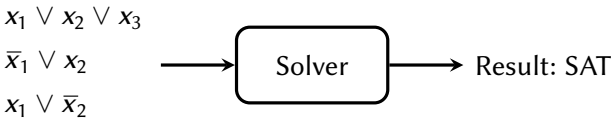
Proof Logging for Projected Enumeration (and Counting?) Problems in VeriPB

Ciaran McCreesh, Jakob Nordström, **Andy Oertel**, and Yong Kiam Tan

32nd International Conference on Principles and Practice of Constraint Programming

July 21, 2026

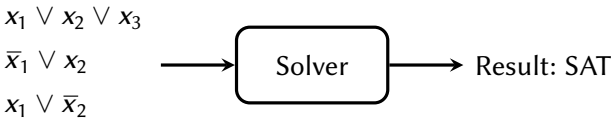
Combinatorial Solving



Combinatorial problem:

- Collection of constraint to be satisfied (e.g., clauses)
- Variables take discrete values (e.g., Boolean with domain $\{0, 1\}$)

Combinatorial Solving



Combinatorial problem:

- Collection of constraint to be satisfied (e.g., clauses)
- Variables take discrete values (e.g., Boolean with domain $\{0, 1\}$)

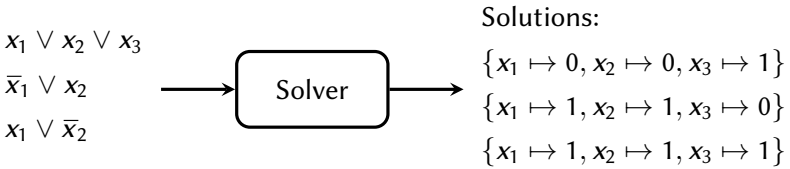
Decision problem:

- Constraints satisfiable or unsatisfiable?

Optimization problem:

- Best solution(s) with respect to objective function

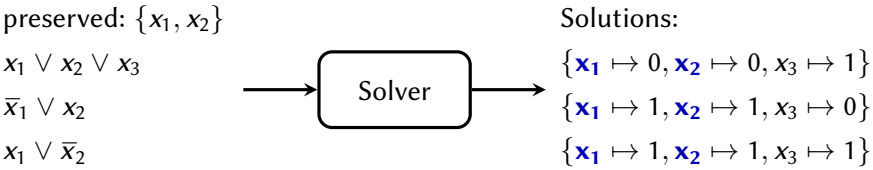
Solution Enumeration



Enumeration problem:

- List solutions
- Complete enumeration: List **all** solutions
- Enumeration in SAT community known as AllSAT [TS16]
- Enumeration used for, e.g., computer algebra and graph theory [AMV22, TTT06]

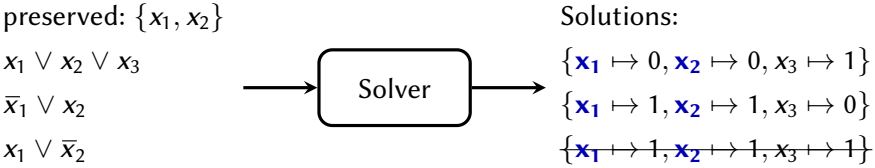
Projected Solution Enumeration



Projected enumeration problem:

- Only interested in unique solutions to preserved variables
- Different names for preserved set in literature:
  - projection [FHH21], priority [ACMS15] or data [BNAH25] variables

Projected Solution Enumeration



Projected enumeration problem:

- Only interested in unique solutions to preserved variables
- Different names for preserved set in literature:
  - projection [FHH21], priority [ACMS15] or data [BNAH25] variables
- Solutions are equivalent if they are equivalent with respect to the preserved variables
- Example: $\{\mathbf{x_1} \mapsto 1, \mathbf{x_2} \mapsto 1, x_3 \mapsto 0\}$, $\{\mathbf{x_1} \mapsto 1, \mathbf{x_2} \mapsto 1, x_3 \mapsto 1\}$ project to $\{\mathbf{x_1} \mapsto 1, \mathbf{x_2} \mapsto 1\}$

Correctness of Solver Result

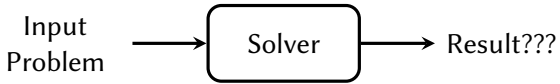
- Solvers can be incorrect [BLB10, CKSW13, AGJ+18, GSD19, GS19, BBN+23, GCS23]

Correctness of Solver Result

- Solvers can be incorrect [BLB10, CKSW13, AGJ+18, GSD19, GS19, BBN+23, GCS23]

How to guarantee correct results?

- Testing: no correctness guarantees
- Formal verification: does not scale to modern solvers
- Proof logging (our approach):

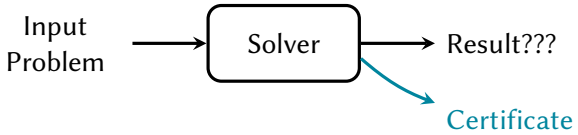


Correctness of Solver Result

- Solvers can be incorrect [BLB10, CKSW13, AGJ+18, GSD19, GS19, BBN+23, GCS23]

How to guarantee correct results?

- Testing: no correctness guarantees
- Formal verification: does not scale to modern solvers
- Proof logging (our approach):



Workflow for proof logging:

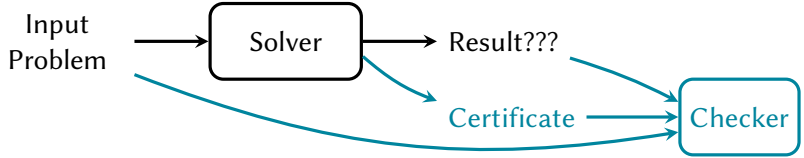
1. Solver outputs certificate for correctness of result

Correctness of Solver Result

- Solvers can be incorrect [BLB10, CKSW13, AGJ+18, GSD19, GS19, BBN+23, GCS23]

How to guarantee correct results?

- Testing: no correctness guarantees
- Formal verification: does not scale to modern solvers
- Proof logging (our approach):



Workflow for proof logging:

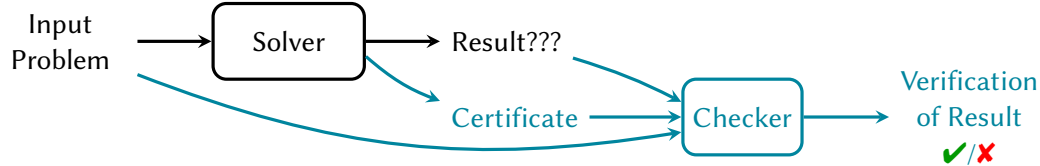
1. Solver outputs certificate for correctness of result
2. Independent checker verifies result w.r.t. input using certificate

Correctness of Solver Result

- Solvers can be incorrect [BLB10, CKSW13, AGJ+18, GSD19, GS19, BBN+23, GCS23]

How to guarantee correct results?

- Testing: no correctness guarantees
- Formal verification: does not scale to modern solvers
- Proof logging (our approach):



Workflow for proof logging:

1. Solver outputs certificate for correctness of result
2. Independent checker verifies result w.r.t. input using certificate
3. Checker confirms correctness of result or that *something* is wrong

Proof Logging for (Projected) Enumeration

Proof logging exists for

- decision problems (e.g., [ZM03, HHW13, BGMN23])
- optimization problems (e.g., [CGS17, GMM+24])
- and to some extent for counting problems (e.g., [FHR22, BNAH25])

Proof Logging for (Projected) Enumeration

Proof logging exists for

- decision problems (e.g., [ZM03, HHW13, BGMN23])
- optimization problems (e.g., [CGS17, GMM+24])
- and to some extent for counting problems (e.g., [FHR22, BNAH25])

No proof logging for enumeration problems!

Proof Logging for (Projected) Enumeration

Proof logging exists for

- decision problems (e.g., [ZM03, HHW13, BGMN23])
- optimization problems (e.g., [CGS17, GMM+24])
- and to some extent for counting problems (e.g., [FHR22, BNAH25])

No proof logging for enumeration problems!

This work:

- First proof logging for (projected) enumeration
- Solution count preserving reformulation proofs
- Formal guarantees for enumeration and count preserving reformulations

Proof Preliminaries

- Boolean variable x: with domain 0 (false) and 1 (true)
- Literal: x or negation $\overline{x} = 1 - x$
- Proof lines: Pseudo-Boolean (PB) constraint (integer linear inequality over literals)

$$3x_1 + 2x_2 + 5\overline{x}_3 \geq 5$$

- Clause: disjunction of literals / at-least-one constraint

$$x \vee \overline{x}_2 \vee \overline{x}_3 \iff x + \overline{x}_2 + \overline{x}_3 \geq 1$$

- Reification: $x \Leftrightarrow C$ for variable x and constraint $C \doteq \sum_i a_i \ell_i \geq A$ is constraints

$$x \Rightarrow C \doteq A\overline{x} + \sum_i a_i \ell_i \geq A$$
$$x \Leftarrow C \doteq \left(\sum_i a_i - A + 1\right)x + \sum_i a_i \overline{\ell}_i \geq \sum_i a_i - A + 1$$

(Projected) Enumeration Proofs in VeriPB

Idea:

- Enumeration: log solutions and disallow them to be logged again
- Goal (complete enumeration): prove that there are no more solutions
  - Same as showing unsatisfiability (by deriving contradiction)

(Projected) Enumeration Proofs in VeriPB

Idea:

- Enumeration: log solutions and disallow them to be logged again
- Goal (complete enumeration): prove that there are no more solutions
  - Same as showing unsatisfiability (by deriving contradiction)

Remaining problems:

- Preventing logging a solution twice (blocking)
- Deriving constraint in proof can remove solutions
- Deleting constraint in proof can add solutions

Blocking (Projected) Solutions

Blocking clause:

- Example: If preserved set $\{x_1, x_2\}$ and solution $\{x_1 \mapsto 1, x_2 \mapsto 0, x_3 \mapsto 1\}$, then solution is blocked by

$$\overline{x}_1 \vee x_2$$

Blocking (Projected) Solutions

Blocking clause:

- Example: If preserved set $\{x_1, x_2\}$ and solution $\{x_1 \mapsto 1, x_2 \mapsto 0, x_3 \mapsto 1\}$, then solution is blocked by

$$\overline{x}_1 \vee x_2$$

- In general: Given preserved set P and solution σ, the clause

$$\bigvee_{\{p \in P : \sigma(p) = 0\}} p \vee \bigvee_{\{p \in P : \sigma(p) = 1\}} \overline{p}$$

  excludes all solutions assigning the same values to the variables in P as σ
- New `solx` rule for logging and blocking solution

Deriving Constraints

Danger:

- Deriving a constraint could remove (projected) solutions

Deriving Constraints

Danger:

- Deriving a constraint could remove (projected) solutions

Implicational rules:

- Derived constraint is satisfied by all solutions to current constraints
- No issue for enumeration!

Deriving Constraints

Danger:

- Deriving a constraint could remove (projected) solutions

Implicational rules:

- Derived constraint is satisfied by all solutions to current constraints
- No issue for enumeration!

Strengthening rules:

- Derived constraint satisfied by at least one but not all solutions to current constraints
- Example: $\{x_1 \vee x_2 \vee x_3, \quad \overline{x}_1 \vee x_2, \quad x_1 \vee \overline{x}_2\}$
  - W.l.o.g. $x_3 \mapsto 1$, strengthening allows us to derive clause x_3
- Strengthening can remove solution that have not been enumerated!

Strengthening in More Detail

Redundance-based strengthening [GN21]:

- Substitution: Mapping variables to literals/values (e.g., $\omega' = \{x_1 \mapsto 1, x_2 \mapsto \overline{x}_3\}$)

Strengthening in More Detail

Redundance-based strengthening [GN21]:

- Substitution: Mapping variables to literals/values (e.g., $\omega' = \{x_1 \mapsto 1, x_2 \mapsto \overline{x}_3\}$)
- Constraint C can be derived from constraints F given substitution ω and showing

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

- If solution σ satisfies F but falsifies C, then $\sigma \circ \omega$ satisfies $F \cup \{C\}$
- E.g., $\sigma' = \{x_1 \mapsto 0, x_2 \mapsto 1, x_3 \mapsto 1\}$, then $\sigma' \circ \omega' = \{x_1 \mapsto 1, x_2 \mapsto \overline{x}_3 \mapsto 0, x_3 \mapsto 1\}$

Strengthening in More Detail

Redundance-based strengthening [GN21]:

- Substitution: Mapping variables to literals/values (e.g., $\omega' = \{x_1 \mapsto 1, x_2 \mapsto \overline{x}_3\}$)
- Constraint C can be derived from constraints F given substitution ω and showing

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

- If solution σ satisfies F but falsifies C, then $\sigma \circ \omega$ satisfies $F \cup \{C\}$
- E.g., $\sigma' = \{x_1 \mapsto 0, x_2 \mapsto 1, x_3 \mapsto 1\}$, then $\sigma' \circ \omega' = \{x_1 \mapsto 1, x_2 \mapsto \overline{x}_3 \mapsto 0, x_3 \mapsto 1\}$
- Observation: Only assignment to variables in domain of substitution are changed
- If ω only changes non-preserved variables, $\sigma \circ \omega$ and σ project to same solution
- Domain of substitution can not contain preserved variables

Strengthening in More Detail

Redundance-based strengthening [GN21]:

- Substitution: Mapping variables to literals/values (e.g., $\omega' = \{x_1 \mapsto 1, x_2 \mapsto \overline{x}_3\}$)
- Constraint C can be derived from constraints F given substitution ω and showing

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

- If solution σ satisfies F but falsifies C, then $\sigma \circ \omega$ satisfies $F \cup \{C\}$
- E.g., $\sigma' = \{x_1 \mapsto 0, x_2 \mapsto 1, x_3 \mapsto 1\}$, then $\sigma' \circ \omega' = \{x_1 \mapsto 1, x_2 \mapsto \overline{x}_3 \mapsto 0, x_3 \mapsto 1\}$
- Observation: Only assignment to variables in domain of substitution are changed
- If ω only changes non-preserved variables, $\sigma \circ \omega$ and σ project to same solution
- Domain of substitution can not contain preserved variables

- Dominance-based strengthening and deletion in VeriPB works similarly
- Key takeaway: Only substitutions need to be restricted

Projected Enumeration Example

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee x_2 \vee x_3$$
$$\overline{x}_1 \vee x_2$$
$$x_1 \vee \overline{x}_2$$

Enumerated solutions:

proof.pbp

```
pseudo-Boolean proof version 3.0
```

Projected Enumeration Example

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee x_2 \vee x_3$$
$$\overline{x}_1 \vee x_2$$
$$x_1 \vee \overline{x}_2$$

Enumerated solutions:

proof.pbp

```
pseudo-Boolean proof version 3.0
% W.l.o.g. x3 is 1 with substitution x3 -> 1
red 1 x3 >= 1 : x3 -> 1;
```

Projected Enumeration Example

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee x_2 \vee x_3$$
$$\overline{x}_1 \vee x_2$$
$$x_1 \vee \overline{x}_2$$

Enumerated solutions:

1. $\{x_1 \mapsto 0, x_2 \mapsto 0\}$

proof.pbp

```
pseudo-Boolean proof version 3.0
% W.l.o.g. x3 is 1 with substitution x3 -> 1
red 1 x3 >= 1 : x3 -> 1;
% Log first solution x1 -> 0 and x2 -> 0
solx ~x1 ~x2;
```

Projected Enumeration Example

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee x_2 \vee x_3$$
$$\overline{x}_1 \vee x_2$$
$$x_1 \vee \overline{x}_2$$

Enumerated solutions:

1. $\{x_1 \mapsto 0, x_2 \mapsto 0\}$

proof.pbp

```
pseudo-Boolean proof version 3.0
% W.l.o.g. x3 is 1 with substitution x3 -> 1
red 1 x3 >= 1 : x3 -> 1;
% Log first solution x1 -> 0 and x2 -> 0
solx ~x1 ~x2;
% No more solutions with x1 -> 0
rup 1 x1 >= 1;
```

Projected Enumeration Example

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee x_2 \vee x_3$$
$$\overline{x}_1 \vee x_2$$
$$x_1 \vee \overline{x}_2$$

Enumerated solutions:

1. $\{x_1 \mapsto 0, x_2 \mapsto 0\}$
2. $\{x_1 \mapsto 1, x_2 \mapsto 1\}$

proof.pbp

```
pseudo-Boolean proof version 3.0
% W.l.o.g. x3 is 1 with substitution x3 -> 1
red 1 x3 >= 1 : x3 -> 1;
% Log first solution x1 -> 0 and x2 -> 0
solx ~x1 ~x2;
% No more solutions with x1 -> 0
rup 1 x1 >= 1;
% Log second solution x1 -> 1 and x2 -> 1
solx x1 x2;
```

Projected Enumeration Example

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee x_2 \vee x_3$$
$$\overline{x}_1 \vee x_2$$
$$x_1 \vee \overline{x}_2$$

Enumerated solutions:

1. $\{x_1 \mapsto 0, x_2 \mapsto 0\}$
2. $\{x_1 \mapsto 1, x_2 \mapsto 1\}$

proof.pbp

```
pseudo-Boolean proof version 3.0
% W.l.o.g. x3 is 1 with substitution x3 -> 1
red 1 x3 >= 1 : x3 -> 1;
% Log first solution x1 -> 0 and x2 -> 0
solx ~x1 ~x2;
% No more solutions with x1 -> 0
rup 1 x1 >= 1;
% Log second solution x1 -> 1 and x2 -> 1
solx x1 x2;
% No more solutions
rup >= 1;
output NONE;
conclusion ENUMERATION_COMPLETE 2;
end pseudo-Boolean proof;
```

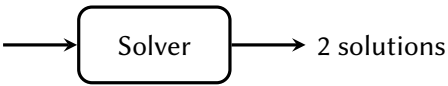
(Projected) Model/Solution Counting

preserved: $\{x_1, x_2\}$

$x_1 \vee x_2 \vee x_3$

$\overline{x}_1 \vee x_2$

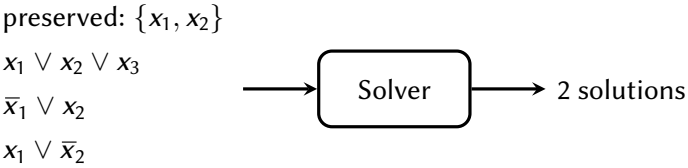
$x_1 \vee \overline{x}_2$



(Projected) counting:

- Give the number of (projected) solutions

(Projected) Model/Solution Counting



(Projected) counting:

- Give the number of (projected) solutions

Idea for certification:

- Inspiration from CPOG framework [BNAH25, BTH25]
- Reformulate problem into “easy-to-count” form preserving (projected) solution count
- Reformulation proofs already supported by VeriPB (e.g., prove equisatisfiable)

Changing the Preserved Set

Idea for changing preserved set:

- Added/removed variable is forced to a value by other preserved variables

Changing the Preserved Set

Idea for changing preserved set:

- Added/removed variable is forced to a value by other preserved variables

Rule:

- Add/remove x w.r.t. constraint C over other preserved variables
- Show that $x \Leftrightarrow C$ holds
- If C is satisfied, then x is 1
- If C is falsified, then x is 0

Counting Example

Original problem

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee \overline{x}_2$$

$$\overline{x}_1 \vee x_2$$

proof.pbp

```
pseudo-Boolean proof version 3.0
```

Counting Example

Original problem

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee \overline{x}_2$$

$$\overline{x}_1 \vee x_2$$

proof.pbp

```
pseudo-Boolean proof version 3.0
% Define variable s1 <==> x1 + x2 >= 2
red 2 ~s1 1 x1 1 x2 >= 2 : s1 0;
red 1 s1 1 ~x1 1 ~x2 >= 1 : s1 1;
```

Counting Example

Original problem

Preserved set: $\{x_1, x_2\}$

Constraints:

$x_1 \vee \overline{x}_2$

$\overline{x}_1 \vee x_2$

proof.pbp

```
pseudo-Boolean proof version 3.0
% Define variable s1 <==> x1 + x2 >= 2
red 2 ~s1 1 x1 1 x2 >= 2 : s1 0;
red 1 s1 1 ~x1 1 ~x2 >= 1 : s1 1;
% Add s1 to preserved with s1 <==> x1 + x2 >= 2
preserved_add s1 1 x1 1 x2 >= 2;
```

Counting Example

Original problem

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee \overline{x}_2$$

$$\overline{x}_1 \vee x_2$$

proof.pbp

```
pseudo-Boolean proof version 3.0
% Define variable s1 <==> x1 + x2 >= 2
red 2 ~s1 1 x1 1 x2 >= 2 : s1 0;
red 1 s1 1 ~x1 1 ~x2 >= 1 : s1 1;
% Add s1 to preserved with s1 <==> x1 + x2 >= 2
preserved_add s1 1 x1 1 x2 >= 2;
% Remove x1 and x2 from preserved
preserved_rm x1 1 s1 1 x2 >= 2;
preserved_rm x2 1 s1 >= 1;
```

Counting Example

Original problem

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee \overline{x}_2$$

$$\overline{x}_1 \vee x_2$$

proof.pbp

```
pseudo-Boolean proof version 3.0
% Define variable s1 <==> x1 + x2 >= 2
red 2 ~s1 1 x1 1 x2 >= 2 : s1 0;
red 1 s1 1 ~x1 1 ~x2 >= 1 : s1 1;
% Add s1 to preserved with s1 <==> x1 + x2 >= 2
preserved_add s1 1 x1 1 x2 >= 2;
% Remove x1 and x2 from preserved
preserved_rm x1 1 s1 1 x2 >= 2;
preserved_rm x2 1 s1 >= 1;
% Delete definitions of s1 using solutions
del id 3 : x1 1 x2 1; del id 4 : x1 0 x2 0;
```

Counting Example

Original problem

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee \overline{x}_2$$

$$\overline{x}_1 \vee x_2$$

proof.pbp

```
pseudo-Boolean proof version 3.0
% Define variable s1 <==> x1 + x2 >= 2
red 2 ~s1 1 x1 1 x2 >= 2 : s1 0;
red 1 s1 1 ~x1 1 ~x2 >= 1 : s1 1;
% Add s1 to preserved with s1 <==> x1 + x2 >= 2
preserved_add s1 1 x1 1 x2 >= 2;
% Remove x1 and x2 from preserved
preserved_rm x1 1 s1 1 x2 >= 2;
preserved_rm x2 1 s1 >= 1;
% Delete definitions of s1 using solutions
del id 3 : x1 1 x2 1; del id 4 : x1 0 x2 0;
% Delete original constraint with one solution
del id 1 : x1 0 x2 0; del id 2 : x1 0 x2 0;
```

Counting Example

Original problem

Preserved set: $\{x_1, x_2\}$

Constraints:

$$x_1 \vee \overline{x}_2$$

$$\overline{x}_1 \vee x_2$$

Reformulated problem

Preserved set: $\{s_1\}$

Constraints: $\emptyset$

Number of solutions: 2

proof.pbp

```
pseudo-Boolean proof version 3.0
% Define variable s1 <==> x1 + x2 >= 2
red 2 ~s1 1 x1 1 x2 >= 2 : s1 0;
red 1 s1 1 ~x1 1 ~x2 >= 1 : s1 1;
% Add s1 to preserved with s1 <==> x1 + x2 >= 2
preserved_add s1 1 x1 1 x2 >= 2;
% Remove x1 and x2 from preserved
preserved_rm x1 1 s1 1 x2 >= 2;
preserved_rm x2 1 s1 >= 1;
% Delete definitions of s1 using solutions
del id 3 : x1 1 x2 1; del id 4 : x1 0 x2 0;
% Delete original constraint with one solution
del id 1 : x1 0 x2 0; del id 2 : x1 0 x2 0;
% Output has same projected solution count
output EQUIENUMERABLE FILE; ...
```

Formally Verified Checking

Can we trust the checker?

- VeriPB checker might contain bugs

Formally Verified Checking

Can we trust the checker?

- VeriPB checker might contain bugs

Formally verified checker CakePB

- Verified in CakeML ecosystem
- Supports enumeration and count preserving reformulation proofs

Formally Verified Checking

Can we trust the checker?

- VeriPB checker might contain bugs

Formally verified checker CakePB

- Verified in CakeML ecosystem
- Supports enumeration and count preserving reformulation proofs

Trust base:

1. Higher-order logic (HOL) definitions of parser and problems
   - kept as simple as possible, easy to check
2. HOL model of CakeML environment and correspondence to real system
   - has been validated extensively
3. HOL4 theorem prover: its logic, implementation, and execution environment
   - separated and trustworthy kernel checks every logical inference

Experimental Results

Maximal clique enumeration [TTT06]:

- Adapted implementation by Gocht et al. [GMM+20]
  - no guarantees about enumeration, no formal verification
- Guarantee complete enumeration
- Formally verified maximal clique enumeration frontend for CakePB
- One instance out of memory for CakePB compared to Gocht et al.

Experimental Results

Maximal clique enumeration [TTT06]:

- Adapted implementation by Gocht et al. [GMM+20]
  - no guarantees about enumeration, no formal verification
- Guarantee complete enumeration
- Formally verified maximal clique enumeration frontend for CakePB
- One instance out of memory for CakePB compared to Gocht et al.

Enumeration in Glasgow Constraint Solver [GMN22]:

- Projected enumeration of pseudo-Boolean encoding of constraint problem
- Proofs checked by CakePB
- Verified implementation on thousands of randomly generated problems

Conclusions & Future Work

Background:

- Proof logging for decision and optimization problems well-established
- No proof logging for enumeration

Conclusions & Future Work

Background:

- Proof logging for decision and optimization problems well-established
- No proof logging for enumeration

This Work:

- Proof logging of (projected) enumeration problems in the VeriPB
- Problem reformulations guaranteeing same (projected) solution count
- Formally verified checker for enumeration and count preserving reformulations

Conclusions & Future Work

Background:

- Proof logging for decision and optimization problems well-established
- No proof logging for enumeration

This Work:

- Proof logging of (projected) enumeration problems in the VeriPB
- Problem reformulations guaranteeing same (projected) solution count
- Formally verified checker for enumeration and count preserving reformulations

Future work:

- Enumeration of solution cubes (all solutions of cube satisfy constraints)
- Enumeration of optimal solutions
- Direct support for counting in VeriPB

Conclusions & Future Work

Background:

- Proof logging for decision and optimization problems well-established
- No proof logging for enumeration

This Work:

- Proof logging of (projected) enumeration problems in the VeriPB
- Problem reformulations guaranteeing same (projected) solution count
- Formally verified checker for enumeration and count preserving reformulations

Future work:

- Enumeration of solution cubes (all solutions of cube satisfy constraints)
- Enumeration of optimal solutions
- Direct support for counting in VeriPB

Thank you for your attention!

References I

[ACMS15] Rehan Abdul Aziz, Geoffrey Chu, Christian Muise, and Peter Stuckey.
#∃SAT: Projected model counting.
In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 121–137, Cham, 2015. Springer International Publishing.

[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.

[AMV22] Özgür Akgün, Martín Mereb, and Leandro Vendramin.
Enumeration of set-theoretic solutions to the yang-baxter equation.
Math. Comput., 91(335):1469–1481, 2022.

[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.

References II

[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*.

[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.

[BNAH25] Randal E Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn JH Heule.
Certified knowledge compilation with application to formally verified model counting.
Journal of Artificial Intelligence Research, 82:2057–2099, 2025.

[BTH25] Randal E. Bryant, Yong Kiam Tan, and Marijn J. H. Heule.
Certifying Projected Knowledge Compilation.
In Jeremias Berg and Jakob Nordström, editors, *28th International Conference on Theory and Applications of Satisfiability Testing (SAT 2025)*, volume 341 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:22, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

References III

[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.

[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.

[FHH21] Johannes K. Fichte, Markus Hecher, and Florim Hamiti.
The model counting competition 2020.
ACM J. Exp. Algorithmics, 26, October 2021.

[FHR22] Johannes Klaus Fichte, Markus Hecher, and Valentin Roland.
Proofs for propositional model counting.
In *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, pages 30:1–30:24, 2022.

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

References IV

[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.

[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.

[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.

[GS19] Graeme Gange and Peter Stuckey.
Certifying optimality in constraint programming.
Presentation at KTH Royal Institute of Technology, February 2019.

References V

[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.

[HHW13] 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.

[TS16] Takahisa Toda and Takehide Soh.
Implementing efficient all solutions sat solvers.
Journal of Experimental Algorithmics (JEA), 21:1–44, 2016.

[TTT06] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi.
The worst-case time complexity for generating all maximal cliques and computational experiments.
Theor. Comput. Sci., 363(1):28–42, 2006.

References VI

[ZM03] Lintao Zhang and S. Malik.
Validating sat solvers using an independent resolution-based checker: practical implementations and other applications.
In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 880–885, 2003.