Certified Symmetry and Dominance Breaking for Combinatorial Optimisation

Bart Bogaerts, **Stephan Gocht**, Ciaran McCreesh, Jakob Nordström

February 2022

Symmetry Breaking — Example

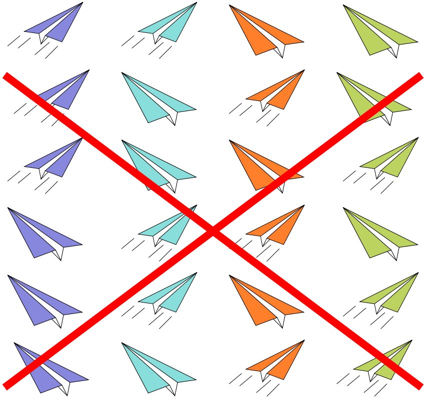
- airline scheduling: have 4 aircraft
  goal: assign which aircraft fly
- only need 2 (due to corona)
- aircraft are interchangeable
  (encoding will be symmetric)

Symmetry Breaking — Example

- airline scheduling: have 4 aircraft
  goal: assign which aircraft fly
- only need 2 (due to corona)
- aircraft are interchangeable
  (encoding will be symmetric)

Symmetry Breaking — Example

- airline scheduling: have 4 aircraft
  goal: assign which aircraft fly
- only need 2 (due to corona)
- aircraft are interchangeable
  (encoding will be symmetric)
- idea: make scheduling easier
  by removing symmetric assignments



Symmetry Breaking

manual and automatic techniques used across combinatorial optimisation paradigms

- constraint programming (CP) [GSVW14]
- Boolean satisfiability (SAT) [BHvMW21]
- mixed-integer programming (MIP) [AW13]

Symmetry Breaking

manual and automatic techniques used across combinatorial optimisation paradigms

- constraint programming (CP) [GSVW14]
- Boolean satisfiability (SAT) [BHvMW21]
- mixed-integer programming (MIP) [AW13]

How can we know we didn't remove too many assignments?

Software Verification — How to Ensure Software Behaves as Intended?

- Software testing
  - run collection of test cases to check if software behaves as intended
    - depends on quality of test cases, likely to miss non-trivial defects
    - can't show absence of bugs, only their presence

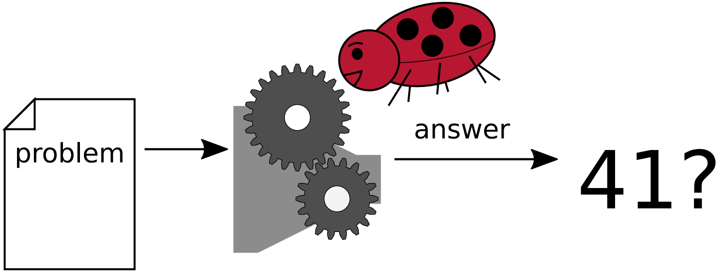
Software Verification — How to Ensure Software Behaves as Intended?

- Software testing
  - run collection of test cases to check if software behaves as intended
    - depends on quality of test cases, likely to miss non-trivial defects
    - can't show absence of bugs, only their presence
- Formal verification
  - formally verify that implementation adheres to specification on all possible inputs
    - out of reach for complex, performance-critical software

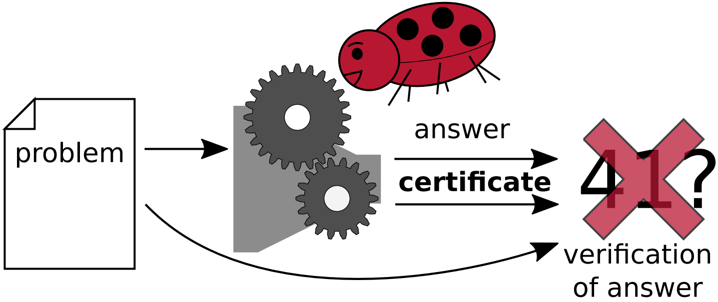
Software Verification — How to Ensure Software Behaves as Intended?

- Software testing
  - run collection of test cases to check if software behaves as intended
    - depends on quality of test cases, likely to miss non-trivial defects
    - can't show absence of bugs, only their presence
- Formal verification
  - formally verify that implementation adheres to specification on all possible inputs
    - out of reach for complex, performance-critical software
- Certifying algorithms, also known as proof logging (this talk)
  - let algorithm output answer and *proof* that answer is correct
  - proof: sequence of simple, efficiently machine-verifiable steps

Detecting Bugs with Certifying Algorithms

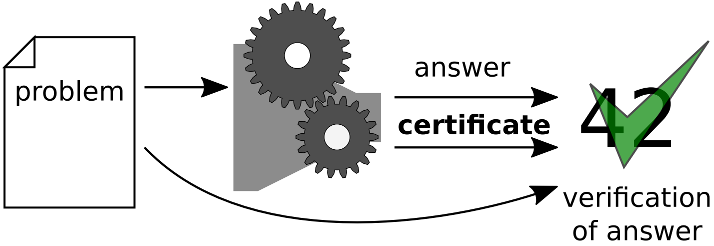


Detecting Bugs with Certifying Algorithms



- verification of answer with external tool can detect bugs

Guaranteeing Correctness with Certifying Algorithms



- successful verification of answer with external tool guarantees correct answer

Why Certifying Algorithms?

- while solving
  - increase trust in solution
  - detect hardware errors

Why Certifying Algorithms?

- while solving
  - increase trust in solution
  - detect hardware errors
- after solving
  - analyse certificate to understand and improve solving process
  - could use certificate to audit solution afterwards

Why Certifying Algorithms?

- while solving
  - increase trust in solution
  - detect hardware errors
- after solving
  - analyse certificate to understand and improve solving process
  - could use certificate to audit solution afterwards
- during development
  - simplifies testing: not necessary to know correct answer a priory
  - find bugs even if result is correct
  - locate first unsound step

Requirements for Certifying Algorithms

- certificate verification
  - should be efficiently machine-verifiable
  - ideally so simple that proof checker can be formally verified
  - want: simple, easy to verify steps / rules

Requirements for Certifying Algorithms

- certificate verification
  - should be efficiently machine-verifiable
  - ideally so simple that proof checker can be formally verified
  - want: simple, easy to verify steps / rules
- certificate production
  - should be easy to implement in any solver
  - should only incur small performance overhead
  - want: expressive rules for concise reasoning

Requirements for Certifying Algorithms

- certificate verification
  - should be efficiently machine-verifiable
  - ideally so simple that proof checker can be formally verified
  - want: simple, easy to verify steps / rules

- certificate production
  - should be easy to implement in any solver
  - should only incur small performance overhead
  - want: expressive rules for concise reasoning

But how?

SAT Solving — A Success Story for Certifying Algorithms ...

- SAT $=$ satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)

SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)
- proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CMS17], LRAT [CHH$^+$17]; DRAT [WHH14] has become standard

SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)
- proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CMS17], LRAT [CHH+17]; DRAT [WHH14] has become standard

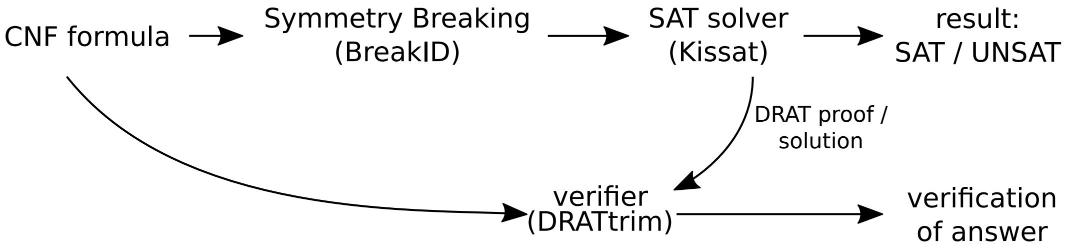
- Can we use SAT technology?

CNF formula → Symmetry Breaking (BreakID) → SAT solver (Kissat) → result: SAT / UNSAT

SAT Solving — A Success Story for Certifying Algorithms . . .

- SAT = satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)
- proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CMS17], LRAT [CHH+17]; DRAT [WHH14] has become standard

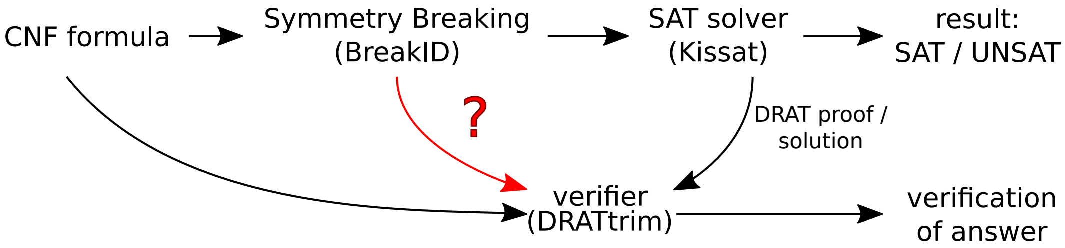
- Can we use SAT technology?



SAT Solving — A Success Story for Certifying Algorithms ...

- SAT = satisfiability of propositional formulas in conjunctive normal form (CNF)
- SAT competition requires solver to produce certificate (aka proof logging)
- proof formats such as RUP [GN03], TraceCheck [Bie06], GRIT [CMS17], LRAT [CHH+17]; DRAT [WHH14] has become standard

- Can we use SAT technology?



. . . But Need for Further Research

- proof logging for symmetry breaking
  - no implementation available
  - approach based on DRAT proposed [HHW15]
  - only for small symmetries which can interact only in simple ways

. . . But Need for Further Research

- proof logging for symmetry breaking
  - no implementation available
  - approach based on DRAT proposed [HHW15]
  - only for small symmetries which can interact only in simple ways

- without symmetry breaking $\Rightarrow$ exponential loss in reasoning power / performance

. . . But Need for Further Research

- proof logging for symmetry breaking
  - no implementation available
  - approach based on DRAT proposed [HHW15]
  - only for small symmetries which can interact only in simple ways
- without symmetry breaking $\Rightarrow$ exponential loss in reasoning power / performance
- DRAT cannot support symmetry breaking $\Rightarrow$ need to investigate other methods

. . . But Need for Further Research

- proof logging for symmetry breaking
  - no implementation available
  - approach based on DRAT proposed [HHW15]
  - only for small symmetries which can interact only in simple ways
- without symmetry breaking $\Rightarrow$ exponential loss in reasoning power / performance
- DRAT cannot support symmetry breaking $\Rightarrow$ need to investigate other methods
- not the only reason to look for other methods, what about
  - MaxSAT solving
  - constraint programming (CP)
  - mixed integer programming (MIP)
  - algebraic reasoning / Gröbner basis computations
  - pseudo-Boolean satisfiability and optimization

New Proof Systems are Being Developed

many new proof systems

- delete symmetry reverse unit propagation (DSRUP) [TD20]
- propagation redundancy (PR) [HKB17]
- branch and bound in integer programming [CGS17, EG21]
- practical polynomial calculus (PAC) [RBK18, KFB20, KFBK22]
- extensible RAT (FRAT) [BCH21]
- propagation redundancy for BDDs [BB21]
- **pseudo-Boolean proofs** [EGMN20, GN21, BGMN22]

High Level Idea of Pseudo-Boolean Proofs

- use pseudo-Boolean constraints (0-1 linear inequalities) to describe problem
  - e.g., $x_1 + x_2 + x_3 \geq 1$ or $2z + x_1 + x_2 + x_3 \geq 2$
  - solution is assignment satisfying all constraints
  - NP-complete $\Rightarrow$ very expressive, but in general difficult to find solution

High Level Idea of Pseudo-Boolean Proofs

- use pseudo-Boolean constraints (0-1 linear inequalities) to describe problem
  - e.g., $x_1 + x_2 + x_3 \geq 1$ or $2z + x_1 + x_2 + x_3 \geq 2$
  - solution is assignment satisfying all constraints
  - NP-complete $\Rightarrow$ very expressive, but in general difficult to find solution
- proof system is small set of rules that
  - are easy to verify
  - allow to add new constraints using previous constraints
  - guarantee that at least one (optimal) solution satisfies all constraints (given that original problem has solution)

High Level Idea of Pseudo-Boolean Proofs

- use pseudo-Boolean constraints (0-1 linear inequalities) to describe problem
  - e.g., $x_1 + x_2 + x_3 \geq 1$ or $2z + x_1 + x_2 + x_3 \geq 2$
  - solution is assignment satisfying all constraints
  - NP-complete $\Rightarrow$ very expressive, but in general difficult to find solution
- proof system is small set of rules that
  - are easy to verify
  - allow to add new constraints using previous constraints
  - guarantee that at least one (optimal) solution satisfies all constraints (given that original problem has solution)
- proof constructs sequence of constraints $D_1, D_2, D_3, \ldots, D_L$
  - each constraint is derived by rule in proof system
  - annotation can contain additional information necessary for efficient verification
  - proves there is no solution if D_L is $0 \geq 1$
  - proves optimality if D_L is bound on objective matching known solution

Our Approach

- use pseudo-Boolean proofs (PBP)
- reference implementation of verifier: VeriPB[1]
- **multi-purpose** format: proof logging for wide range of problems / algorithms
  - reasoning with 0-1 linear inequalities (by design)

[1] `https://gitlab.com/MIAOresearch/VeriPB`

Our Approach

- use pseudo-Boolean proofs (PBP)
- reference implementation of verifier: VeriPB[1]
- **multi-purpose** format: proof logging for wide range of problems / algorithms
  - reasoning with 0-1 linear inequalities (by design)
  - constraint programming, including all-different constraints [EGMN20, GMN22]

[1] `https://gitlab.com/MIAOresearch/VeriPB`

Our Approach

- use pseudo-Boolean proofs (PBP)
- reference implementation of verifier: VeriPB[1]
- **multi-purpose** format: proof logging for wide range of problems / algorithms
  - reasoning with 0-1 linear inequalities (by design)
  - constraint programming, including all-different constraints [EGMN20, GMN22]
  - subgraph isomorphism [GMN20]
  - clique and maximum common (connected) subgraph [GMM$^+$20]

[1] `https://gitlab.com/MIAOresearch/VeriPB`

Our Approach

- use pseudo-Boolean proofs (PBP)
- reference implementation of verifier: VeriPB[1]
- **multi-purpose** format: proof logging for wide range of problems / algorithms
  - reasoning with 0-1 linear inequalities (by design)
  - constraint programming, including all-different constraints [EGMN20, GMN22]
  - subgraph isomorphism [GMN20]
  - clique and maximum common (connected) subgraph [GMM+20]
  - SAT solving by generalizing DRAT [GN21]
  - parity/ XOR reasoning [GN21]
  - symmetry and dominance breaking (for SAT, PB, CP, clique) [BGMN22]

[1] `https://gitlab.com/MIAOresearch/VeriPB`

Our Approach

- use pseudo-Boolean proofs (PBP)
- reference implementation of verifier: VeriPB[1]
- **multi-purpose** format: proof logging for wide range of problems / algorithms
  - reasoning with 0-1 linear inequalities (by design)
  - constraint programming, including all-different constraints [EGMN20, GMN22]
  - subgraph isomorphism [GMN20]
  - clique and maximum common (connected) subgraph [GMM+20]
  - SAT solving by generalizing DRAT [GN21]
  - parity/ XOR reasoning [GN21]
  - symmetry and dominance breaking (for SAT, PB, CP, clique) [BGMN22]
  - pseudo-Boolean solving via translation to CNF [GMNO22]

[1] `https://gitlab.com/MIAOresearch/VeriPB`

Our Approach

- use pseudo-Boolean proofs (PBP)
- reference implementation of verifier: VeriPB[1]
- **multi-purpose** format: proof logging for wide range of problems / algorithms
  - reasoning with 0-1 linear inequalities (by design)
  - constraint programming, including all-different constraints [EGMN20, GMN22]
  - subgraph isomorphism [GMN20]
  - clique and maximum common (connected) subgraph [GMM+20]
  - SAT solving by generalizing DRAT [GN21]
  - parity/ XOR reasoning [GN21]
  - symmetry and dominance breaking (for SAT, PB, CP, clique) [BGMN22]
  - pseudo-Boolean solving via translation to CNF [GMNO22]

This Work

- proof logging for symmetry and dominance breaking
- applied to SAT, constraint programming and max clique solving
- support for optimization

[1] `https://gitlab.com/MIAOresearch/VeriPB`

Running Example

$$\begin{aligned} \text{min: } & 4x_1 + 2x_2 + x_3 \\ \text{s.t. } & x_1 + x_2 + x_3 \geq 2 \end{aligned}$$

- boolean variable x is 0 (false) or 1 (true) (e.g. $x_1 = 1$ means green aircraft flies)
- pseudo-Boolean constraint: linear inequality over variables
- formula F: set of constraints
- objective function f to be minimized

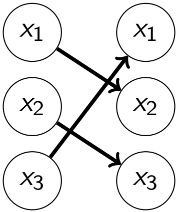
Goal: find assignment minimizing objective and satisfying all constraints

Background — Symmetric Formulas

- given permutation π
- formula F has (syntactic) symmetry if $F = F{\restriction_\pi}$

Background — Symmetric Formulas

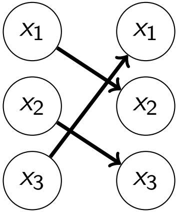
- given permutation π
- formula F has (syntactic) symmetry if $F = F{\restriction_\pi}$
- example:
  - let $\pi = \{\, x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \,\}$



Background — Symmetric Formulas

- given permutation π
- formula F has (syntactic) symmetry if $F = F_{\upharpoonright\pi}$
- example:
  - let $\pi = \{\, x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \,\}$

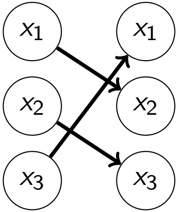
$$\begin{aligned} &(x_1 + x_2 + x_3 \geq 2)_{\upharpoonright\pi} \\ \equiv\ & x_2 + x_3 + x_1 \geq 2 \end{aligned}$$



Background — Symmetric Formulas

- given permutation π
- formula F has (syntactic) symmetry if $F = F{\restriction_\pi}$
- example:
  - let $\pi = \{\, x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \,\}$

$$
\begin{aligned}
&(x_1 + x_2 + x_3 \geq 2){\restriction_\pi} \\
\equiv\ & x_2 + x_3 + x_1 \geq 2
\end{aligned}
$$



  - constraint is same as before
    $\Rightarrow$ formula is symmetric (ignoring objective)

Background — Symmetry Breaking

- let $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$

Background — Symmetry Breaking

- let $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- add blocking constraint to remove symmetric assignments
- usually removing lexicographic larger assignments
  viewing $x_1x_2x_3$ as bitstring, e.g., $011 \preceq_{\text{lex}} 110$

Background — Symmetry Breaking

- let $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- add blocking constraint to remove symmetric assignments
- usually removing lexicographic larger assignments
  viewing $x_1 x_2 x_3$ as bitstring, e.g., $011 \preceq_{\text{lex}} 110$

$$\begin{aligned} \text{want to encode} \quad & x_1 x_2 x_3 \preceq_{\text{lex}} (x_1 x_2 x_3)\restriction_\pi \\ \text{same as} \quad & x_1 x_2 x_3 \preceq_{\text{lex}} \; x_2 x_3 x_1 \end{aligned}$$

Background — Symmetry Breaking

- let $\pi = \{\, x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \,\}$
- add blocking constraint to remove symmetric assignments
- usually removing lexicographic larger assignments
  viewing $x_1 x_2 x_3$ as bitstring, e.g., $011 \preceq_{\text{lex}} 110$

$$\begin{aligned} \text{want to encode} \quad & x_1 x_2 x_3 \preceq_{\text{lex}} (x_1 x_2 x_3)\restriction_{\pi} \\ \text{same as} \quad & x_1 x_2 x_3 \preceq_{\text{lex}} \; x_2 x_3 x_1 \end{aligned}$$

- alternative: view bitstring as binary number
- easy to encode using pseudo-Boolean constraint

$$4x_1 + 2x_2 + x_3 \leq 4x_2 + 2x_3 + x_1$$

Background — Symmetry Breaking

- let $\pi = \{\, x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \,\}$
- add blocking constraint to remove symmetric assignments
- usually removing lexicographic larger assignments
  viewing $x_1x_2x_3$ as bitstring, e.g., $011 \preceq_{\text{lex}} 110$

$$\begin{aligned} \text{want to encode} \quad & x_1x_2x_3 \preceq_{\text{lex}} (x_1x_2x_3){\restriction_\pi} \\ \text{same as} \quad & x_1x_2x_3 \preceq_{\text{lex}} \; x_2x_3x_1 \end{aligned}$$

- alternative: view bitstring as binary number
- easy to encode using pseudo-Boolean constraint

$$4x_1 + 2x_2 + x_3 \leq 4x_2 + 2x_3 + x_1$$

- can be simplified to $3x_1 - 2x_2 - x_3 \leq 0$

Output from Symmetry Breaking

- give formula $x_1 + x_2 + x_3 \geq 2$ to symmetry breaker (falsified by red assignments)

Truth Table

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

Output from Symmetry Breaking

- give formula $x_1 + x_2 + x_3 \geq 2$ to symmetry breaker (falsified by red assignments)
- output:
  - for permutation $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$

Truth Table

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

π

π

Output from Symmetry Breaking

- give formula $x_1 + x_2 + x_3 \geq 2$ to symmetry breaker (falsified by red assignments)
- output:
  - for permutation $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
  - (syntactic) symmetry detected (ignoring objective) $(x_1 + x_2 + x_3 \geq 2)_{\restriction \pi} = x_2 + x_3 + x_1 \geq 2$

Truth Table

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	π
4	1	0	0	
3	0	1	1	
5	1	0	1	π
6	1	1	0	
7	1	1	1	

Notation: $C_{\restriction \pi}$ substitutes variables in C as specified by π

Output from Symmetry Breaking

- give formula $x_1 + x_2 + x_3 \geq 2$ to symmetry breaker (falsified by red assignments)
- output:
  - for permutation $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
  - (syntactic) symmetry detected (ignoring objective) $(x_1 + x_2 + x_3 \geq 2)\restriction_\pi = x_2 + x_3 + x_1 \geq 2$
  - breaking constraint $3x_1 - 2x_2 - x_3 \leq 0$ (falsified by purple assignments)

Truth Table

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

π

π

Notation: $C\restriction_\pi$ substitutes variables in C as specified by π

Output from Symmetry Breaking

- give formula $x_1 + x_2 + x_3 \geq 2$ to symmetry breaker (falsified by red assignments)
- output:
  - for permutation $\pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
  - (syntactic) symmetry detected (ignoring objective) $(x_1 + x_2 + x_3 \geq 2)\restriction_\pi = x_2 + x_3 + x_1 \geq 2$
  - breaking constraint $3x_1 - 2x_2 - x_3 \leq 0$ (falsified by purple assignments)

How prove adding constraint is OK?

Truth Table

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

π

π

Notation: $C\restriction_\pi$ substitutes variables in C as specified by π

The Dominance Rule

Idea (From Dominance Breaking)

assignment is dominated if we find strictly better assignment $\Rightarrow$ can remove dominated assignments

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

The Dominance Rule

Idea (From Dominance Breaking)

assignment is dominated if we find strictly better assignment $\Rightarrow$ can remove dominated assignments

- allow to add constraint C, e.g, $3x_1 - 2x_2 - x_3 \leq 0$
- if for every ρ falsifying C but satisfying F (purple)
- we find ρ' that satisfies F and $f(\rho) > f(\rho')$

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

The Dominance Rule

Idea (From Dominance Breaking)

assignment is dominated if we find strictly better assignment $\Rightarrow$ can remove dominated assignments

- allow to add constraint C, e.g, $3x_1 - 2x_2 - x_3 \leq 0$
- if for every ρ falsifying C but satisfying F (purple)
- we find ρ' that satisfies F and $f(\rho) > f(\rho')$

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

Notation: $F \models F'$: satisfying assignment to F is also satisfying assignment to F'
In general not efficiently verifiable, however can provide explicit proof.

The Dominance Rule

Idea (From Dominance Breaking)

assignment is dominated if we find strictly better assignment $\Rightarrow$ can remove dominated assignments

- allow to add constraint C, e.g, $3x_1 - 2x_2 - x_3 \leq 0$
- if for every ρ falsifying C but satisfying F (purple)
- we find ρ' that satisfies F and $f(\rho) > f(\rho')$

Dominance Rule (simplified)

Can derive constraint C from formula F if a *witnessing* substitution ω is provided such that

$$F \cup \{\neg C\} \models F{\restriction_\omega} \cup \{f > f{\restriction_\omega}\}$$

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

Notation: $F \models F'$: satisfying assignment to F is also satisfying assignment to F'
In general not efficiently verifiable, however can provide explicit proof.

The Dominance Rule — Example

- want to add $C: 3x_1 - 2x_2 - x_3 \leq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- **have to prove:**
  $F \cup \{ \neg C \} \models F\restriction_\omega \cup \{ f > f\restriction_\omega \}$, i.e.,

Dominance Rule (simplified)

Add C if there is *witnessing* substitution ω s.t.

$$F \cup \{ \neg C \} \models F\restriction_\omega \cup \{ f > f\restriction_\omega \}$$

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

ω

ω

The Dominance Rule — Example

- want to add $C: 3x_1 - 2x_2 - x_3 \leq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- **have to prove:** $F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$, i.e.,
- given F: $x_1 + x_2 + x_3 \geq 2$ and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$

Dominance Rule (simplified)

Add C if there is *witnessing* substitution ω s.t.

$$F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$$

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

ω

ω

The Dominance Rule — Example

- want to add $C: 3x_1 - 2x_2 - x_3 \leq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- **have to prove:**
  $F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$, i.e.,
- given F: $x_1 + x_2 + x_3 \geq 2$
  and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$
- derive $$F_{\restriction\omega} \equiv (x_1 + x_2 + x_3 \geq 2)_{\restriction\omega} \equiv x_2 + x_3 + x_1 \geq 2$$

Dominance Rule (simplified)

Add C if there is *witnessing* substitution ω s.t.

$$F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$$

objective value	x_1	x_2	x_3	
0	0	0	0	
1	0	0	1	
2	0	1	0	
4	1	0	0	
3	0	1	1	
5	1	0	1	ω
6	1	1	0	ω
7	1	1	1	

The Dominance Rule — Example

- want to add $C: 3x_1 - 2x_2 - x_3 \leq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- **have to prove:**
  $F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$, i.e.,
- given F: $x_1 + x_2 + x_3 \geq 2$
  and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$
- derive
$$\begin{aligned} F_{\restriction\omega} &\equiv (x_1 + x_2 + x_3 \geq 2)_{\restriction\omega} \\ &\equiv x_2 + x_3 + x_1 \geq 2 \end{aligned}$$

Dominance Rule (simplified)

Add C if there is *witnessing* substitution ω s.t.

$$F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$$

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

ω
ω

The Dominance Rule — Example

- want to add $C\colon 3x_1 - 2x_2 - x_3 \le 0$
- choose $\omega = \pi = \{\, x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \,\}$
- **have to prove:**
  $F \cup \{\, \neg C \,\} \models F{\restriction_\omega} \cup \{\, f > f{\restriction_\omega} \,\}$, i.e.,
- given F: $x_1 + x_2 + x_3 \ge 2$
  and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$
- derive $F{\restriction_\omega} \equiv (x_1 + x_2 + x_3 \ge 2){\restriction_\omega}$
  $\equiv x_2 + x_3 + x_1 \ge 2$
- and
  $f > f{\restriction_\omega} \equiv 4x_1 + 2x_2 + x_3 > (4x_1 + 2x_2 + x_3){\restriction_\omega}$
  $\equiv 4x_1 + 2x_2 + x_3 > 4x_2 + 2x_3 + x_1$
  $\equiv 3x_1 - 2x_2 - x_3 > 0$

Dominance Rule (simplified)

Add C if there is *witnessing* substitution ω s.t.

$$F \cup \{\, \neg C \,\} \models F{\restriction_\omega} \cup \{\, f > f{\restriction_\omega} \,\}$$

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

ω ω

The Dominance Rule — Example

- want to add $C\colon 3x_1 - 2x_2 - x_3 \leq 0$
- choose $\omega = \pi = \{ x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_1 \}$
- **have to prove:**
  $F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$, i.e.,
- given F: $x_1 + x_2 + x_3 \geq 2$
  and $\neg C$: $3x_1 - 2x_2 - x_3 > 0$
- derive
  $$F_{\restriction\omega} \equiv (x_1 + x_2 + x_3 \geq 2)_{\restriction\omega} \equiv x_2 + x_3 + x_1 \geq 2$$
- and
  $$f > f_{\restriction\omega} \equiv 4x_1 + 2x_2 + x_3 > (4x_1 + 2x_2 + x_3)_{\restriction\omega} \equiv 4x_1 + 2x_2 + x_3 > 4x_2 + 2x_3 + x_1 \equiv 3x_1 - 2x_2 - x_3 > 0$$

Dominance Rule (simplified)

Add C if there is *witnessing* substitution ω s.t.

$$F \cup \{ \neg C \} \models F_{\restriction\omega} \cup \{ f > f_{\restriction\omega} \}$$

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

ω

ω

The Dominance Rule — Further Remarks

Dominance Rule (simplified)

Can derive constraint C from formula F if a *witnessing* substitution ω is provided such that

$$F \cup \{\neg C\} \models F{\restriction_\omega} \cup \{f > f{\restriction_\omega}\}$$

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

ω

ω

The Dominance Rule — Further Remarks

Dominance Rule (simplified)

Can derive constraint C from formula F if a *witnessing* substitution ω is provided such that

$$F \cup \{\neg C\} \models F{\restriction_\omega} \cup \{f > f{\restriction_\omega}\}$$

- "$\models$" replaced by **efficiently machine-verifiable** proof system (cutting planes)

objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

ω

ω

The Dominance Rule — Further Remarks

Dominance Rule (simplified)

Can derive constraint C from formula F if a *witnessing* substitution ω is provided such that

$$F \cup \{\neg C\} \models F{\restriction_\omega} \cup \{f > f{\restriction_\omega}\}$$

- "$\models$" replaced by **efficiently machine-verifiable** proof system (cutting planes)
- in paper: any strict order instead of $f > f{\restriction_\omega}$

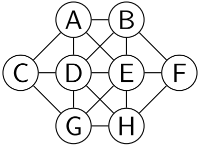
objective value	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

ω

ω

Supported Applications – Symmetry Breaking

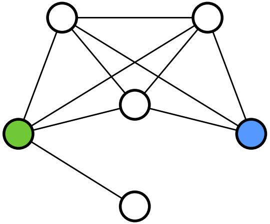
- SAT
  challenge: translate breaking constraint from PB to CNF
- Constraint Programming
  challenge: integer domains instead of 0-1



example: The Crystal Maze puzzle. Place numbers 1 to 8 without repetition, so that adjacent circles do not have consecutive numbers. Puzzle can be mirrored horizontally. Without loss of generality number in A smaller than number in G.

Supported Applications – Dominance Breaking

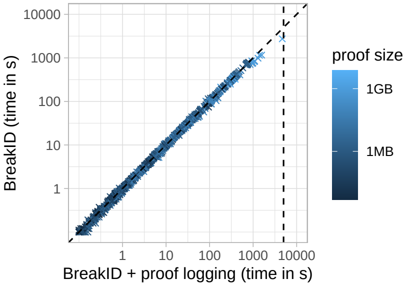
- maximum clique solving (find largest fully connected component)
  challenge: lazy breaking

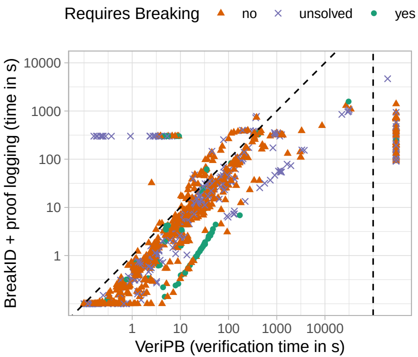


example: consider green but not blue node (every neighbour of blue is also neighbour of green)

Experiments

- evaluated on SAT competition benchmarks
- used BreakID[2] to find and break symmetries





- proof logging overhead negligible
- verification at most 20 times slower than solving for 95 % of instance

[2] https://bitbucket.org/krr/breakid/

Future Work

understand power of dominance rule:

- can we simulate dominance rule with redundance rule or extended cutting planes?

Future Work

understand power of dominance rule:

- can we simulate dominance rule with redundance rule or extended cutting planes?

improve performance:

- binary format / on-the-fly compression
- trimming proof while verifying (as for DRAT [HHW13])

Future Work

understand power of dominance rule:

- can we simulate dominance rule with redundance rule or extended cutting planes?

improve performance:

- binary format / on-the-fly compression
- trimming proof while verifying (as for DRAT [HHW13])

increase trustworthiness:

- formally verified verifier

Future Work

understand power of dominance rule:

- can we simulate dominance rule with redundance rule or extended cutting planes?

improve performance:

- binary format / on-the-fly compression
- trimming proof while verifying (as for DRAT [HHW13])

increase trustworthiness:

- formally verified verifier

proof logging for more algorithms and problems:

- symmetric explanation learning [DBB17]
- MaxSAT
- more propagators in constraint programming
- integer programming

Conclusion

- proof logging is well-established standard for SAT solving
- not sufficient for all techniques used in SAT (e.g. symmetry breaking)

3

4

Conclusion

- proof logging is well-established standard for SAT solving
- not sufficient for all techniques used in SAT (e.g. symmetry breaking)

our work: proof logging for symmetry breaking (BreakID[3]) + verification (VeriPB[4])

- simple to implement + efficient proof checking
- fully evaluated for symmetry breaking on SAT competition benchmarks
- proof of concept for
  - symmetry breaking in constraint programming
  - dynamic dominance breaking for maximum clique

[3] `https://bitbucket.org/krr/breakid/`
[4] `https://gitlab.com/MIAOresearch/VeriPB`

Conclusion

- proof logging is well-established standard for SAT solving
- not sufficient for all techniques used in SAT (e.g. symmetry breaking)

our work: proof logging for symmetry breaking (BreakID[3]) + verification (VeriPB[4])

- simple to implement + efficient proof checking
- fully evaluated for symmetry breaking on SAT competition benchmarks
- proof of concept for
  - symmetry breaking in constraint programming
  - dynamic dominance breaking for maximum clique

future work: understand power of dominance rule, improve performance, increase trustworthiness, proof logging for more algorithms and problems

[3] `https://bitbucket.org/krr/breakid/`
[4] `https://gitlab.com/MIAOresearch/VeriPB`

Strengthening Rules (simplified)

- for formula F, objective f, and sequence of derived constraints $D_1, D_2, \ldots$
- let G_i, be set of constraints added so far ($G_i = F \cup \{ D_1, \ldots, D_{i-1} \}$)
- redundance based strengthening:
  (generalize redundancy from SAT [HKB17, BT19] to PB and optimization)

$$\frac{G_i \cup \{ \neg D_i \} \models (G_i \cup D_i)_{\restriction\omega} \cup \{ f_{\restriction\omega} \leq f \}}{D_i}$$

- dominance based strengthening:

$$\frac{G_i \cup \{ \neg D_i \} \models F_{\restriction\omega} \cup \{ f_{\restriction\omega} < f \}}{D_i}$$

- rules are annotated by:
  - used substitution ω
  - explicit proof for "$\models$"

References I

[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.

[BB21] Lee A. Barnett and Armin Biere.
Non-clausal Redundancy Properties.
In André Platzer and Geoff Sutcliffe, editors, *Proceedings of the 28th International Conference on Automated Deduction (CADE 28)*, volume 12699 of *Lecture Notes in Computer Science*, pages 252–272, 2021.

[BCH21] Seulkee Baek, Mario Carneiro, and Marijn J. H. Heule.
A Flexible Proof Format for SAT Solver-Elaborator Communication.
In *Proceedings of the 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '21)*, volume 12651 of *Lecture Notes in Computer Science*, pages 59–75. Springer, March-April 2021.

[BGMN22] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström.
Certified Symmetry and Dominance Breaking for Combinatorial Optimisation.
In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI '22)*, February 2022. To appear.

References II

[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.

[Bie06] Armin Biere.
TraceCheck.
`http://fmv.jku.at/tracecheck/`, 2006.

[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.

[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 III

[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.

[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.

[DBB17] Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe.
Symmetric Explanation Learning: Effective Dynamic Symmetry Handling for SAT.
In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 83–100. Springer, August 2017.

References IV

[EG21] Leon Eifler and Ambros Gleixner.
A Computational Status Update for Exact Rational Mixed Integer Programming.
In *Proceedings of the 22nd International Conference on Integer Programming and Combinatorial Optimization (IPCO '21)*, volume 12707 of *Lecture Notes in Computer Science*, pages 163–177. Springer, May 2021.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström.
Justifying All Differences Using Pseudo-Boolean Reasoning.
In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '20)*, volume 34, pages 1486–1494. AAAI Press, 2020.

[GMM+20] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble.
Certifying Solvers for Clique and Maximum Common (Connected) Subgraph Problems.
In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, 2020.

[GMN20] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström.
Subgraph Isomorphism Meets Cutting Planes: Solving With Certified Solutions.
In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, (IJCAI '20)*, pages 1134–1140, 2020.

References V

[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)*, 2022.

[GMNO22] Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel.
Certified CNF Translations for Pseudo-Boolean Solving.
In *Proceedings of the 25nd International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, 2022.

[GN03] Evgueni Goldberg and Yakov Novikov.
Verification of Proofs of Unsatisfiability for CNF Formulas.
In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, pages 886–891, March 2003.

[GN21] Stephan Gocht and Jakob Nordström.
Certifying Parity Reasoning Efficiently Using Pseudo-Boolean Proofs.
In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '21)*, volume 35, pages 3768–3777, 2021.

[GSVW14] Maria Garcia de la Banda, Peter J. Stuckey, Pascal Van Hentenryck, and Mark Wallace.
The future of optimization technology.
Constraints, 19(2):126–138, April 2014.

References VI

[HHW13] 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.

[HHW15] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler.
Expressing Symmetry Breaking in DRAT Proofs.
In *Proceedings of the 25th International Conference on Automated Deduction (CADE-25)*, volume 9195 of *Lecture Notes in Computer Science*, pages 591–606. Springer, August 2015.

[HKB17] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere.
Short Proofs Without New Variables.
In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 130–147. Springer, August 2017.

[KFB20] Daniela Kaufmann, Mathias Fleury, and Armin Biere.
The Proof Checkers Pacheck and Pastèque for the Practical Algebraic Calculus.
In *Proceedings of Formal Methods in Computer Aided Design, FMCAD 2020*, pages 264–269, 2020.

References VII

[KFBK22] Daniela Kaufmann, Mathias Fleury, Armin Biere, and Manuel Kauers.
Practical algebraic calculus and Nullstellensatz with the checkers Pacheck and Pastèque and Nuss-Checker.
Formal Methods in System Design, 2022.

[RBK18] Daniela Ritirc, Armin Biere, and Manuel Kauers.
A practical polynomial calculus for arithmetic circuit verification.
In *Proceedings of the 3rd International Workshop on Satisfiability Checking and Symbolic Computation (SC2'18)*, pages 61–76, 2018.

[TD20] Rodrigue Konan Tchinda and Clémentin Tayou Djamégni.
On certifying the UNSAT result of dynamic symmetry-handling-based SAT solvers.
Constraints, 25(3-4):251–279, 2020.

[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.