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

Jakob Nordström (UCPH & LU)

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

Pragmatics of SAT ’23 3/29
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

Jakob Nordström (UCPH & LU)
A One-Size-Fits-All Proof Logging System?
Pragmatics of SAT ’23
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

Diagram:
- Input
- Solver
- Answer
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 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?
Proof format for certifying solver should be

- very powerful: minimal overhead for sophisticated reasoning

Pragmatics of SAT ’23 5/29
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 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!
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 [CHH17+,], ...
- 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 😊
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 \( \ell_i: x_i \) or \( \overline{x}_i \) (where \( x_i + \overline{x}_i = 1 \))
- variables \( x_i \) take values \( 0 = false \) or \( 1 = true \)
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** \( \ell_i: x_i \) or \( \overline{x_i} \) (where \( x_i + \overline{x_i} = 1 \))
- variables \( x_i \) take values \( 0 = \text{false} \) or \( 1 = \text{true} \)

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

\[ x \lor \overline{y} \lor z \iff x + \overline{y} + z \geq 1 \]

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
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:

\[ r \Rightarrow x_1 + 2\bar{x}_2 + 3x_3 + 4\bar{x}_4 + 5x_5 \geq 7 \]
\[ r \Leftarrow x_1 + 2\bar{x}_2 + 3x_3 + 4\bar{x}_4 + 5x_5 \geq 7 \]
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:

\[
\begin{align*}
    r \Rightarrow & \quad x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 \geq 7 \\
    r \Leftarrow & \quad x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 \geq 7
\end{align*}
\]
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

\[ f \langle M \rangle \]

preserve \( \langle \text{var1} \rangle \langle \text{var2} \rangle \ldots \langle \text{varN} \rangle \)

\( \langle \text{derivation part} \rangle \)

output \( \langle \text{output part} \rangle \)

conclusion \( \langle \text{conclusion part} \rangle \)

end pseudo-Boolean proof
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 $C$
- 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 $\infty$

Derived set $D$
- All constraints derived during search
- Also intermediate constraints used in proof logging

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**

**Literal axioms**

From the input

\[ l_i \geq 0 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input axioms**

**Literal axioms**

**Addition**

From the input

\[ l_i \geq 0 \]

\[ \sum_i a_i l_i \geq A \quad \sum_i b_i l_i \geq B \]

\[ \sum_i (a_i + b_i) l_i \geq A + B \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input axioms

Literal axioms

Addition

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

From the input

\[
\ell_i \geq 0
\]

\[
\sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B
\]

\[
\sum_i (a_i + b_i) \ell_i \geq A + B
\]

\[
\sum_i a_i \ell_i \geq A
\]

\[
\sum_i ca_i \ell_i \geq cA
\]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input axioms**

**Literal axioms**

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

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

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

From the input

\[
\ell_i \geq 0
\]

\[
\sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B
\]

\[
\sum_i (a_i + b_i) \ell_i \geq A + B
\]

\[
\sum_i a_i \ell_i \geq A
\]

\[
\sum_i c a_i \ell_i \geq c A
\]

\[
\sum_i a_i \ell_i \geq A
\]

\[
\sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil
\]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input axioms

Literal axioms

Addition

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

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

Saturation (constraint in normalized form)

From the input

\[
\begin{align*}
\ell_i & \geq 0 \\
\sum_i a_i \ell_i & \geq A \\
\sum_i b_i \ell_i & \geq B \\
\sum_i (a_i + b_i) \ell_i & \geq A + B \\
\sum_i a_i \ell_i & \geq A \\
\sum_i c a_i \ell_i & \geq cA \\
\sum_i \lceil \frac{a_i}{c} \rceil \ell_i & \geq \lceil \frac{A}{c} \rceil \\
\sum_i a_i \ell_i & \geq A \\
\sum_i \min(a_i, A) \cdot \ell_i & \geq A
\end{align*}
\]
Cutting Planes Toy Example

\[ w + 2x + y \geq 2 \]
Cutting Planes Toy Example

\[ \begin{align*}
\text{Mul by 2} & \quad w + 2x + y \geq 2 \\
& \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}
\end{align*} \]
Cutting Planes Toy Example

\[
\begin{align*}
\text{Mul by 2} \quad & \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad w + 2x + 4y + 2z \geq 5
\end{align*}
\]
Cutting Planes Toy Example

\[
\begin{align*}
\text{Mul by 2} & \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \\
\text{Add} & \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}
\end{align*}
\]
Cutting Planes Toy Example

\[
\begin{align*}
\text{Mul by 2} & \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \\
\text{Add} & \quad \frac{z \geq 0}{z \geq 0}
\end{align*}
\]
Cutting Planes Toy Example

\[
\begin{align*}
\text{Mul by 2} & \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \quad \frac{z \geq 0}{2z \geq 0} \\
\text{Add} & \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \quad \frac{z \geq 0}{2z \geq 0} \\
\end{align*}
\]
Cutting Planes Toy Example

\[
\begin{align*}
\text{Mul by 2} & \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \\
\text{Add} & \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \\
\text{Add} & \quad \frac{3w + 6x + 6y + 2z \geq 9}{2z \geq 0} \\
\end{align*}
\]

Such a calculation can be written in a proof line assuming handles \( C_1 = 2x + y + w \geq 2 \) and \( C_2 = 2x + 4y + 2z + w \geq 5 \) using postfix notation something like \( C_1 \times C_2 \times Ax(z) = z \geq 0 \).
Cutting Planes Toy Example

\[ \begin{align*}
\text{Mul by 2} & \quad w + 2x + y \geq 2 \\
\text{Add} & \quad 2w + 4x + 2y \geq 4 \\
\text{Add} & \quad 3w + 6x + 6y + 2z \geq 9 \\
\text{Mul by 2} & \quad 2z \geq 0 \\
\text{Add} & \quad 3w + 6x + 6y + 2 \geq 9
\end{align*} \]
Cutting Planes Toy Example

\[
\begin{align*}
\text{Mul by 2} & \quad w + 2x + y & \geq 2 \\
\text{Add} & \quad 2w + 4x + 2y & \geq 4 \\
\text{Add} & \quad 3w + 6x + 6y + 2z & \geq 9 \\
\text{Mul by 2} & \quad 2z & \geq 0 \\
\text{Add} & \quad 3w + 6x + 6y & \geq 7 \\
\end{align*}
\]
Cutting Planes Toy Example

\[
\begin{align*}
\text{Mul by 2} & \quad w + 2x + y \geq 2 \\
\text{Add} & \quad 2w + 4x + 2y \geq 4 \\
\text{Add} & \quad 3w + 6x + 6y + 2z \geq 9 \\
\text{Div by 3} & \quad w + 2x + 2y \geq 2\frac{1}{3}
\end{align*}
\]

\[
\begin{align*}
\text{Add} & \quad w + 2x + 4y + 2z \geq 5 \\
\text{Div by 3} & \quad 3w + 6x + 6y \geq 7 \\
\text{Mul by 2} & \quad 2z \geq 0
\end{align*}
\]
Cutting Planes Toy Example

\[ w + 2x + y \geq 2 \]
\[ 2w + 4x + 2y \geq 4 \]
\[ 3w + 6x + 6y + 2z \geq 9 \]
\[ w + 2x + 2y \geq 3 \]

\[ w + 2x + 4y + 2z \geq 5 \]
\[ 2w \geq 0 \]
\[ 2z \geq 0 \]
\[ \frac{3w + 6x + 6y}{3} \geq 7 \]

Jakob Nordström (UCPH & LU)
### Cutting Planes Toy Example

\[
\begin{align*}
&\quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \\
&\quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \\
&\quad \frac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 3}
\end{align*}
\]

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

\[
\begin{align*}
C_1 & \triangleq 2x + y + w \geq 2 \\
C_2 & \triangleq 2x + 4y + 2z + w \geq 5 \\
Ax(\overline{z}) & \triangleq \overline{z} \geq 0
\end{align*}
\]
Cutting Planes Toy Example

\begin{align*}
\text{Mul by 2} & \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \\
\text{Add} & \quad \frac{w + 2x + 4y \geq 5}{2w + 4x + 2y \geq 4} \\
\text{Add} & \quad \frac{3w + 6x + 6y \geq 9}{w + 2x + 2y \geq 3} \\
\text{Div by 3} & \quad \frac{3w + 6x + 6y \geq 9}{w + 2x + 2y \geq 3}
\end{align*}

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

\[ C_1 \triangleq 2x + y + w \geq 2 \]
\[ C_2 \triangleq 2x + 4y + 2z + w \geq 5 \]
\[ Ax(\overline{z}) \triangleq \overline{z} \geq 0 \]

using postfix notation something like

\[ C_1 \text{ Mul} \quad C_2 \text{ Add} \quad Ax(\overline{z}) \text{ 2 Mul  Add 3 Div} \]
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** $\omega$: substitution mapping variables to truth values or literals

**Redundance-based strengthening** (witness $\omega$ show how to “patch assignment”)

Derive constraint $C$ from $C \cup D$ if exists witness $\omega$ such that

$$C \cup D \cup \{\neg C\} \vdash (C \cup D \cup \{C\})|_\omega \cup \{f|_\omega \leq f\} \cup O(\vec{z}|_\omega, \vec{z})$$
**Strengthening Rules**

**Witness** $\omega$: substitution mapping variables to truth values or literals

**Redundance-based strengthening** (witness $\omega$ show how to “patch assignment”)

Derive constraint $C$ from $C \cup D$ if exists witness $\omega$ such that

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

**Dominance-based strengthening** (witness $\omega$ “drives down potential”)

Derive constraint $C$ from $C \cup D$ if exists witness $\omega$ such that

$$C \cup D \cup \{\neg C\} \vdash C\upharpoonright \omega \cup \{f\upharpoonright \omega \leq f\} \cup O(\vec{z}\upharpoonright \omega, \vec{z}) \cup \neg O(\vec{z}, \vec{z}\upharpoonright \omega)$$
**Strengthening Rules**

**Witness** $\omega$: substitution mapping variables to truth values or literals

**Redundance-based strengthening** *(witness $\omega$ show how to “patch assignment”)*

Derive constraint $C$ from $C \cup D$ if exists witness $\omega$ such that

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

**Dominance-based strengthening** *(witness $\omega$ “drives down potential”)*

Derive constraint $C$ from $C \cup D$ if exists witness $\omega$ such that

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

- Witness $\omega$ 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 $C$ and derived set $D$
(For SAT solvers, support generic delete command in augmented format that translates to right type of deletion behind the scenes)
Pseudo-Boolean Proof Logging Basics
Pseudo-Boolean Proof Logging for Different Purposes
Pseudo-Boolean Proof Logging Outlook

Proof Logging Goals
VERIPB Proof Fundamentals
Strengthening Rules and Deletion

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 $C$ and derived set $D$
(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 $D$
2. OK from core set $C$ only if $C'$ can be rederived from $C \setminus \{C\}$ with redundancy 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 $C$ and derived set $D$
(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 $D$
2. OK from core set $C$ only if $C'$ can be rederived from $C \setminus \{C\}$ with redundancy rule
   (otherwise unchecked deletion — special conditions apply)
Conclusions for Decision Problems

NONE
Status is undetermined

SAT \[ : \langle \text{assignment} \rangle \]
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 \[ : \langle \text{constraint ID} \rangle \]
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 $\alpha$ found is logged with the `soli` command “log solution and improve”:

- $\alpha$ checked against current core set $C$
- Objective-improving constraint $\sum_i w_i l_i \leq -1 + \sum_i w_i \cdot \alpha(l_i)$ added to core set (forces search for better solutions)
Any solution $\alpha$ found is logged with `soli “log solution and improve” command

- $\alpha$ checked against current core set $C$
- 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

- $\alpha$ 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 $\langle LB \rangle$ [ : $\langle \text{constraint ID} \rangle$ ] $\langle UB \rangle$ [ : $\langle \text{assignment} \rangle$ ]
$\langle LB \rangle$ and $\langle UB \rangle$ are integers or $\text{inf}$; 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

\texttt{preserve \langle var1 \rangle \langle var2 \rangle \ldots \langle varN \rangle}

in proof preamble (after loading formula) specifies set $V$ of \texttt{preserved variables}
Projected Model Enumeration and Preserved Variables

Command

\texttt{preserve \textit{var1} \textit{var2} \ldots \textit{varN}}

in proof preamble (after loading formula) specifies set $V$ of \textit{preserved variables}

Preserved variables cannot appear in domain of any witness $\omega$ for strengthening rules
Projected Model Enumeration and Preserved Variables

Command

\texttt{preserve } ⟨\texttt{var1}⟩ ⟨\texttt{var2}⟩ \ldots ⟨\texttt{varN}⟩

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” \texttt{solx} command

\begin{itemize}
  \item \( α \) checked against current core set \( C \)
  \item Solution-excluding constraint \( \bigvee_{x \in V}(x \neq α(x)) \) added to core set (forces search for other solutions)
\end{itemize}
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 \begin{array}{c}
\text{constraint ID} \\
\end{array}
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 function update command

\texttt{obju \{constraint ID 1\} \{constraint ID 2\} : \{f_{\text{new}}\}}

changes objective function of (potentially reformulated) problem

Specifies two constraints in core set showing \( f_{\text{old}} = f_{\text{new}} \):

- \( f_{\text{old}} \leq f_{\text{new}} \) is implied by \( \{\text{constraint ID 1}\} \)
- \( f_{\text{old}} \geq f_{\text{new}} \) is implied by \( \{\text{constraint ID 2}\} \)
Using $\text{VeRIPB}$ 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 $\text{VeRIPB}$ 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

<table>
<thead>
<tr>
<th>DRAT</th>
<th>VeriPB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x1</td>
</tr>
<tr>
<td>-2</td>
<td>~x2</td>
</tr>
<tr>
<td>1 -2 3 0</td>
<td>1 x1 1 ~x2 1 x3 &gt;= 1 ;</td>
</tr>
<tr>
<td>1 -2 3 0 is RUP</td>
<td>rup 1 x1 1 ~x2 1 x3 &gt;= 1 ;</td>
</tr>
<tr>
<td>1 -2 3 0 is RAT</td>
<td>red 1 x1 1 ~x2 1 x3 &gt;= 1 ; x1 -&gt; 1</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>DRAT</th>
<th>VeriPB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x1</td>
</tr>
<tr>
<td>-2</td>
<td>~x2</td>
</tr>
<tr>
<td>1 -2 3 0</td>
<td>1 x1 1 ~x2 1 x3 &gt;= 1 ;</td>
</tr>
<tr>
<td>1 -2 3 0 is RUP</td>
<td>rup 1 x1 1 ~x2 1 x3 &gt;= 1 ;</td>
</tr>
<tr>
<td>1 -2 3 0 is RAT</td>
<td>red 1 x1 1 ~x2 1 x3 &gt;= 1 ; x1 -&gt; 1</td>
</tr>
</tbody>
</table>

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])

And more...
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 the 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 II


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Conference Details</th>
</tr>
</thead>
</table>


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Conference/Journal</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>[RvBW06]</td>
<td>Francesca Rossi, Peter van Beek, and Toby Walsh</td>
<td>Handbook of Constraint Programming</td>
<td>Handbook of Constraint Programming, volume 2 of Foundations of Artificial Intelligence</td>
<td>2006</td>
</tr>
</tbody>
</table>