints encoding

$$a_{\geq i} \Rightarrow a_{\geq j} \quad \text{and} \quad a_{\geq h} \Rightarrow a_{\geq i}$$

for the closest values $j < i < h$ that already exist

We can do this:

- Inside the pseudo-Boolean model where needed
- Otherwise lazily during proof logging

Table Constraints

Constraints can be specified extensionally as list of feasible tuples, called a table

Variable assignments must match some row in table

Table Constraints

Constraints can be specified extensionally as list of feasible tuples, called a table
Variable assignments must match some row in table

Given table constraint

$$(A, B, C) \in [(1,2,3), (1,3,4), (2,2,5)]$$

define

$$3\bar{t}_1 + a_{=1} + b_{=2} + c_{=3} \geq 3 \qquad \text{i.e.,} \quad t_1 \Rightarrow (a_{=1} \wedge b_{=2} \wedge c_{=3})$$
$$3\bar{t}_2 + a_{=1} + b_{=4} + c_{=4} \geq 3 \qquad \text{i.e.,} \quad t_2 \Rightarrow (a_{=1} \wedge b_{=4} \wedge c_{=4})$$
$$3\bar{t}_3 + a_{=2} + b_{=2} + c_{=5} \geq 3 \qquad \text{i.e.,} \quad t_3 \Rightarrow (a_{=2} \wedge b_{=2} \wedge c_{=5})$$

using tuple selector variables

$$t_1 + t_2 + t_3 = 1$$

A Constraint Programming Solver with Pseudo-Boolean Proof Logging

Proof-of-concept constraint programming solver at

`https://github.com/ciaranm/glasgow-constraint-solver`

Supports proof logging for global constraints including:

- All-different
- Integer linear inequality (including for very large domains)
- Smart table and regular
- Minimum / maximum of an array
- Element (kind of array indexing)
- Absolute value
- (Hamiltonian) Circuit

Details in [EGMN20, GMN22, MM23]

Using VeriPB for SAT Solving

1. Use dedicated tools for Gaussian elimination [GN21], symmetry breaking [BGMN23], PB-to-CNF translation [GMNO22], et cetera
2. Concatenate with CDCL solver DRAT proof rewritten in VeriPB format (`https://gitlab.com/MIAOresearch/tools-and-utilities/kissat_fork`)

Using VeriPB for SAT Solving

1. Use dedicated tools for Gaussian elimination [GN21], symmetry breaking [BGMN23], PB-to-CNF translation [GMNO22], et cetera
2. Concatenate with CDCL solver DRAT proof rewritten in VeriPB format (`https://gitlab.com/MIAOresearch/tools-and-utilities/kissat_fork`)

Short dictionary for DRAT-to-VeriPB translations

DRAT	VeriPB
`1`	`x1`
`-2`	`~x2`
`1 -2 3 0`	`1 x1 1 ~x2 1 x3 >= 1 ;`
`1 -2 3 0` is RUP	`rup 1 x1 1 ~x2 1 x3 >= 1 ;`
`1 -2 3 0` is RAT	`red 1 x1 1 ~x2 1 x3 >= 1 ; x1 -> 1`

Using VeriPB for SAT Solving

1. Use dedicated tools for Gaussian elimination [GN21], symmetry breaking [BGMN23], PB-to-CNF translation [GMNO22], et cetera
2. Concatenate with CDCL solver DRAT proof rewritten in VeriPB format (https://gitlab.com/MIAOresearch/tools-and-utilities/kissat_fork)

Short dictionary for DRAT-to-VeriPB translations

DRAT	VeriPB
1	x1
-2	~x2
1 -2 3 0	1 x1 1 ~x2 1 x3 >= 1 ;
1 -2 3 0 is RUP	rup 1 x1 1 ~x2 1 x3 >= 1 ;
1 -2 3 0 is RAT	red 1 x1 1 ~x2 1 x3 >= 1 ; x1 -> 1

3. But LRAT syntactically rewritten for VeriPB should allow way faster proof checking — see latest version of CaDiCaL [CaD]

VeriPB Documentation

VeriPB tutorial at *CP '22* [BMN22]

- video at `youtu.be/s_5BIi4I22w`
- updated slides for *IJCAI '23* tutorial [BMN23]

Description of VeriPB and CakePB [BMM+23] for SAT 2023 competition

- Available at `satcompetition.github.io/2023/checkers.html`

Specific details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23]

Lots of concrete example files at `gitlab.com/MIAOresearch/software/VeriPB`

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging

- Trim proof while verifying (as in DRAT-trim [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker *(work in progress [BMM+23])*

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging

- Trim proof while verifying (as in DRAT-trim [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker *(work in progress [BMM+23])*

Proof logging for other combinatorial problems and techniques

- Model counting
- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(work on SCIP in [CGS17, EG21, DEGH23])*
- Satisfiability modulo theories (SMT) solving *(work on cvc5, Z3, ... [BBC+23])*

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging

- Trim proof while verifying (as in DRAT-trim [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker *(work in progress [BMM+23])*

Proof logging for other combinatorial problems and techniques

- Model counting
- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(work on SCIP in [CGS17, EG21, DEGH23])*
- Satisfiability modulo theories (SMT) solving *(work on cvc5, Z3, ... [BBC+23])*

And more...

- Use proof logs for algorithm analysis or explainability purposes
- Lots of other challenging problems and interesting ideas

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging

- Trim proof while verifying (as in DRAT-trim [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker *(work in progress [BMM+23])*

Proof logging for other combinatorial problems and techniques

- Model counting
- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(work on SCIP in [CGS17, EG21, DEGH23])*
- Satisfiability modulo theories (SMT) solving *(work on cvc5, Z3, ... [BBC+23])*

And more...

- Use proof logs for algorithm analysis or explainability purposes
- Lots of other challenging problems and interesting ideas
- We're hiring! Talk to me to join the pseudo-Boolean proof logging revolution! ☺

Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **Action point:** What problems can VeriPB solve for you? ☺

Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **Action point:** What problems can VeriPB solve for you? ☺

Thank you for your attention!

References I

[ABM+11] Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, Christine Rizkallah, and Pascal Schweitzer. An introduction to certifying algorithms. *it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 53(6):287–293, December 2011.

[ADH+19] Blair Archibald, Fraser Dunlop, Ruth Hoffmann, Ciaran McCreesh, Patrick Prosser, and James Trimble. Sequential and parallel solution-biased search for subgraph algorithms. In *Proceedings of the 16th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '19)*, volume 11494 of *Lecture Notes in Computer Science*, pages 20–38. Springer, June 2019.

[AGJ+18] Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Metamorphic testing of constraint solvers. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP '18)*, volume 11008 of *Lecture Notes in Computer Science*, pages 727–736. Springer, August 2018.

[AW13] Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In Michael Jünger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization*, pages 449–481. Springer, 2013.

References II

[Bar95] Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, January 1995.

[BB09] Robert Brummayer and Armin Biere. Fuzzing and delta-debugging SMT solvers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories (SMT '09)*, pages 1–5, August 2009.

[BBC+23] Haniel Barbosa, Clark Barrett, Byron Cook, Bruno Dutertre, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Cesare Tinelli, and Yoni Zohar. Generating and exploiting automated reasoning proof certificates. *Communications of the ACM*, 66(10):86—95, October 2023.

[BBN+23] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, July 2023.

[BGMN23] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI '22*.

References III

[BHvMW21] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, February 2021.

[BLB10] Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, July 2010.

[BMM+23] Bart Bogaerts, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. Documentation of VeriPB and CakePB for the SAT competition 2023. Available at `https://satcompetition.github.io/2023/checkers.html`, March 2023.

[BMN22] Bart Bogaerts, Ciaran McCreesh, and Jakob Nordström. Solving with provably correct results: Beyond satisfiability, and towards constraint programming. Tutorial at the *28th International Conference on Principles and Practice of Constraint Programming*. Slides available at `http://www.jakobnordstrom.se/presentations/`, August 2022.

References IV

[BMN23] Bart Bogaerts, Ciaran McCreesh, and Jakob Nordström. Combinatorial solving with provably correct results. Tutorial at the *32nd International Joint Conference on Artificial Intelligence*. Slides available at `http://www.jakobnordstrom.se/presentations/`, August 2023.

[BR07] Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming—A look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, February 2007.

[BT19] Samuel R. Buss and Neil Thapen. DRAT proofs, propagation redundancy, and extended resolution. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT '19)*, volume 11628 of *Lecture Notes in Computer Science*, pages 71–89. Springer, July 2019.

[CaD] CaDiCaL. `http://fmv.jku.at/cadical/`.

[CCT87] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

References V

[CGS17] Kevin K. H. Cheung, Ambros M. Gleixner, and Daniel E. Steffy. Verifying integer programming results. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO '17)*, volume 10328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, June 2017.

[CHH+17] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, August 2017.

[CKSW13] William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, September 2013.

[CMS17] Luís Cruz-Filipe, João P. Marques-Silva, and Peter Schneider-Kamp. Efficient certified resolution proof checking. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 118–135. Springer, April 2017.

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

References VI

[DEGH23] Jasper van Doornmalen, Leon Eifler, Ambros Gleixner, and Christopher Hojny. A proof system for certifying symmetry and optimality reasoning in integer programming. Technical Report 2311.03877, arXiv.org, November 2023.

[EG21] Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. In *Proceedings of the 22nd International Conference on Integer Programming and Combinatorial Optimization (IPCO '21)*, volume 12707 of *Lecture Notes in Computer Science*, pages 163–177. Springer, May 2021.

[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 (AAAI '20)*, pages 1486–1494, February 2020.

[GMM$^+$20] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.

References VII

[GMN20] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 1134–1140, July 2020.

[GMN22] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In *Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP '22)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, August 2022.

[GMNO22] Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:25, August 2022.

[GN21] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.

References VIII

[Goc22] Stephan Gocht. *Certifying Correctness for Combinatorial Algorithms by Using Pseudo-Boolean Reasoning*. PhD thesis, Lund University, June 2022. Available at `https://portal.research.lu.se/en/publications/certifying-correctness-for-combinatorial-algorithms-by-using-pseu`.

[GS19] Graeme Gange and Peter Stuckey. Certifying optimality in constraint programming. Presentation at KTH Royal Institute of Technology. Slides available at `https://www.kth.se/polopoly_fs/1.879851.1550484700!/CertifiedCP.pdf`, February 2019.

[GSD19] Xavier Gillard, Pierre Schaus, and Yves Deville. SolverCheck: Declarative testing of constraints. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 565–582. Springer, October 2019.

[GSS] The Glasgow subgraph solver. `https://github.com/ciaranm/glasgow-subgraph-solver`.

[HHW13a] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, pages 181–188, October 2013.

References IX

[HHW13b] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, June 2013.

[JHB12] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR '12)*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, June 2012.

[KB22] Daniela Kaufmann and Armin Biere. Fuzzing and delta debugging and-inverter graph verification tools. In *Proceedings of the 16th International Conference on Tests and Proofs (TAP '22)*, volume 13361 of *Lecture Notes in Computer Science*, pages 69–88. Springer, July 2022.

[KM21] Sonja Kraiczy and Ciaran McCreesh. Solving graph homomorphism and subgraph isomorphism problems faster through clique neighbourhood constraints. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI '21)*, pages 1396–1402, August 2021.

References X

[MM23] Matthew McIlree and Ciaran McCreesh. Proof logging for smart extensional constraints. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP '23)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:17, August 2023.

[MMNS11] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, May 2011.

[NPB22] Aina Niemetz, Mathias Preiner, and Clark W. Barrett. Murxla: A modular and highly extensible API fuzzer for SMT solvers. In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV '22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 92–106. Springer, August 2022.

[PB23] Tobias Paxian and Armin Biere. Uncovering and classifying bugs in MaxSAT solvers through fuzzing and delta debugging. In *Proceedings of the 14th International Workshop on Pragmatics of SAT*, volume 3545 of *CEUR Workshop Proceedings*, pages 59–71. CEUR-WS.org, July 2023.

[PR16] Tobias Philipp and Adrián Rebola-Pardo. DRAT proofs for XOR reasoning. In *Proceedings of the 15th European Conference on Logics in Artificial Intelligence (JELIA '16)*, volume 10021 of *Lecture Notes in Computer Science*, pages 415–429. Springer, November 2016.

References XI

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.

[VDB22] Dieter Vandesande, Wolf De Wulf, and Bart Bogaerts. QMaxSATpb: A certified MaxSAT solver. In *Proceedings of the 16th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR '22)*, volume 13416 of *Lecture Notes in Computer Science*, pages 429–442. Springer, September 2022.

[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.