End-to-End Verification for Subgraph Solving

Jakob Nordström

University of Copenhagen and Lund University

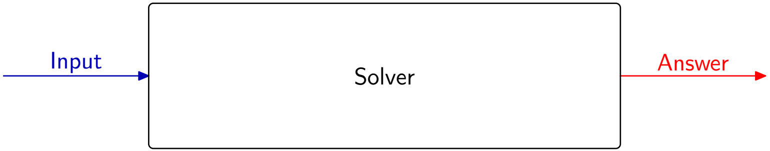
15th Pragmatics of SAT International Workshop
Pune, India
August 20, 2024

Joint work with Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Andy Oertel, and Yong Kiam Tan

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

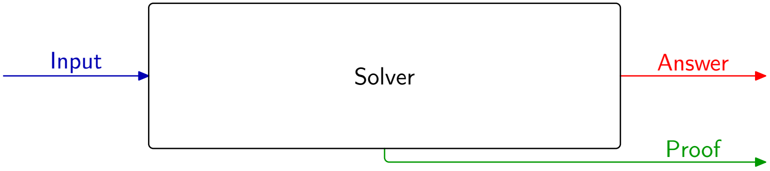
- Astounding progress last couple of decades on combinatorial solvers for, e.g.:
  - Boolean satisfiability (SAT) solving and optimization [BHvMW21]
  - Constraint programming [RvBW06]
  - Mixed integer linear programming [AW13, BR07]
  - Satisfiability modulo theories (SMT) solving [BHvMW21]
- Solvers very fast, but sometimes wrong (even most mature ones in industry and academia) [BLB10, CKSW13, AGJ+18, GSD19, GS19, BMN22, BBN+23]
- Only currently realistic solution: 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



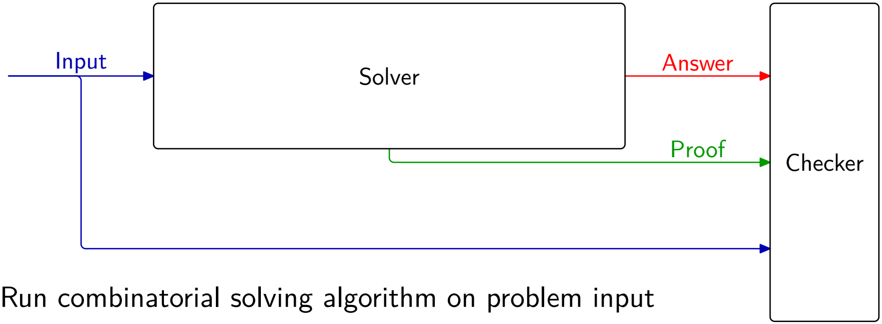
1. Run combinatorial solving algorithm on problem input

Proof Logging with Certifying Solvers: Workflow



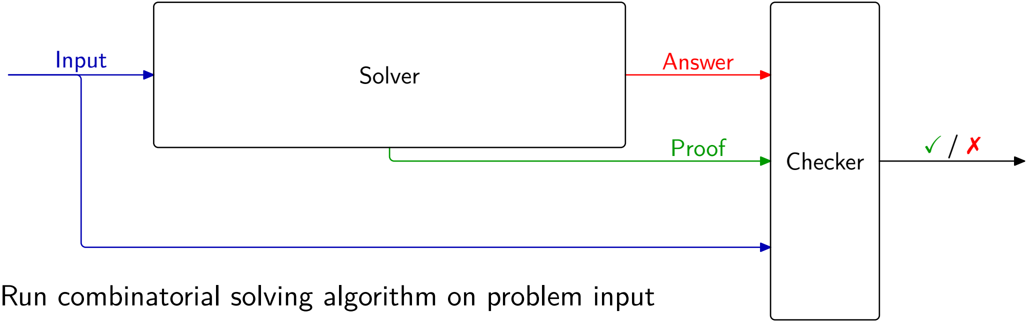
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

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

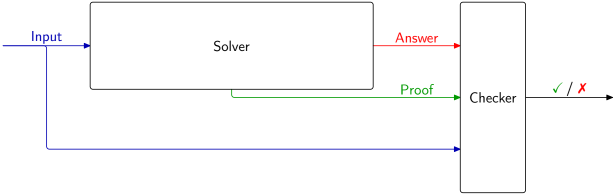
Proof Logging with Certifying Solvers: Workflow



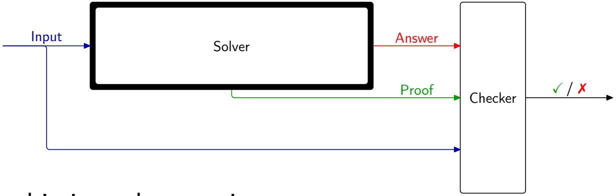
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

Proof Logging Desiderata

Proof format for certifying solver should be



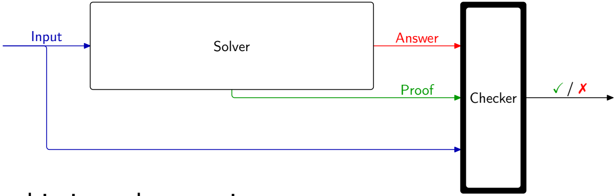
Proof Logging Desiderata



Proof format for certifying solver should be

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

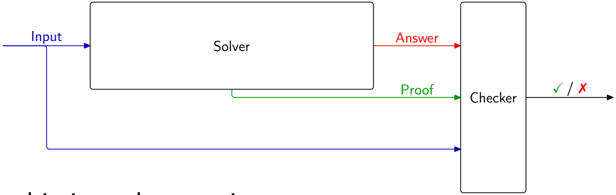
Proof Logging Desiderata



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

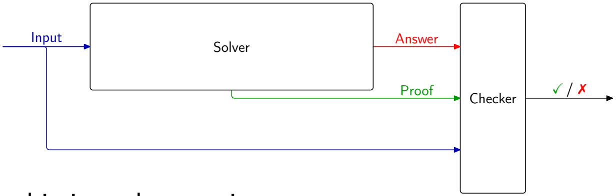


Proof format for certifying solver should be

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

Clear conflict expressivity vs. simplicity!
Asking for both perhaps a little bit too good to be true?

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?

Earlier proof logging approaches for SAT, MaxSAT, constraint programming, et cetera have struggled with this trade-off (and failed to master it)

My Main Message

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

My Main Message

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

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

My Main Message

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

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

Purpose of this talk:

1. Review basic set-up for proof logging beyond SAT

My Main Message

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

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

Purpose of this talk:

1. Review basic set-up for proof logging beyond SAT
2. Discuss a highly non-obvious application: Subgraph solving

My Main Message

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

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

Purpose of this talk:

1. Review basic set-up for proof logging beyond SAT
2. Discuss a highly non-obvious application: Subgraph solving
3. Describe a fully formally verified pipeline for such graph problems

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 ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)
- variables x_i take values $0 = \mathit{false}$ or $1 = \mathit{true}$

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

Some Types of Pseudo-Boolean Constraints

1. Disjunctive clauses

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

2. Cardinality constraints

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

3. General pseudo-Boolean constraints

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

Successful Applications of VeriPB Proof Logging

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

Successful Applications of VeriPB Proof Logging

Surprisingly, pseudo-Boolean reasoning is 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]
4. **Subgraph solving** [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]

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]

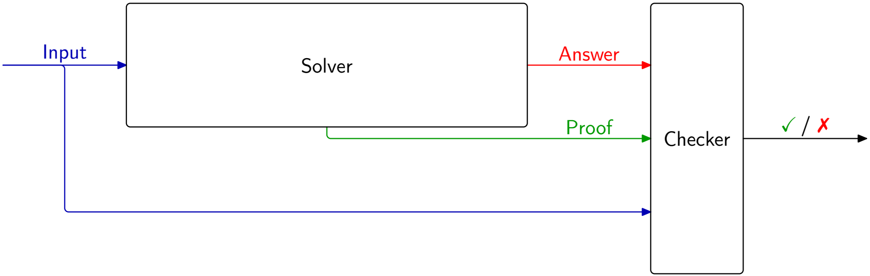
Otherwise

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

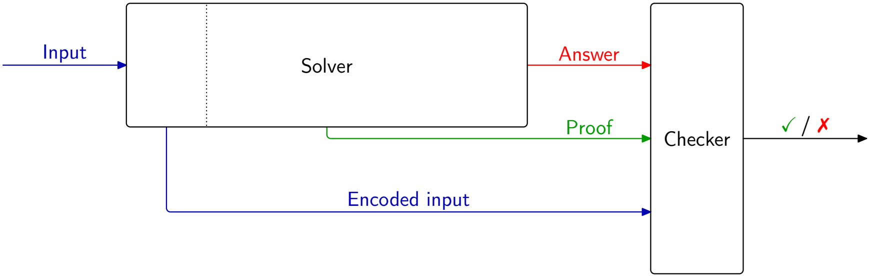
Goldilocks compromise between expressivity and simplicity:

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

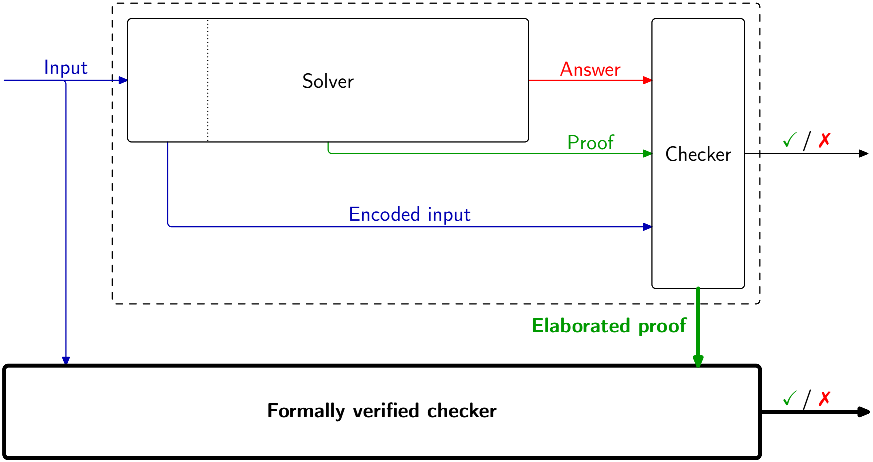
Proof Logging with Formally Verified Checking: Full Workflow



Proof Logging with Formally Verified Checking: Full Workflow



Proof Logging with Formally Verified Checking: Full Workflow



Subgraph Problems and the Glasgow Subgraph Solver

Some important subgraph problems

- Maximum clique in a given graph
- Subgraph isomorphism of pattern graph in target graph
- Maximum common connected subgraph of two given graphs

Subgraph Problems and the Glasgow Subgraph Solver

Some important subgraph problems

- Maximum clique in a given graph
- Subgraph isomorphism of pattern graph in target graph
- Maximum common connected subgraph of two given graphs

The Glasgow Subgraph Solver [ADH+19, GSS]

- State-of-the-art solver for such problems
- Sometimes the only solver returning an answer
- Can we trust that such an answer is correct?

Pseudo-Boolean Proof Logging for Subgraph Solving

All reasoning steps in Glasgow Subgraph Solver can be formalized efficiently in the cutting planes proof system [GMN20, GMM+20]

Pseudo-Boolean Proof Logging for Subgraph Solving

All reasoning steps in Glasgow Subgraph Solver can be formalized efficiently in the cutting planes proof system [GMN20, GMM+20]

Means that

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

Pseudo-Boolean Proof Logging for Subgraph Solving

All reasoning steps in Glasgow Subgraph Solver can be formalized efficiently in the cutting planes proof system [GMN20, GMM⁺20]

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

Let's see how this works for subgraph isomorphism

The Subgraph Isomorphism Problem

Input

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

The Subgraph Isomorphism Problem

Input

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

Task

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

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

Subgraph Isomorphism as a 0–1 Integer Linear Program

- 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)

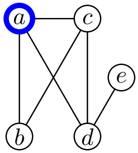
0–1 integer linear (pseudo-Boolean) encoding

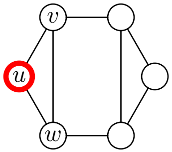
$$\sum_{v \in V(\mathcal{T})} x_{a,v} = 1 \qquad \text{[every pattern vertex } a \text{ maps somewhere]}$$

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

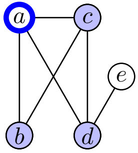
$$\overline{x}_{a,u} + \sum_{v \in N(u)} x_{b,v} \geq 1 \qquad \text{[pattern edge } (a,b) \text{ maps to target edge } (u,v)\text{]}$$

Degree Preprocessing Example (Vertex a Cannot Map to Vertex u)

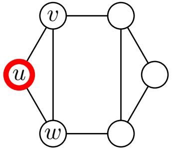




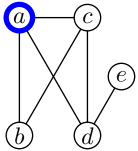
Degree Preprocessing Example (Vertex a Cannot Map to Vertex u)



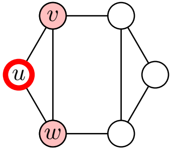
$$\overline{x}_{a,u} + x_{b,v} + x_{b,w} \geq 1$$
$$\overline{x}_{a,u} + x_{c,v} + x_{c,w} \geq 1$$
$$\overline{x}_{a,u} + x_{d,v} + x_{d,w} \geq 1$$



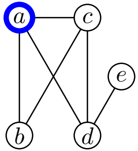
Degree Preprocessing Example (Vertex a Cannot Map to Vertex u)



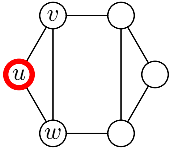
$$\overline{x}_{a,u} + x_{b,v} + x_{b,w} \geq 1$$
$$\overline{x}_{a,u} + x_{c,v} + x_{c,w} \geq 1$$
$$\overline{x}_{a,u} + x_{d,v} + x_{d,w} \geq 1$$
$$\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} \geq 4$$
$$\overline{x}_{a,w} + \overline{x}_{b,w}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



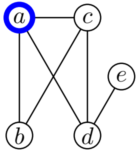
Degree Preprocessing Example (Vertex a Cannot Map to Vertex u)



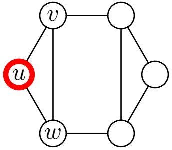
$$\overline{x}_{a,u} + x_{b,v} + x_{b,w} \geq 1$$
$$\overline{x}_{a,u} + x_{c,v} + x_{c,w} \geq 1$$
$$\overline{x}_{a,u} + x_{d,v} + x_{d,w} \geq 1$$
$$\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} \geq 4$$
$$\overline{x}_{a,w} + \overline{x}_{b,w}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



Degree Preprocessing Example (Vertex a Cannot Map to Vertex u)

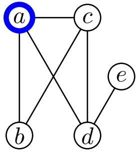


$$\overline{x}_{a,u} + x_{b,v} + x_{b,w} \geq 1$$
$$\overline{x}_{a,u} + x_{c,v} + x_{c,w} \geq 1$$
$$\overline{x}_{a,u} + x_{d,v} + x_{d,w} \geq 1$$
$$\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} \geq 4$$
$$\overline{x}_{a,w} + \overline{x}_{b,w}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

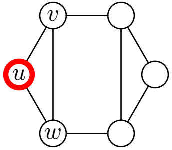


Sum up all constraints & divide by 3 to obtain

Degree Preprocessing Example (Vertex a Cannot Map to Vertex u)



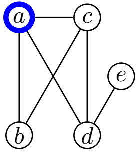
$$\overline{x}_{a,u} + x_{b,v} + x_{b,w} \geq 1$$
$$\overline{x}_{a,u} + x_{c,v} + x_{c,w} \geq 1$$
$$\overline{x}_{a,u} + x_{d,v} + x_{d,w} \geq 1$$
$$\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} \geq 4$$
$$\overline{x}_{a,w} + \overline{x}_{b,w}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



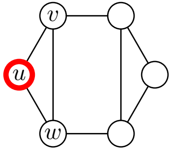
Sum up all constraints & divide by 3 to obtain

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

Degree Preprocessing Example (Vertex a Cannot Map to Vertex u)



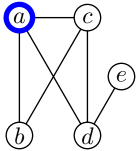
$$
\begin{aligned}
\overline{x}_{a,u} + x_{b,v} + x_{b,w} &\geq 1 \\
\overline{x}_{a,u} + x_{c,v} + x_{c,w} &\geq 1 \\
\overline{x}_{a,u} + x_{d,v} + x_{d,w} &\geq 1 \\
\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} &\geq 4 \\
\overline{x}_{a,w}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



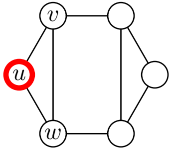
Sum up all constraints & divide by 3 to obtain

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

Degree Preprocessing Example (Vertex a Cannot Map to Vertex u)



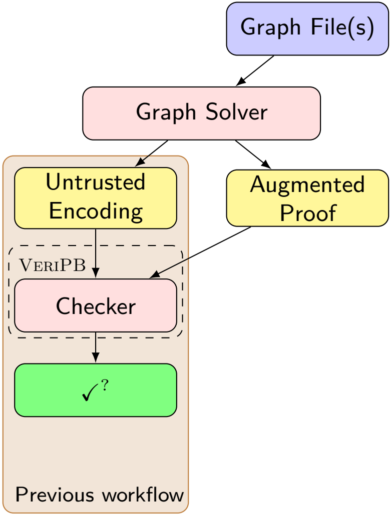
$$\begin{aligned}
\overline{x}_{a,u} + x_{b,v} + x_{b,w} &\geq 1 \\
\overline{x}_{a,u} + x_{c,v} + x_{c,w} &\geq 1 \\
\overline{x}_{a,u} + x_{d,v} + x_{d,w} &\geq 1 \\
\overline{x}_{a,v} + \overline{x}_{b,v} + \overline{x}_{c,v} + \overline{x}_{d,v} + \overline{x}_{e,v} &\geq 4 \\
\overline{x}_{a,w}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



Sum up all constraints & divide by 3 to obtain

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

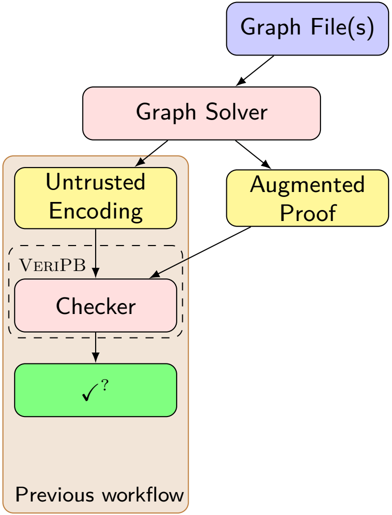
Workflow for Subgraph Solvers with Pseudo-Boolean Proof Logging



Problem solved!

- Modern graph solvers are complex
- Cannot trust (or maybe even understand) the code
- But now we have pseudo-Boolean proof logging [GMN20, GMM+20]

Workflow for Subgraph Solvers with Pseudo-Boolean Proof Logging



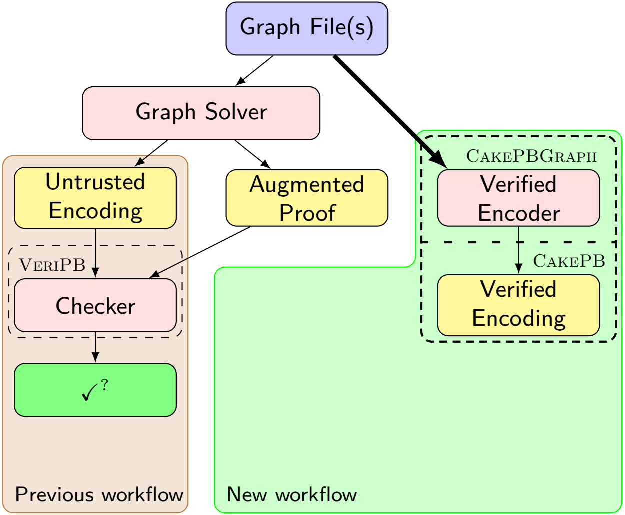
Problem solved!

- Modern graph solvers are complex
- Cannot trust (or maybe even understand) the code
- But now we have pseudo-Boolean proof logging [GMN20, GMM+20]

Or is it. . .

- Can we trust the proof checker?
- And how do we know the 0–1 ILP encoding is correct?

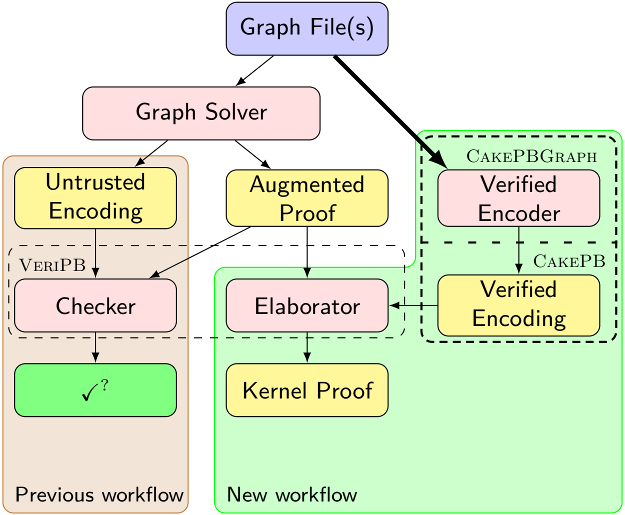
End-to-End Verification Workflow



Verified workflow:

1. Encode problem using formally verified encoder

End-to-End Verification Workflow



Verified workflow:

1. Encode problem using formally verified encoder
2. Elaborate augmented proof to kernel proof

Proof Elaboration

Two versions of the VeriPB proof format:

- Augmented proof contains helpful, powerful rules for easier proof logging
- Kernel proof has restricted subset of proof rules that are easier to check

Proof Elaboration

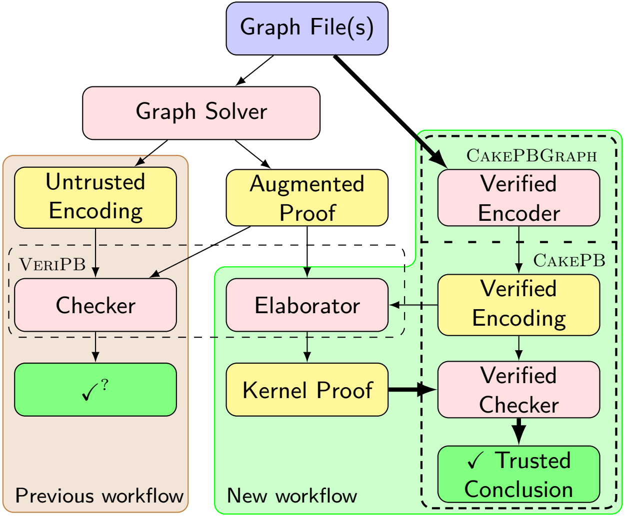
Two versions of the VeriPB proof format:

- Augmented proof contains helpful, powerful rules for easier proof logging
- Kernel proof has restricted subset of proof rules that are easier to check

How to get kernel proof?

- VeriPB can elaborate an augmented proof to a kernel proof

End-to-End Verification Workflow (cont.)



Verified workflow:

1. Encode problem using formally verified encoder
2. Elaborate augmented proof to kernel proof
3. Check kernel proof using formally verified checker

Can We Trust This Workflow?

The following needs to be trusted or closely inspected:

- Higher-order logic (HOL) definitions of input parser and problems
  - easy to check
- HOL model of CakeML environment and correspondence to real system
  - validated extensively
- HOL4 theorem prover, including logic, implementation, and execution environment
  - well established

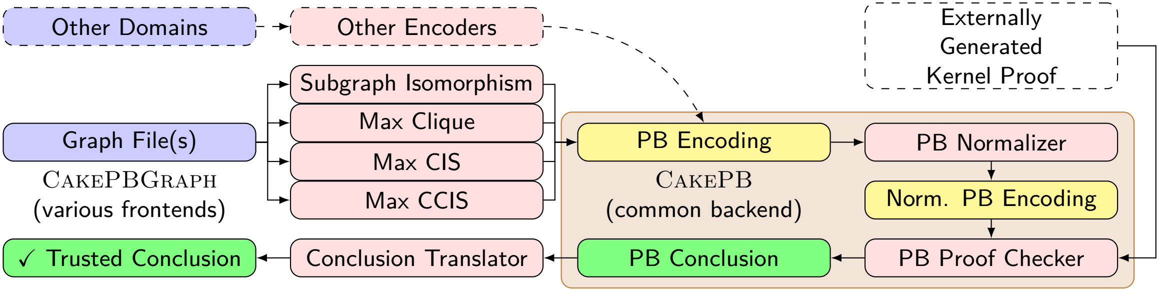
Can We Trust This Workflow?

The following needs to be trusted or closely inspected:

- Higher-order logic (HOL) definitions of input parser and problems
  - easy to check
- HOL model of CakeML environment and correspondence to real system
  - validated extensively
- HOL4 theorem prover, including logic, implementation, and execution environment
  - well established

Gives the highest assurance standard for formally verified checker CakePB (`https://gitlab.com/MIAOresearch/software/cakepb`)

Extensible Checking Framework



- Common backend: Performs reasoning on 0–1 ILP (a.k.a. pseudo-Boolean encoding)
- Frontend: Translates specific problem class into 0–1 ILP and back

Future Research Directions

Proof processing

- Trimming proof while verifying (as in DRAT-TRIM [HHW13a])
- Solution reconstruction
- Composition of proofs

Future Research Directions

Proof processing

- Trimming proof while verifying (as in DRAT-TRIM [HHW13a])
- Solution reconstruction
- Composition of proofs

Proof logging for other combinatorial problems

- Automated planning *(building on [ERH17, ERH18])*
- Mixed integer linear programming *(suggested extension of* VeriPB *in [DEGH23])*
- SMT solving *(work on solvers* CVC5, SMTInterpol, Z3, … *[BBC+23, HS22])*

Future Research Directions

Proof processing

- Trimming proof while verifying (as in DRAT-TRIM [HHW13a])
- Solution reconstruction
- Composition of proofs

Proof logging for other combinatorial problems

- Automated planning *(building on [ERH17, ERH18])*
- 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

Proof processing

- Trimming proof while verifying (as in DRAT-TRIM [HHW13a])
- Solution reconstruction
- Composition of proofs

Proof logging for other combinatorial problems

- Automated planning *(building on [ERH17, ERH18])*
- 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! ☺

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 (or only) approach
- Cutting planes reasoning with pseudo-Boolean constraints hits a sweet spot between simplicity and expressivity (for much more general problems)
- Can be combined with formal methods to yield end-to-end verification

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 (or only) approach
- Cutting planes reasoning with pseudo-Boolean constraints hits a sweet spot between simplicity and expressivity (for much more general problems)
- Can be combined with formal methods to yield end-to-end verification
- **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 the most promising (or only) approach
- Cutting planes reasoning with pseudo-Boolean constraints hits a sweet spot between simplicity and expressivity (for much more general problems)
- Can be combined with formal methods to yield end-to-end verification
- **Action point:** What problems can VeriPB solve for you? ☺

Thank you for your attention!

References I

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

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

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

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

References II

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

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

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

[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)*, September 2024. To appear.

[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*.

References III

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

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

[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.

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

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

References IV

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

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

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

[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.

References V

[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)*, September 2024. To appear.

[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.

[ERH17] Salomé Eriksson, Gabriele Röger, and Malte Helmert. Unsolvability certificates for classical planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS '17)*, pages 88–97, June 2017.

[ERH18] Salomé Eriksson, Gabriele Röger, and Malte Helmert. A proof system for unsolvable planning tasks. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS '18)*, pages 65–73, June 2018.

References VI

[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 368h AAAI Conference on Artificial Intelligence (AAAI '24)*, pages 8038–8047, February 2024.

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

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

References VII

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

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

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

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

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

References VIII

[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.

[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.

References IX

[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.

[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.

[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.

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

References X

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

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