# Combinatorial Solving with Provably Correct Results

Jakob Nordström

University of Copenhagen and Lund University

Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
September 16, 2025

# Based on Joint Work With...

- Markus Anders
- Jeremias Berg
- Bart Bogaerts
- Benjamin Bogø
- Wolf De Wulf
- Emir Demirović
- Simon Dold
- Jan Elffers
- Ambros Gleixner
- Stephan Gocht
- Arthur Gontier
- Malte Helmert
- Alexander Hoen
- Hannes Ihalainen
- Matti Järvisalo
- Wietze Koops
- Daniel Le Berre
- Ruben Martins
- Ross McBride
- Ciaran McCreesh
- Matthew McIlree
- Magnus O. Myreen
- Andy Oertel
- Tobias Paxian
- Patrick Prosser
- Adrián Rebola-Pardo
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- Yong Kiam Tan
- James Trimble
- Dieter Vandesande
- Marc Vinyals

# Based on Joint Work With...

- Markus Anders
- Jeremias Berg
- **Bart Bogaerts**
- Benjamin Bogø
- Wolf De Wulf
- Emir Demirović
- Simon Dold
- Jan Elffers
- Ambros Gleixner
- **Stephan Gocht**
- Arthur Gontier
- Malte Helmert
- Alexander Hoen
- Hannes Ihalainen
- Matti Järvisalo
- Wietze Koops
- Daniel Le Berre
- Ruben Martins
- Ross McBride
- **Ciaran McCreesh**
- Matthew McIlree
- Magnus O. Myreen
- Andy Oertel
- Tobias Paxian
- Patrick Prosser
- Adrián Rebola-Pardo
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- Yong Kiam Tan
- James Trimble
- Dieter Vandesande
- Marc Vinyals

# Based on Joint Work With...

- Markus Anders
- Jeremias Berg
- **Bart Bogaerts**
- **Benjamin Bogø**
- Wolf De Wulf
- Emir Demirović
- Simon Dold
- Jan Elffers
- Ambros Gleixner
- **Stephan Gocht**
- Arthur Gontier
- Malte Helmert
- Alexander Hoen
- Hannes Ihalainen
- Matti Järvisalo
- **Wietze Koops**
- Daniel Le Berre
- Ruben Martins
- Ross McBride
- **Ciaran McCreesh**
- Matthew McIlree
- Magnus O. Myreen
- **Andy Oertel**
- Tobias Paxian
- Patrick Prosser
- Adrián Rebola-Pardo
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- Yong Kiam Tan
- James Trimble
- Dieter Vandesande
- Marc Vinyals

# Based on Joint Work With...

- Markus Anders
- Jeremias Berg
- **Bart Bogaerts**
- **Benjamin Bogø**
- Wolf De Wulf
- Emir Demirović
- Simon Dold
- Jan Elffers
- Ambros Gleixner
- **Stephan Gocht**
- Arthur Gontier
- Malte Helmert
- Alexander Hoen
- Hannes Ihalainen
- Matti Järvisalo
- **Wietze Koops**
- Daniel Le Berre
- Ruben Martins
- Ross McBride
- **Ciaran McCreesh**
- Matthew McIlree
- **Magnus O. Myreen**
- **Andy Oertel**
- Tobias Paxian
- Patrick Prosser
- Adrián Rebola-Pardo
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- **Yong Kiam Tan**
- James Trimble
- Dieter Vandesande
- Marc Vinyals

# The Success Story of Combinatorial Solving and Optimization

- Rich field of mathematics and computer science
- Impact in other areas of science and also industry, e.g.:
  - airline scheduling
  - hardware verification
  - donor-recipients matching for kidney transplants [MO12, BvdKM$^+$21]
- Discrete problems — computationally very challenging (NP-complete or worse)
- Lots of effort last couple of decades spent on developing sophisticated so-called combinatorial solvers that often work surprisingly well in practice for, e.g.,
  - Boolean satisfiability (SAT) solving [BHvMW21]
  - Constraint programming [RvBW06]
  - Mixed integer linear programming [AW13, BR07]
  - Satisfiability modulo theories (SMT) solving [BHvMW21]

# And the Dirty Little Secret...

- Solvers very fast, but sometimes wrong (even best commercial ones) [BLB10, CKSW13, AGJ$^+$18, GSD19, BMN22, GCS23]
- Even worse: No way of knowing for sure when errors happen
- Solvers can propose infeasible "solutions" (but erroneous claims can in principle be checked)
- More challenging: How to achieve reliable claims of infeasibility?
- Or of optimality?
- Even off-by-one mistakes can snowball into large errors if solver used as subroutine

# What Can Be Done About Solver Bugs?

- **Software testing**
  Very useful, but bugs slip through even with careful domain-specific testing
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence

# What Can Be Done About Solver Bugs?

- **Software testing**
  Very useful, but bugs slip through even with careful domain-specific testing
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence

- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to level of complexity in modern solvers
  (Despite valiant efforts in, e.g., [Fle20])

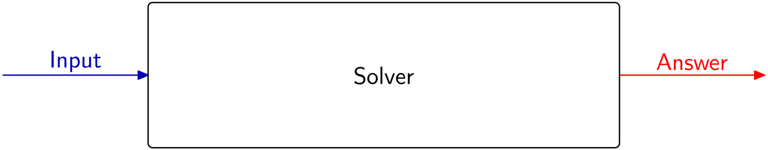
# What Can Be Done About Solver Bugs?

- **Software testing**
  Very useful, but bugs slip through even with careful domain-specific testing
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence

- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to level of complexity in modern solvers
  (Despite valiant efforts in, e.g., [Fle20])

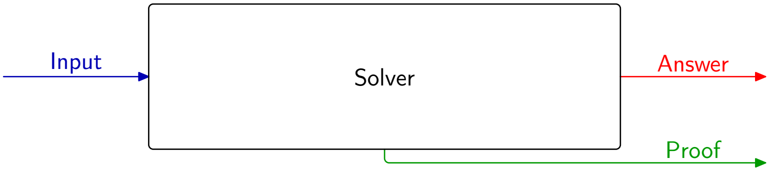
- **Proof logging**
  Make solver certifying [ABM+11, MMNS11] by adding code so that it outputs
  1. not only answer but also
  2. simple, machine-verifiable proof that answer is correct

# Proof Logging with Certifying Solvers: Workflow



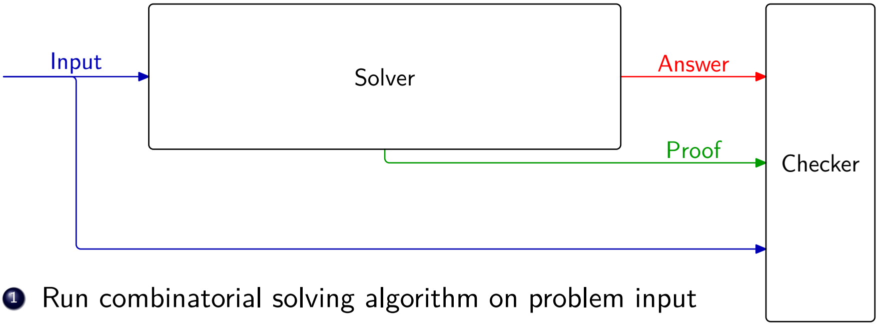

1. Run combinatorial solving algorithm on problem input

# Proof Logging with Certifying Solvers: Workflow



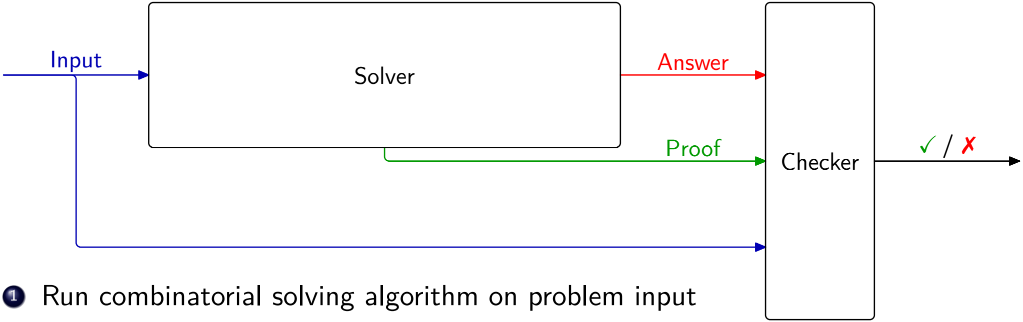

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

# 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

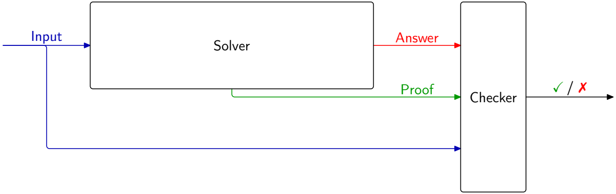
# Proof Logging with Certifying Solvers: Workflow



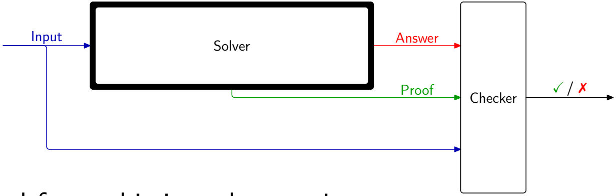

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

# Proof Logging Desiderata

Proof format for certifying solver should be



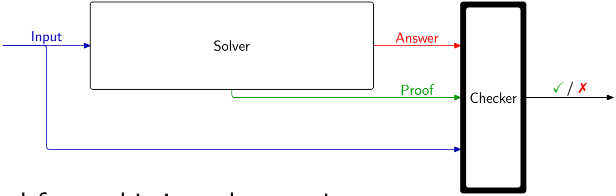
# Proof Logging Desiderata




Proof format for certifying solver should be

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

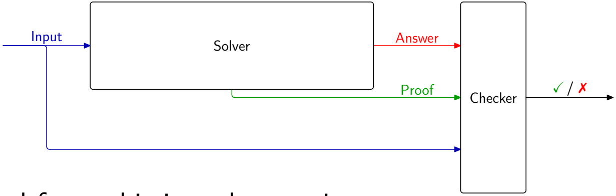
# Proof Logging Desiderata




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

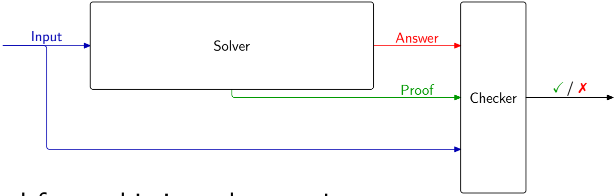



Proof format for certifying solver should be

- **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?

# Some Previous Proof Logging Work

**Boolean satisfiability (SAT) solving**

- Well established since over decade with several proof formats such as $\mathrm{DRAT}$ [HHW13a, HHW13b, WHH14], $\mathrm{GRIT}$ [CMS17], $\mathrm{LRAT}$ [CHH⁺17], ...
- But no efficient support for most advanced techniques such as
  - Gaussian elimination
  - symmetry breaking

# Some Previous Proof Logging Work

**Boolean satisfiability (SAT) solving**

- Well established since over decade with several proof formats such as DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], ...
- But no efficient support for most advanced techniques such as
  - Gaussian elimination
  - symmetry breaking

**Constraint programming**

- Either have to trust that propagations done correctly [DFS12, OSC09, VS10]
- Or suffer from exponential slow-down to generate verifiable proofs [GCS23]

# Some Previous Proof Logging Work

### Boolean satisfiability (SAT) solving

- Well established since over decade with several proof formats such as DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH$^+$17], ...
- But no efficient support for most advanced techniques such as
  - Gaussian elimination
  - symmetry breaking

### Constraint programming

- Either have to trust that propagations done correctly [DFS12, OSC09, VS10]
- Or suffer from exponential slow-down to generate verifiable proofs [GCS23]

### Mixed integer linear programming

- Work on proof format VIPR [CGS17, EG23]
- But only for exact solving and without support for advanced techniques

# Message of This Talk

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

# Message of This Talk

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

- Build on successes in proof logging for SAT solving
- 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`)

# Message of This Talk

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

- Build on successes in proof logging for SAT solving
- 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 ☺

# Message of This Talk

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

- Build on successes in proof logging for SAT solving
- 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. Describe foundations of proof logging method

# Message of This Talk

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

- Build on successes in proof logging for SAT solving
- 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. Describe foundations of proof logging method
3. Discuss future challenges and directions

# 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 [GMM+20, KM21, BBN+23, EG23, KLM+25]
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 print statements (plus some book-keeping) to solver code

# Design Principles for Proof Logging

## Proof logging implementation

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

## Performance goals

- Proof logging overhead small constant fraction of running time ($\lessapprox 10\%$)
- 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 print statements (plus some book-keeping) to solver code

## Performance goals

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

## Proof system

- Keep language simple — no XOR constraints, CP propagators, symmetries, ...
- But reason efficiently about such notions using power of proof system
- Combine proof logging with formally verified proof checker

# Proof Language: 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 $\ell_i$: $x_i$ 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. Disjunctive 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**

- Boolean satisfiability (SAT) solving
- (linear) pseudo-Boolean solving
- subgraph 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**

- Boolean satisfiability (SAT) solving
- (linear) pseudo-Boolean solving
- subgraph 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**

- Boolean satisfiability (SAT) solving
- (linear) pseudo-Boolean solving
- subgraph 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**

- Boolean satisfiability (SAT) solving
- (linear) pseudo-Boolean solving
- subgraph 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 using big-M 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 using big-M 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]

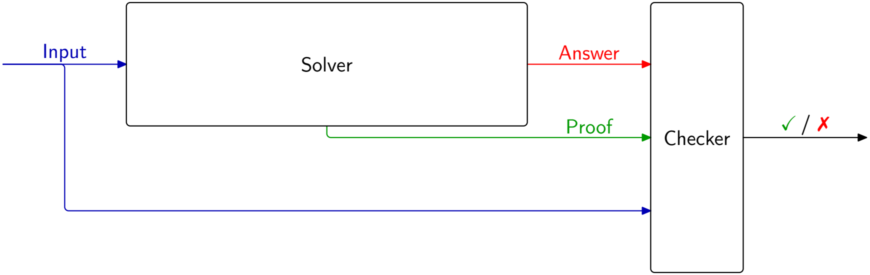
**Goldilocks compromise** between expressivity and simplicity:

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

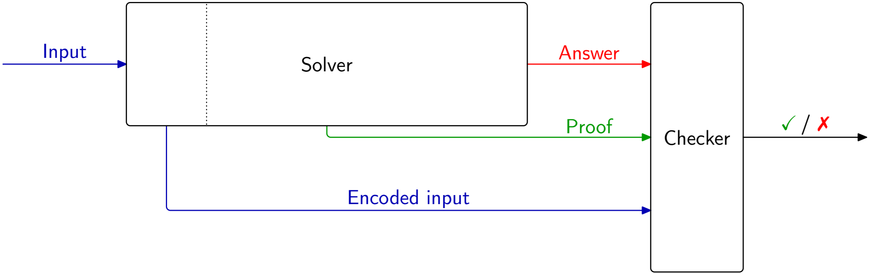
$$r \Rightarrow x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7 \qquad 7\overline{r} + x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

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

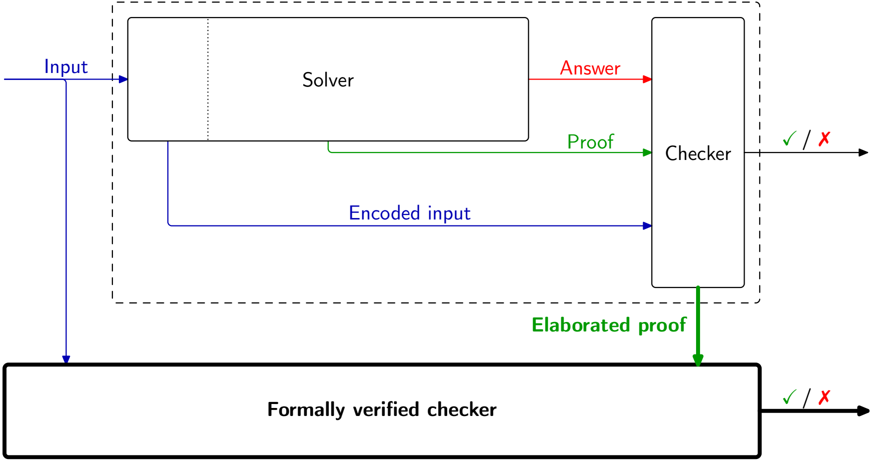
# Proof Logging with Formally Verified Checking: Full Workflow



# Proof Logging with Formally Verified Checking: Full Workflow



# Proof Logging with Formally Verified Checking: Full Workflow



# VeriPB Proof 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 (or upper and lower bounds)
   - other types of conclusions

# VeriPB Proof Configuration (Slightly Simplified)

**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 [but not used by solver]
- Any satisfying assignment to $\mathcal{C}$ can be extended to $\mathcal{D}$

# VeriPB Proof Configuration (Slightly Simplified)

**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 [but not used by solver]
- Any satisfying assignment to $\mathcal{C}$ can be extended to $\mathcal{D}$

**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 $\infty$

# 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 $c \in \mathbb{N}^+$

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

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

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

# 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 $c \in \mathbb{N}^+$
$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil}$$

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

$$w + 2x + 4y + 2z \geq 5$$

# Cutting Planes Toy Example

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

# Cutting Planes Toy Example

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

# 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

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

# Cutting Planes Toy Example

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

# Cutting Planes Toy Example

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

# Cutting Planes Toy Example

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

# Cutting Planes Toy Example

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

# 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 referring to constraints by labels and to literal axioms by the literal involved as

$$
\begin{aligned}
\texttt{@C1} &\doteq 2x + y + w \geq 2 \\
\texttt{@C2} &\doteq 2x + 4y + 2z + w \geq 5 \\
\texttt{\sim z} &\doteq \overline{z} \geq 0
\end{aligned}
$$

# 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 referring to constraints by labels and to literal axioms by the literal involved as

$$
\begin{aligned}
\texttt{@C1} &\doteq 2x+y+w \geq 2 \\
\texttt{@C2} &\doteq 2x+4y+2z+w \geq 5 \\
\texttt{\~{}z} &\doteq \overline{z} \geq 0
\end{aligned}
$$

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

```
pol  @C1  2  *  @C2  +  ~z  2  *  +  3  d  ;
```

# Deriving Non-implied Constraints by Redundance-Based Strengthening

$C$ is said to be "redundant" with respect to $F$ if $F$ and $F \cup \{C\}$ are equisatisfiable

*[apologies for the terminology — this is inherited from SAT proof logging]*

# Deriving Non-implied Constraints by Redundance-Based Strengthening

$C$ is said to be "redundant" with respect to $F$ if $F$ and $F \cup \{C\}$ are equisatisfiable

*[apologies for the terminology — this is inherited from SAT proof logging]*

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

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

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

# Deriving Non-implied Constraints by Redundance-Based Strengthening

$C$ is said to be "redundant" with respect to $F$ if $F$ and $F \cup \{C\}$ are equisatisfiable

*[apologies for the terminology — this is inherited from SAT proof logging]*

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

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

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

- Proof sketch for interesting direction: If $\alpha$ satisfies $F$ but falsifies $C$, then $\alpha$ satisfies $(F \cup \{C\})\restriction_{\omega}$, i.e., $\alpha \circ \omega$ satisfies $F \cup \{C\}$

# Deriving Non-implied Constraints by Redundance-Based Strengthening

$C$ is said to be "redundant" with respect to $F$ if $F$ and $F \cup \{C\}$ are equisatisfiable

*[apologies for the terminology — this is inherited from SAT proof logging]*

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

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

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

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

# Example: Deriving $r \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$2\overline{r} + x + y \geq 2 \qquad\qquad r + \overline{x} + \overline{y} \geq 1$$

using condition $F \cup \{\neg C\} \models (F \cup \{C\})\restriction_\omega$

# Example: Deriving $r \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$2\overline{r} + x + y \geq 2 \qquad\qquad r + \overline{x} + \overline{y} \geq 1$$

using condition $F \cup \{\neg C\} \models (F \cup \{C\})\restriction_\omega$

1. $F \cup \{\neg(2\overline{r} + x + y \geq 2)\} \models (F \cup \{2\overline{r} + x + y \geq 2\})\restriction_\omega$

# Example: Deriving $r \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$2\overline{r} + x + y \geq 2 \qquad\qquad r + \overline{x} + \overline{y} \geq 1$$

using condition $F \cup \{\neg C\} \models (F \cup \{C\})\restriction_\omega$

1. $F \cup \{\neg(2\overline{r} + x + y \geq 2)\} \models (F \cup \{2\overline{r} + x + y \geq 2\})\restriction_\omega$
   Choose $\omega = \{r \mapsto 0\}$ — $F$ untouched; new constraint satisfied

# Example: Deriving $r \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$2\overline{r} + x + y \geq 2 \qquad\qquad r + \overline{x} + \overline{y} \geq 1$$

using condition $F \cup \{\neg C\} \models (F \cup \{C\})\restriction_\omega$

1. $F \cup \{\neg(2\overline{r} + x + y \geq 2)\} \models (F \cup \{2\overline{r} + x + y \geq 2\})\restriction_\omega$
   Choose $\omega = \{r \mapsto 0\}$ — $F$ untouched; new constraint satisfied
2. $F \cup \{2\overline{r} + x + y \geq 2, \neg(r + \overline{x} + \overline{y} \geq 1)\} \models$
   $(F \cup \{2\overline{r} + x + y \geq 2, r + \overline{x} + \overline{y} \geq 1\})\restriction_\omega$

# Example: Deriving $r \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$2\overline{r} + x + y \geq 2 \qquad\qquad r + \overline{x} + \overline{y} \geq 1$$

using condition $F \cup \{\neg C\} \models (F \cup \{C\})\restriction_\omega$

1. $F \cup \{\neg(2\overline{r} + x + y \geq 2)\} \models (F \cup \{2\overline{r} + x + y \geq 2\})\restriction_\omega$
   Choose $\omega = \{r \mapsto 0\}$ — $F$ untouched; new constraint satisfied
2. $F \cup \{2\overline{r} + x + y \geq 2, \neg(r + \overline{x} + \overline{y} \geq 1)\} \models$
   $(F \cup \{2\overline{r} + x + y \geq 2, r + \overline{x} + \overline{y} \geq 1\})\restriction_\omega$
   Choose $\omega = \{r \mapsto 1\}$ — $F$ untouched; new constraint satisfied
   Premise $\neg(r + \overline{x} + \overline{y} \geq 1)$ forces $x \mapsto 1$ and $y \mapsto 1$, hence $(2\overline{r} + x + y \geq 2)\restriction_\omega$ is satisfied even though $r \mapsto 1$

# Example: Deriving $r \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$2\overline{r} + x + y \geq 2 \qquad\qquad r + \overline{x} + \overline{y} \geq 1$$

using condition $F \cup \{\neg C\} \models (F \cup \{C\})\restriction_\omega$

1. $F \cup \{\neg(2\overline{r} + x + y \geq 2)\} \models (F \cup \{2\overline{r} + x + y \geq 2\})\restriction_\omega$
   Choose $\omega = \{r \mapsto 0\}$ — $F$ untouched; new constraint satisfied
2. $F \cup \{2\overline{r} + x + y \geq 2, \neg(r + \overline{x} + \overline{y} \geq 1)\} \models$
   $(F \cup \{2\overline{r} + x + y \geq 2, r + \overline{x} + \overline{y} \geq 1\})\restriction_\omega$
   Choose $\omega = \{r \mapsto 1\}$ — $F$ untouched; new constraint satisfied
   Premise $\neg(r + \overline{x} + \overline{y} \geq 1)$ forces $x \mapsto 1$ and $y \mapsto 1$, hence $(2\overline{r} + x + y \geq 2)\restriction_\omega$ is satisfied even though $r \mapsto 1$

```
red 2 ~r 1 x 1 y  >= 2 : r -> 0 ;
red 1 r 1 ~x 1 ~y >= 1 : r -> 1 ;
```

# 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 $\omega$ 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 $\omega$ 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 $\omega$ 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 $\omega$ 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 $\omega$ strictly improves solution

### Dominance-based strengthening [BGMN23]

Add constraint $C$ to derived set $\mathcal{D}$ if exists witness substitution $\omega$ 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 $\omega$ 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 $\omega$ strictly improves solution

## Dominance-based strengthening [BGMN23]

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

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

- Applying $\omega$ 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 $\omega$ such that

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

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

# Soundness of Dominance Rule

## Dominance-based strengthening

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

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

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

1. Suppose $\alpha$ 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 $\omega$ such that

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

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

1. Suppose $\alpha$ 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 $\omega$ such that

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

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

1. Suppose $\alpha$ 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 $\omega$ such that

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

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

1. Suppose $\alpha$ 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 $\omega$ such that

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

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

1. Suppose $\alpha$ 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 $\omega$ such that

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

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

1. Suppose $\alpha$ 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 $\omega$ such that

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

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

1. Suppose $\alpha$ 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 $\omega$ such that

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

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

1. Suppose $\alpha$ 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 $\alpha'$ satisfying $\mathcal{C} \cup \{C\}$

# Strengthening Rules: Proof Format

red ⟨Constraint $C$⟩ : ⟨*var1*⟩ -> ⟨*val1*⟩ ... ⟨*varN*⟩ -> ⟨*valN*⟩ : subproof
    *subproofs for proof goals*
qed;

dom ⟨Constraint $C$⟩ : ⟨*var1*⟩ -> ⟨*val1*⟩ ... ⟨*varN*⟩ -> ⟨*valN*⟩ : subproof
    *subproofs for proof goals*
qed;

# Strengthening Rules: Proof Format

red $\langle$Constraint $C\rangle$ : $\langle$*var1*$\rangle$ -> $\langle$*val1*$\rangle$ ... $\langle$*varN*$\rangle$ -> $\langle$*valN*$\rangle$ : subproof
*subproofs for proof goals*
qed;

dom $\langle$Constraint $C\rangle$ : $\langle$*var1*$\rangle$ -> $\langle$*val1*$\rangle$ ... $\langle$*varN*$\rangle$ -> $\langle$*valN*$\rangle$ : subproof
*subproofs for proof goals*
qed;

- Witness $\omega$ should be explicitly specified in proof log
- Subproofs of proof goals should also be explicit
- But can be skipped for proof goals “obvious” to proof checker

# Successful Applications of VeriPB Proof Logging

Surprisingly, pseudo-Boolean reasoning with strengthening rules sufficient to efficiently certify wide range of combinatorial solving techniques:

# Successful Applications of VeriPB Proof Logging

Surprisingly, pseudo-Boolean reasoning with strengthening rules sufficient to efficiently certify wide range of combinatorial solving techniques:

1. Boolean satisfiability (SAT) solving including advanced techniques such as
   - Gaussian elimination [GN21]
   - symmetry breaking [BGMN23]
2. SAT-based optimization (MaxSAT) [VDB22, BBN+23, BBN+24, IOT+24]
3. (Linear) Pseudo-Boolean solving [GMNO22, KLM+25]
4. Subgraph solving (max clique, subgraph isomorphism, max common connected subgraph) [GMN20, GMM+20, GMM+24]
5. Dynamic programming and decision diagrams [DMM+24]
6. Presolving in $0$–$1$ integer linear programming [HOGN24]
7. Constraint programming [EGMN20, GMN22, MM23, MMN24, MM25]
8. Automated planning [DHN+25]

# Three Pseudo-Boolean Proof Logging Vignettes

1. Symmetry breaking [BGMN23]
2. Graph solving (subgraph isomorphism) [GMN20, GMM$^+$20, GMM$^+$24]
3. Constraint programming [EGMN20, GMN22, MM23, MMN24, MM25]

# 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. Use dominance to derive (for proof log only) pseudo-Boolean 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. Use dominance to derive (for proof log only) pseudo-Boolean 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. Use dominance to derive (for proof log only) pseudo-Boolean 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. Use dominance to derive (for proof log only) pseudo-Boolean 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:

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

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., one-to-one mappings $\varphi$ such that 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
2. Local derivations can be chained into global correctness proof
3. Proof checkable by stand-alone verifier that knows nothing about graphs
4. With end-to-end fully formally verified result [GMM+24]

# 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)

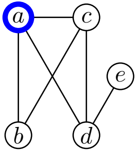
**Pseudo-Boolean encoding**

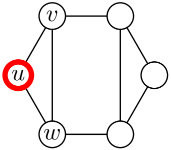
$$\sum_{v \in V(\mathcal{T})} x_{a,v} = 1 \qquad \text{[every } a \text{ maps somewhere]}$$

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

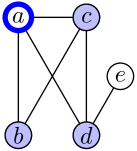
$$\overline{x}_{a,u} + \sum_{v \in N(u)} x_{b,v} \geq 1 \qquad \text{[edge } (a,b) \text{ maps to edge } (u,v)\text{]}$$

# Pseudo-Boolean Proof Logging Example: Degree Preprocessing

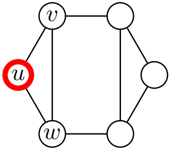


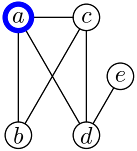
# Pseudo-Boolean Proof Logging Example: Degree Preprocessing



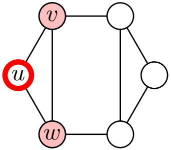

$$\overline{x}_{a,u} + x_{b,v} + x_{b,w} \geq 1$$
$$\overline{x}_{a,u} + x_{c,v} + x_{c,w} \geq 1$$
$$\overline{x}_{a,u} + x_{d,v} + x_{d,w} \geq 1$$



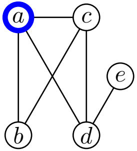
# Pseudo-Boolean Proof Logging Example: Degree Preprocessing



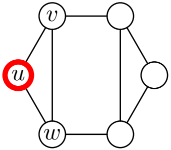

$$\begin{aligned}
\overline{x}_{a,u} + x_{b,v} + x_{b,w} &\geq 1 \\
\overline{x}_{a,u} + x_{c,v} + x_{c,w} &\geq 1 \\
\overline{x}_{a,u} + x_{d,v} + x_{d,w} &\geq 1 \\
\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} &\geq 4 \\
\overline{x}_{a,w}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



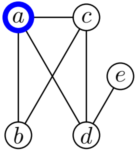
# Pseudo-Boolean Proof Logging Example: Degree Preprocessing



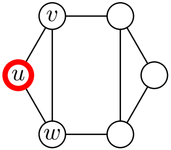

$$\begin{aligned}
\overline{x}_{a,u} + x_{b,v} + x_{b,w} &\geq 1 \\
\overline{x}_{a,u} + x_{c,v} + x_{c,w} &\geq 1 \\
\overline{x}_{a,u} + x_{d,v} + x_{d,w} &\geq 1 \\
\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} &\geq 4 \\
\overline{x}_{a,w}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



# Pseudo-Boolean Proof Logging Example: Degree Preprocessing

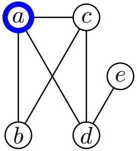



$$
\begin{aligned}
\overline{x}_{a,u} + x_{b,v} + x_{b,w} &\geq 1 \\
\overline{x}_{a,u} + x_{c,v} + x_{c,w} &\geq 1 \\
\overline{x}_{a,u} + x_{d,v} + x_{d,w} &\geq 1 \\
\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} &\geq 4 \\
\overline{x}_{a,w}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

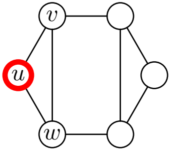



Sum up all constraints & divide by $3$ to obtain

# Pseudo-Boolean Proof Logging Example: Degree Preprocessing



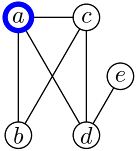

$$\begin{aligned}
\overline{x}_{a,u} + x_{b,v} + x_{b,w} &\geq 1 \\
\overline{x}_{a,u} + x_{c,v} + x_{c,w} &\geq 1 \\
\overline{x}_{a,u} + x_{d,v} + x_{d,w} &\geq 1 \\
\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} &\geq 4 \\
\overline{x}_{a,w}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

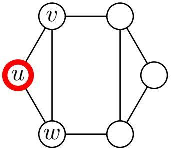



Sum up all constraints & divide by $3$ to obtain

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

# Pseudo-Boolean Proof Logging Example: Degree Preprocessing



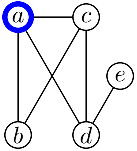




$$
\begin{aligned}
\overline{x}_{a,u} + x_{b,v} + x_{b,w} &\geq 1 \\
\overline{x}_{a,u} + x_{c,v} + x_{c,w} &\geq 1 \\
\overline{x}_{a,u} + x_{d,v} + x_{d,w} &\geq 1 \\
\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} &\geq 4 \\
\overline{x}_{a,w}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

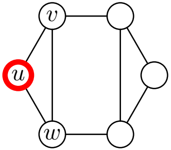
Sum up all constraints & divide by $3$ to obtain

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

# Pseudo-Boolean Proof Logging Example: Degree Preprocessing




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




Sum up all constraints & divide by 3 to obtain

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

# Constraint Programming: Integer Variables (1/2)

How to deal with integer variables in constraint programming?

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

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

# Constraint Programming: Integer Variables (1/2)

How to deal with integer variables in constraint programming?

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. . .

# Constraint Programming: Integer Variables (1/2)

How to deal with integer variables in constraint programming?

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

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

This doesn't work for large domains...

We can instead use a binary encoding:

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

Bad properties for solver propagation, but that isn't a problem for proof logging

# Constraint Programming: Integer Variables (2/2)

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

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

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

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

# Constraint Programming: Integer Variables (2/2)

We can mix binary and order encodings! Define big-M 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

# Constraint Programming: Integer Variables (2/2)

We can mix binary and order encodings! Define big-M 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

# Constraint Programming: Table Constraints

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

Variable assignments must match some row in table

# Constraint Programming: 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

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

using tuple selector variables

$$t_1 + t_2 + t_3 = 1$$

# A Constraint Programming Solver with Pseudo-Boolean Proof Logging

Proof-of-concept CP solver at `github.com/ciaranm/glasgow-constraint-solver` supports proof logging for global constraints:

- 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
- and more...

Details in [EGMN20, GMN22, MM23, MMN24, MM25]

# Future Research Directions

**Performance and reliability of pseudo-Boolean proof logging and checking**

- Trim proof while verifying (as in DRAT-TRIM [HHW13a])
- Compress proof file using binary format
- Formally verifed end-to-end checking (as in [GMM+24, IOT+24, KLM+25])
- Faster proof logging and checking!

# Future Research Directions

**Performance and reliability of pseudo-Boolean proof logging and checking**

- Trim proof while verifying (as in DRAT-TRIM [HHW13a])
- Compress proof file using binary format
- Formally verifed end-to-end checking (as in [GMM⁺24, IOT⁺24, KLM⁺25])
- Faster proof logging and checking!

**Proof logging for other combinatorial problems and techniques**

- Model enumeration and counting
- Mixed integer linear programming *(suggested extension of VERIPB in [DEGH23])*
- SMT solving *(work on solvers CVC5, SMTINTERPOL, Z3, ... [BBC⁺23, HS22])*

# Future Research Directions

**Performance and reliability of pseudo-Boolean proof logging and checking**

- Trim proof while verifying (as in DRAT-TRIM [HHW13a])
- Compress proof file using binary format
- Formally verifed end-to-end checking (as in [GMM+24, IOT+24, KLM+25])
- Faster proof logging and checking!

**Proof logging for other combinatorial problems and techniques**

- Model enumeration and counting
- Mixed integer linear programming *(suggested extension of VeriPB in [DEGH23])*
- SMT solving *(work on solvers* CVC5, SMTInterpol, Z3, ... *[BBC+23, HS22])*

**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 and checking**

- Trim proof while verifying (as in DRAT-TRIM [HHW13a])
- Compress proof file using binary format
- Formally verifed end-to-end checking (as in [GMM+24, IOT+24, KLM+25])
- Faster proof logging and checking!

**Proof logging for other combinatorial problems and techniques**

- Model enumeration and counting
- Mixed integer linear programming *(suggested extension of VeriPB in [DEGH23])*
- SMT solving *(work on solvers cvc5, SMTInterpol, Z3, . . . [BBC+23, HS22])*

**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! ☺

# VeriPB Resources

VeriPB tutorials

- Slides from tutorials at *CP '22* [BMN22] and *IJCAI '23* [BMN23]
- Video tutorial at `https://youtu.be/s_5BIi4I22w`
- Videos from updated version at *WHOOPS '25* will hopefully be online soon at `https://jakobnordstrom.se/WHOOPS25/`

# VeriPB Resources

VeriPB tutorials

- Slides from tutorials at *CP '22* [BMN22] and *IJCAI '23* [BMN23]
- Video tutorial at `https://youtu.be/s_5BIi4I22w`
- Videos from updated version at *WHOOPS '25* will hopefully be online soon at `https://jakobnordstrom.se/WHOOPS25/`

Technical documentation [ABB+25] for SAT 2025 competition

- Available at `https://satcompetition.github.io/2025/output.html`

# VeriPB Resources

VeriPB tutorials

- Slides from tutorials at *CP '22* [BMN22] and *IJCAI '23* [BMN23]
- Video tutorial at `https://youtu.be/s_5BIi4I22w`
- Videos from updated version at *WHOOPS '25* will hopefully be online soon at `https://jakobnordstrom.se/WHOOPS25/`

Technical documentation [ABB+25] for SAT 2025 competition

- Available at `https://satcompetition.github.io/2025/output.html`

Details on specific proof logging techniques in [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23, BBN+24, DMM+24, GMM+24, HOGN24, IOT+24, MMN24, DHN+25, JBBJ25, KLM+25, MM25]

# VeriPB Resources

VeriPB tutorials

- Slides from tutorials at *CP '22* [BMN22] and *IJCAI '23* [BMN23]
- Video tutorial at `https://youtu.be/s_5BIi4I22w`
- Videos from updated version at *WHOOPS '25* will hopefully be online soon at `https://jakobnordstrom.se/WHOOPS25/`

Technical documentation [ABB$^+$25] for SAT 2025 competition

- Available at `https://satcompetition.github.io/2025/output.html`

Details on specific proof logging techniques in [EGMN20, GMN20, GMM$^+$20, GN21, GMN22, GMNO22, VDB22, BBN$^+$23, BGMN23, MM23, BBN$^+$24, DMM$^+$24, GMM$^+$24, HOGN24, IOT$^+$24, MMN24, DHN$^+$25, JBBJ25, KLM$^+$25, MM25]

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

# 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


[ABB+25] Markus Anders, Bart Bogaerts, Benjamin Bogø, Arthur Gontier, Wietze Koops, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Adrián Rebola-Pardo, and Yong Kiam Tan. Documentation of VeriPB and CakePB for the SAT competition 2025. Available at `https://satcompetition.github.io/2025/output.html`, April 2025.

[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.

# References II


[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.

[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.

# References III


[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.

[BBN+24] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, Tobias Paxian, and Dieter Vandesande. Certifying without loss of generality reasoning in solution-improving maximum satisfiability. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:28, September 2024.

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

[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.

# References IV


[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.

[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 `https://jakobnordstrom.se/presentations/`, August 2022.

[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 `https://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.

# References V


[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.

[BvdKM+21] Péter Biró, Joris van de Klundert, David F. Manlove, William Pettersson, Tommy Andersson, Lisa Burnapp, Pavel Chromy, Pablo Delgado, Piotr Dworczak, Bernadette Haase, Aline Hemke, Rachel Johnson, Xenia Klimentova, Dirk Kuypers, Alessandro Nanni Costa, Bart Smeulders, Frits C. R. Spieksma, María O. Valentín, and Ana Viana. Modelling and optimisation in European kidney exchange programmes. *European Journal of Operational Research*, 291(2):447–456, June 2021.

[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 VI


[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.

[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.

[DFS12] Nicholas Downing, Thibaut Feydy, and Peter J. Stuckey. Explaining alldifferent. In *Proceedings of the 35th Australasian Computer Science Conference (ACSC '12)*, pages 115–124, January 2012.

# References VII


[DHN+25] Simon Dold, Malte Helmert, Jakob Nordström, Gabriele Röger, and Tanja Schindler. Pseudo-Boolean proof logging for optimal classical planning. In *Proceedings of the 35th International Conference on Automated Planning and Scheduling (ICAPS '25)*, November 2025. To appear.

[DMM+24] Emir Demirović, Ciaran McCreesh, Matthew McIlree, Jakob Nordström, Andy Oertel, and Konstantin Sidorov. Pseudo-Boolean reasoning about states and transitions to certify dynamic programming and decision diagram algorithms. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:21, September 2024.

[EG23] Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. *Mathematical Programming*, 197(2):793–812, February 2023.

[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.

# References VIII


[Fle20] Mathias Fleury. *Formalization of Logical Calculi in Isabelle/HOL*. PhD thesis, Universität des Saarlandes, 2020. Available at `https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/28722`.

[GCS23] Graeme Gange, Geoffrey Chu, and Peter J. Stuckey. Certifying optimality in constraint programming. Manuscript. Available at `https://people.eng.unimelb.edu.au/pstuckey/papers/certified-cp.pdf`, 2023.

[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.

[GMM$^+$24] Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. End-to-end verification for subgraph solving. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI '24)*, pages 8038–8047, February 2024.

# References IX


[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 X


[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`.

[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.

[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.

# References XI


[HOGN24] Alexander Hoen, Andy Oertel, Ambros Gleixner, and Jakob Nordström. Certifying MIP-based presolve reductions for $0$–$1$ integer linear programs. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14742 of *Lecture Notes in Computer Science*, pages 310–328. Springer, May 2024.

[HS22] Jochen Hoenicke and Tanja Schindler. A simple proof format for SMT. In *Proceedings of the 20th Internal Workshop on Satisfiability Modulo Theories (SMT '22)*, volume 3185 of *CEUR Workshop Proceedings*, pages 54–70, August 2022.

[IOT$^+$24] Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström. Certified MaxSAT preprocessing. In *Proceedings of the 12th International Joint Conference on Automated Reasoning (IJCAR '24)*, volume 14739 of *Lecture Notes in Computer Science*, pages 396–418. Springer, July 2024.

[JBBJ25] Christoph Jabs, Jeremias Berg, Bart Bogaerts, and Matti Järvisalo. Certifying pareto-optimality in multi objective maximum satisfiability. In *Proceedings of the 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '25)*, volume 15697 of *Lecture Notes in Computer Science*, pages 108–129. Springer, May 2025.

# References XII


[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.

[KLM+25] Wietze Koops, Daniel Le Berre, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Yong Kiam Tan, and Marc Vinyals. Practically feasible proof logging for pseudo-Boolean optimization. In *Proceedings of the 31st International Conference on Principles and Practice of Constraint Programming (CP '25)*, volume 340 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:27, August 2025.

[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 XIII


[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.

[MM25] Matthew McIlree and Ciaran McCreesh. Certifying bounds propagation for integer multiplication constraints. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI '25)*, pages 11309–11317, February-March 2025.

[MMN24] Matthew McIlree, Ciaran McCreesh, and Jakob Nordström. Proof logging for the circuit constraint. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14743 of *Lecture Notes in Computer Science*, pages 38–55. Springer, May 2024.

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

# References XIV


[MO12] David F. Manlove and Gregg O'Malley. Paired and altruistic kidney donation in the UK: Algorithms and experimentation. In *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA '12)*, volume 7276 of *Lecture Notes in Computer Science*, pages 271–282. Springer, June 2012.

[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.

[OSC09] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, January 2009.

[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.

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

# References XV


[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.

[VS10] Michael Veksler and Ofer Strichman. A proof-producing CSP solver. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI '10)*, pages 204–209, July 2010.

[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.