A One-Size-Fits-All Proof Logging System?

Jakob Nordström

University of Copenhagen
and Lund University

14th Pragmatics of SAT Workshop
Alghero, Italy
July 4, 2023

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
  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
  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

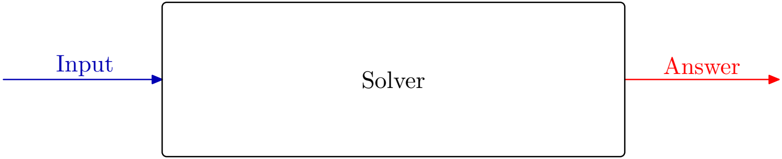
What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  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

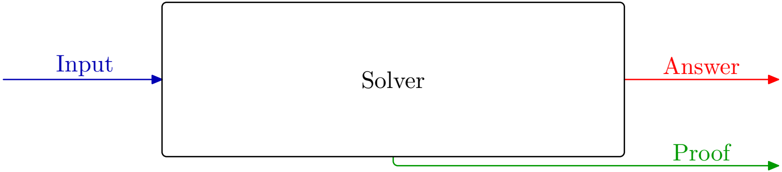
- **Proof logging**
  Make solver certifying [ABM+11, MMNS11] by outputting
  1. not only answer but also
  2. simple, machine-verifiable proof that answer is correct

Proof Logging with Certifying Solvers: Workflow



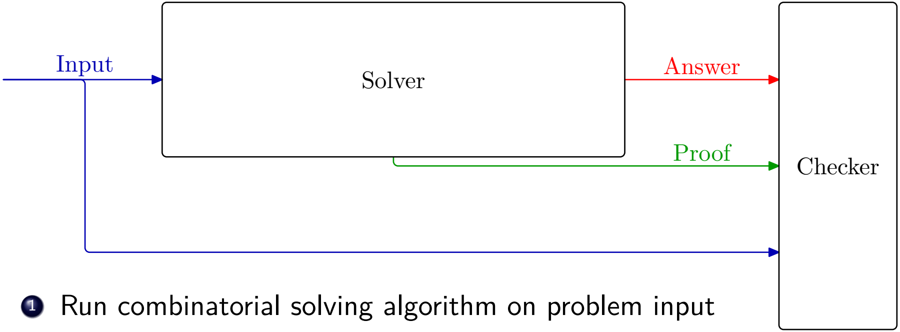
1. Run combinatorial solving algorithm on problem input

Proof Logging with Certifying Solvers: Workflow



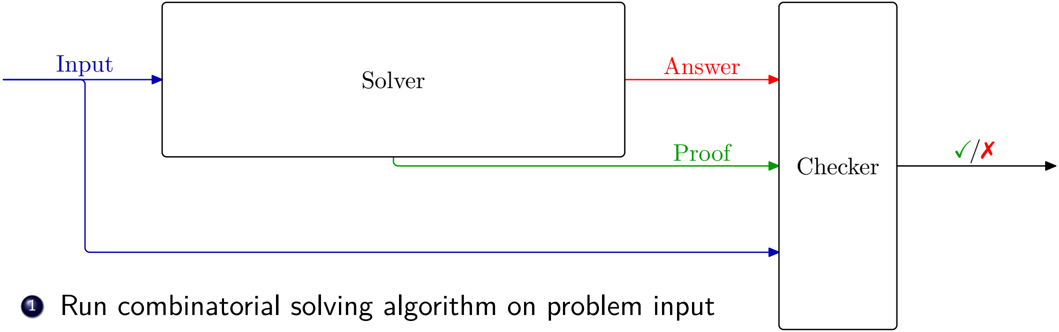
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

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

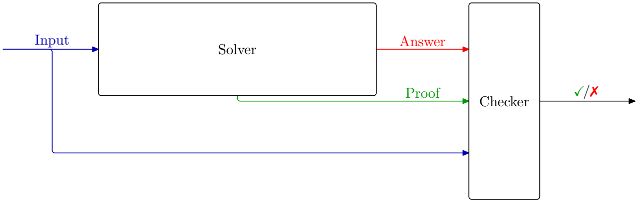
Proof Logging with Certifying Solvers: Workflow



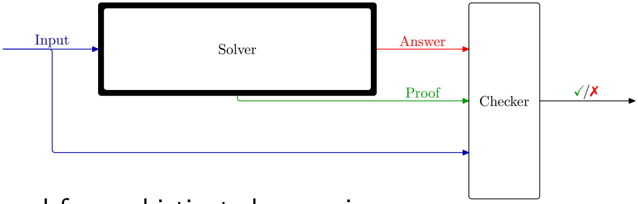
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

Proof Logging Desiderata

Proof format for certifying solver should be



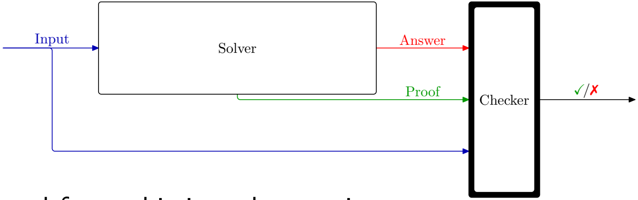
Proof Logging Desiderata



Proof format for certifying solver should be

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

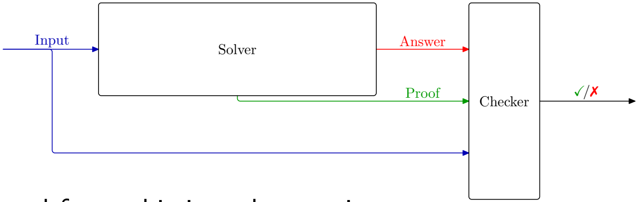
Proof Logging Desiderata



Proof format for certifying solver should be

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

Proof Logging Desiderata

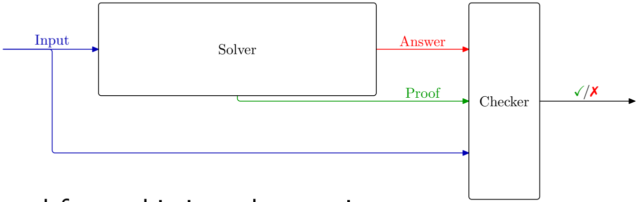


Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning
- **dead simple:** checking correctness of proofs should be 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 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, BGMN22]
- 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, BGMN22]
- 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, BGMN22]
- Implemented in VeriPB (https://gitlab.com/MIAOresearch/software/VeriPB)

Purpose of this talk:

1. Marketing pitch ☺
2. Solicit feedback

Pseudo-Boolean Constraints

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

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
- 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
- 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
- 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
- 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 sometime in the future

Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging

Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging

Otherwise

- do trusted or verified translation to 0-1 ILP
- provide proof logging for 0-1 ILP formulation

Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging

Otherwise

- do trusted or verified translation to 0-1 ILP
- provide proof logging for 0-1 ILP formulation

Goldilocks compromise between expressivity and simplicity:

1. 0-1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments (even for SAT)
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

Otherwise

- do trusted or verified translation to 0-1 ILP
- provide proof logging for 0-1 ILP formulation

Goldilocks compromise between expressivity and simplicity:

1. 0-1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments (even for SAT)
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

Otherwise

- do trusted or verified translation to 0-1 ILP
- provide proof logging for 0-1 ILP formulation

Goldilocks compromise between expressivity and simplicity:

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

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

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

VeriPB Proof Structure

1. **Preamble**
   Load input formula
   Specify settings
2. **Derivation section**
   Derivations of new constraints
   Logging of solutions
3. **Output section**
   Listing of constraints currently in database
   Input to next stage (or for debugging)
4. **Conclusions section**
   Specification of what was established
   - satisfiability / unsatisfiability
   - optimality
   - enumeration of solutions

VeriPB Proof Structure: Syntax

```
pseudo-Boolean proof version 2.0
f ⟨M⟩
preserve ⟨var1⟩ ⟨var2⟩ ... ⟨varN⟩
⟨derivation part⟩
output ⟨output part⟩
conclusion ⟨conclusion part⟩
end pseudo-Boolean proof
```

VeriPB Proof Configuration

Core set $\mathcal{C}$

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

Derived set $\mathcal{D}$

- All constraints derived during search
- Also intermediate constraints used in proof logging

VeriPB Proof Configuration

Core set $\mathcal{C}$

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

Derived set $\mathcal{D}$

- All constraints derived during search
- Also intermediate constraints used in proof logging

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 ∞

Order $\mathcal{O}$

- Pseudo-Boolean formula encoding pre-order (reflexive and transitive)
- Syntactic proof of properties required
- Applied to specified variable set $\vec{z}$

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 $c \in \mathbb{N}^+$
$$\frac{\sum_i a_i\ell_i \geq A}{\sum_i ca_i\ell_i \geq cA}$$

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 $c \in \mathbb{N}^+$
$$\frac{\sum_i a_i\ell_i \geq A}{\sum_i ca_i\ell_i \geq cA}$$

Division for any $c \in \mathbb{N}^+$
(constraint in normalized form)
$$\frac{\sum_i a_i\ell_i \geq A}{\sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil}$$

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

Mul by 2 $$\frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$$

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

Mul by 2
$$\frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$$

Add
$$\frac{2w + 4x + 2y \geq 4 \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}$$

Mul by 2
$$\frac{\overline{z} \geq 0}{2\overline{z} \geq 0}$$

Add
$$\frac{3w + 6x + 6y + 2z \geq 9 \qquad 2\overline{z} \geq 0}{3w + 6x + 6y \geq 7}$$

Div by 3
$$\frac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 2\tfrac{1}{3}}$$

Cutting Planes Toy Example

$$\text{Mul by } 2 \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$$

$$\text{Add} \quad \frac{2w + 4x + 2y \geq 4 \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}$$

$$\frac{\overline{z} \geq 0}{2\overline{z} \geq 0} \quad \text{Mul by } 2$$

$$\text{Add} \quad \frac{3w + 6x + 6y + 2z \geq 9 \qquad 2\overline{z} \geq 0}{3w + 6x + 6y \geq 7}$$

$$\text{Div by } 3 \quad \frac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 3}$$

Cutting Planes Toy Example

Mul by 2: $$\frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$$

Mul by 2: $$\frac{\overline{z} \geq 0}{2\overline{z} \geq 0}$$

Add: $$\frac{2w + 4x + 2y \geq 4 \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}$$

Add: $$\frac{3w + 6x + 6y + 2z \geq 9 \qquad 2\overline{z} \geq 0}{3w + 6x + 6y \geq 7}$$

Div by 3: $$\frac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 3}$$

Such a calculation can be written in a proof line assuming handles

$$\begin{aligned} C_1 &\doteq 2x + y + w \geq 2 \\ C_2 &\doteq 2x + 4y + 2z + w \geq 5 \\ Ax(\overline{z}) &\doteq \overline{z} \geq 0 \end{aligned}$$

Cutting Planes Toy Example

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

Such a calculation can be written in a proof line assuming handles

$$
\begin{aligned}
C_1 &\doteq 2x + y + w \geq 2 \\
C_2 &\doteq 2x + 4y + 2z + w \geq 5 \\
Ax(\overline{z}) &\doteq \overline{z} \geq 0
\end{aligned}
$$

using postfix notation something like

$$C_1 \;\; 2 \;\; \texttt{Mul} \;\; C_2 \;\; \texttt{Add} \;\; Ax(\overline{z}) \;\; 2 \;\; \texttt{Mul} \;\; \texttt{Add} \;\; 3 \;\; \texttt{Div}$$

More About VeriPB Proofs

Variables

- start with a letter in `A-Z` or `a-z`
- continue with characters in `A-Z`, `a-z`, `0-9`, or square and curly brackets, hyphen, underscore, and caret
- contain at least two characters

Constraints

Are referred to by positive integers (constraint IDs)

Derivation rules and requirements

Come in two flavours

1. kernel format for formally verified proof checker
2. augmented format with convenience rules such as reverse unit propagation (RUP)

Strengthening Rules

Witness ω: substitution mapping variables to truth values or literals

Redundance-based strengthening (witness ω show how to "patch assignment")

Derive constraint C from $\mathcal{C} \cup \mathcal{D}$ if exists witness ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash (\mathcal{C} \cup \mathcal{D} \cup \{C\})\!\upharpoonright_\omega \cup \{f\!\upharpoonright_\omega \leq f\} \cup \mathcal{O}(\vec{z}\!\upharpoonright_\omega, \vec{z})$$

Strengthening Rules

Witness ω: substitution mapping variables to truth values or literals

Redundance-based strengthening (witness ω show how to "patch assignment")

Derive constraint C from $\mathcal{C} \cup \mathcal{D}$ if exists witness ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash (\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{f\restriction_\omega \leq f\} \cup \mathcal{O}(\vec{z}\restriction_\omega, \vec{z})$$

Dominance-based strengthening (witness ω "drives down potential")

Derive constraint C from $\mathcal{C} \cup \mathcal{D}$ if exists witness ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega \leq f\} \cup \mathcal{O}(\vec{z}\restriction_\omega, \vec{z}) \cup \neg\mathcal{O}(\vec{z}, \vec{z}\restriction_\omega)$$

Strengthening Rules

Witness ω: substitution mapping variables to truth values or literals

Redundance-based strengthening (witness ω show how to "patch assignment")

Derive constraint C from $\mathcal{C} \cup \mathcal{D}$ if exists witness ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash (\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{f\restriction_\omega \leq f\} \cup \mathcal{O}(\vec{z}\restriction_\omega, \vec{z})$$

Dominance-based strengthening (witness ω "drives down potential")

Derive constraint C from $\mathcal{C} \cup \mathcal{D}$ if exists witness ω such that

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash \mathcal{C}\restriction_\omega \cup \{f\restriction_\omega \leq f\} \cup \mathcal{O}(\vec{z}\restriction_\omega, \vec{z}) \cup \neg\mathcal{O}(\vec{z}, \vec{z}\restriction_\omega)$$

- Witness ω should be specified in proof log
- Derivations should also be explicit, or be "obvious" to proof checker (like by RUP)

Checked and Unchecked Deletion

Important to allow deletions of constraints from database

But powerful strengthening rules create problems:

- Unsatisfiable formulas can turn satisfiable
- Satisfiable formulas can turn unsatisfiable(!)

Checked and Unchecked Deletion

Important to allow deletions of constraints from database
But powerful strengthening rules create problems:

- Unsatisfiable formulas can turn satisfiable
- Satisfiable formulas can turn unsatisfiable(!)

Solution: distinguish between deletion from core set $\mathcal{C}$ and derived set $\mathcal{D}$
(For SAT solvers, support generic delete command in augmented format that translates to right type of deletion behind the scenes)

Checked and Unchecked Deletion

Important to allow deletions of constraints from database
But powerful strengthening rules create problems:

- Unsatisfiable formulas can turn satisfiable
- Satisfiable formulas can turn unsatisfiable(!)

Solution: distinguish between deletion from core set $\mathcal{C}$ and derived set $\mathcal{D}$
(For SAT solvers, support generic delete command in augmented format that translates to right type of deletion behind the scenes)

Deletion of constraint C is:

1. always OK from derived set $\mathcal{D}$
2. OK from core set $\mathcal{C}$ only if C can be rederived from $\mathcal{C} \setminus \{C\}$ with redundance rule

Checked and Unchecked Deletion

Important to allow deletions of constraints from database
But powerful strengthening rules create problems:

- Unsatisfiable formulas can turn satisfiable
- Satisfiable formulas can turn unsatisfiable(!)

Solution: distinguish between deletion from core set $\mathcal{C}$ and derived set $\mathcal{D}$
(For SAT solvers, support generic delete command in augmented format that translates to right type of deletion behind the scenes)

Deletion of constraint C is:

1. always OK from derived set $\mathcal{D}$
2. OK from core set $\mathcal{C}$ only if C can be rederived from $\mathcal{C} \setminus \{C\}$ with redundance rule (otherwise unchecked deletion — special conditions apply)

Conclusions for Decision Problems

`NONE`
Status is undetermined

`SAT [ :` ⟨*assignment*⟩`]`
Propagate given assignment w.r.t. database, then check against original formula
If no assignment given, then

- solution should have been logged
- no unchecked deletion must have occurred

`UNSAT [ :` ⟨*constraint ID*⟩`]`
Only valid if no solution has been logged
Check that specified constraint is contradictory (technically: negative slack)
If no constraint given, check that database unit propagates to contradiction

Optimization Problems

Any solution α found is logged with `soli` "log solution and improve" command

- α checked against current core set $\mathcal{C}$
- Objective-improving constraint $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \cdot \alpha(\ell_i)$ added to core set (forces search for better solutions)

Optimization Problems

Any solution α found is logged with `soli` "log solution and improve" command

- α checked against current core set $\mathcal{C}$
- Objective-improving constraint $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \cdot \alpha(\ell_i)$ added to core set (forces search for better solutions)

Note that

- α need not be solution for original formula
- but such solution can be reconstructed from the proof

Proof format supports not just optimality, but also non-tight upper and lower bounds

Conclusions for Optimization Problems

`NONE`
No solution or lower bound found

`BOUNDS` $\langle \mathit{LB} \rangle$ `[ :` $\langle$*constraint ID*$\rangle$ `]` $\langle \mathit{UB} \rangle$ `[ :` $\langle$*assignment*$\rangle$ `]`
$\langle LB \rangle$ and $\langle UB \rangle$ are integers or `inf`; optimality if $\langle LB \rangle = \langle UB \rangle$

Conclusions for Optimization Problems

`NONE`
No solution or lower bound found

`BOUNDS` $\langle$*LB*$\rangle$ `[ :` $\langle$*constraint ID*$\rangle$ `]` $\langle$*UB*$\rangle$ `[ :` $\langle$*assignment*$\rangle$ `]`
$\langle LB \rangle$ and $\langle UB \rangle$ are integers or `inf`; optimality if $\langle LB \rangle = \langle UB \rangle$

Lower bound
Constraint $\langle$*constraint ID*$\rangle$, if specified, should imply lower bound
Otherwise, $f \geq \langle LB \rangle$ should be "obvious" to proof checker from current database

Conclusions for Optimization Problems

`NONE`
No solution or lower bound found

`BOUNDS` $\langle$*LB*$\rangle$ `[ :` $\langle$*constraint ID*$\rangle$ `]` $\langle$*UB*$\rangle$ `[ :` $\langle$*assignment*$\rangle$ `]`
$\langle LB \rangle$ and $\langle UB \rangle$ are integers or `inf`; optimality if $\langle LB \rangle = \langle UB \rangle$

Lower bound
Constraint $\langle$*constraint ID*$\rangle$, if specified, should imply lower bound
Otherwise, $f \geq \langle LB \rangle$ should be "obvious" to proof checker from current database

Upper bound
Propagate given assignment w.r.t. database, then check against original formula
If no assignment given, then

- solution with value $\langle UB \rangle$ should have been logged
- no unchecked deletion must have occurred

Projected Model Enumeration and Preserved Variables

Command

`preserve` $\langle \mathit{var1} \rangle$ $\langle \mathit{var2} \rangle$ ... $\langle \mathit{varN} \rangle$

in proof preamble (after loading formula) specifies set V of preserved variables

Projected Model Enumeration and Preserved Variables

Command

`preserve` $\langle var1 \rangle$ $\langle var2 \rangle$ ... $\langle varN \rangle$

in proof preamble (after loading formula) specifies set V of preserved variables

Preserved variables cannot appear in domain of any witness ω for strengthening rules

Projected Model Enumeration and Preserved Variables

Command

`preserve` $\langle \mathit{var1} \rangle$ $\langle \mathit{var2} \rangle$ ... $\langle \mathit{varN} \rangle$

in proof preamble (after loading formula) specifies set V of preserved variables

Preserved variables cannot appear in domain of any witness ω for strengthening rules

Any solution α found is logged with "log solution and exclude" `solx` command

- α checked against current core set $\mathcal{C}$
- Solution-excluding constraint $\bigvee_{x \in V}(x \neq \alpha(x))$ added to core set (forces search for other solutions)

Conclusions for Projected Model Enumeration Problems

`NONE`
No solution or contradiction found

`ENUMERATION PARTIAL :` $\langle N \rangle$
The number of `solx` commands in the proof log is $\langle N \rangle$
No unchecked deletion must have occurred

`ENUMERATION COMPLETE :` $\langle N \rangle$ `[ :` $\langle \mathit{constraint\ ID} \rangle$ `]`
The list of solutions found and enumerated is complete
The number of `solx` commands in the proof log is $\langle N \rangle$
Check that specified constraint is contradictory (technically: negative slack)
If no constraint given, check that database unit propagates to contradiction
No unchecked deletion must have occurred

Problem Reformulation and Output Section

`NONE`
No output

`DERIVABLE`
Any unsatisfiability / lower bound shown for output will be valid also for input

`EQUI-SATISFIABLE`
Input and output are equisatisfiable
true for decision problems with checked deletion

`EQUI-OPTIMAL`
Input and output have same optimal value
(or optimal solution was found and the output is unsatisfiable)

`EQUI-ENUMERABLE`
Input and output have the same number of projected solutions
(and no solutions have been logged)

Objective Update

Objective function update command

`obju` $\langle$*constraint ID 1*$\rangle$ $\langle$*constraint ID 2*$\rangle$ `:` $\langle f_{\mathrm{new}} \rangle$

changes objective function of (potentially reformulated) problem

Specifies two constraints in core set showing $f_{\mathrm{old}} = f_{\mathrm{new}}$

- $f_{\mathrm{old}} \leq f_{\mathrm{new}}$ is implied by $\langle$*constraint ID 1*$\rangle$
- $f_{\mathrm{old}} \geq f_{\mathrm{new}}$ is implied by $\langle$*constraint ID 2*$\rangle$

Using VeriPB for SAT Solving

1. Use dedicated tools for Gaussian elimination [GN21], symmetry breaking [BGMN22], 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 [BGMN22], 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 [BGMN22], 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 be way faster to check

VeriPB Documentation

VeriPB tutorial [BMN22] (video at `https://youtu.be/s_5BIi4I22w`)

And upcoming half-day tutorial at *IJCAI '23*!

Description of VeriPB and CakePB [BMM+23] for SAT 2023 competition (available at `https://satcompetition.github.io/2023/checkers.html`)

Specific details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, BGMN22, GMN22, GMNO22, VDB22, BBN+23]

Lots of concrete example files at `https://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

- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(work on SCIP in [CGS17, EG21])*
- Satisfiability modulo theories (SMT) solving *(work by Bjørner and others)*

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

- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(work on SCIP in [CGS17, EG21])*
- Satisfiability modulo theories (SMT) solving *(work by Bjørner and others)*

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

- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(work on SCIP in [CGS17, EG21])*
- Satisfiability modulo theories (SMT) solving *(work by Bjørner and others)*

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.

[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.

[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.

[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)*, July 2023. To appear.

References II

[BGMN22] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI '22)*, pages 3698–3707, February 2022.

[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.

References III

[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.

[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.

[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.

[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.

References IV

[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.

[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.

References V

[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.

[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.

References VI

[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.

[Goc22] Stephan Gocht. *Certifying Correctness for Combinatorial Algorithms by Using Pseudo-Boolean Reasoning*. PhD thesis, Lund University, Lund, Sweden, 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.

References VII

[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.

[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.

[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.

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

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

References VIII

[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.