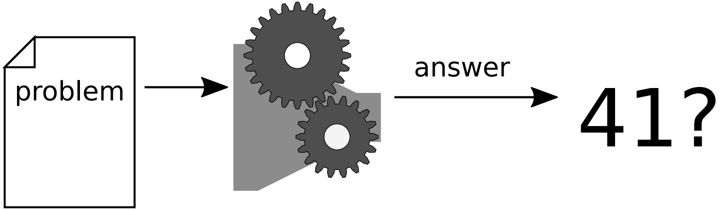
Certifying Correctness for Combinatorial Algorithms by Using Pseudo-Boolean Reasoning

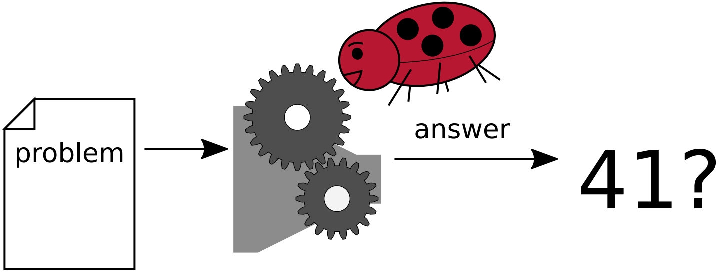
Stephan Gocht

1st June 2022

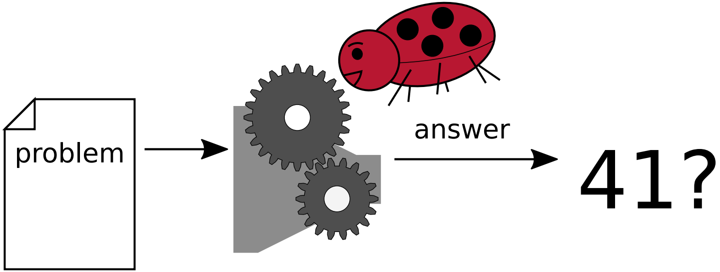
Do you trust your computer?



Do you trust your computer?



Do you trust your computer?



- what if answer is used for high-stakes decision?
  - e.g., combinatorial auction, kidney exchange program

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

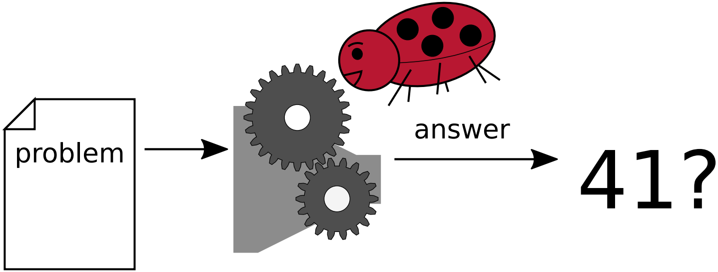
Software Verification — How to ensure software behaves as intended?

- 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

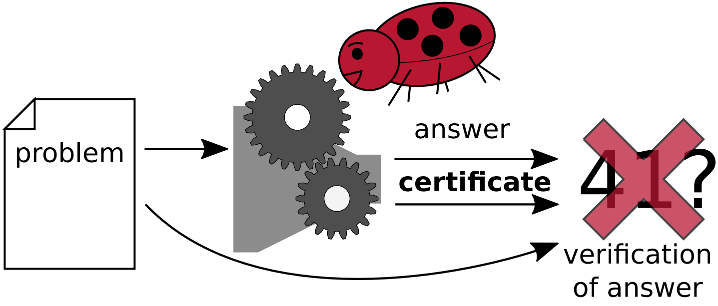
Software Verification — How to ensure software behaves as intended?

- 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

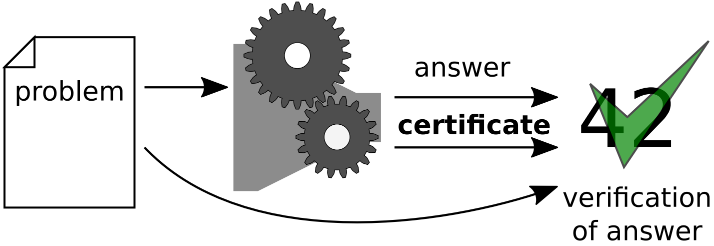


Detecting Bugs with Certifying Algorithms



- 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

. . . But Need for Further Research

- some SAT techniques don't have efficient DRAT proof logging
  - parity reasoning
  - symmetry breaking
  - symmetric explanation learning

. . . But Need for Further Research

- some SAT techniques don't have efficient DRAT proof logging
  - parity reasoning
  - symmetry breaking
  - symmetric explanation learning

- not using these techniques $\Rightarrow$ exponential loss in reasoning power / performance

. . . But Need for Further Research

- some SAT techniques don't have efficient DRAT proof logging
  - parity reasoning
  - symmetry breaking
  - symmetric explanation learning

- not using these techniques $\Rightarrow$ exponential loss in reasoning power / performance

- How about practical proof logging for other solving paradigms?
  - MaxSAT solving
  - constraint programming (CP)
  - mixed integer programming (MIP)
  - algebraic reasoning / Gröbner basis computations
  - pseudo-Boolean satisfiability and optimization

... But Need for Further Research

- some SAT techniques don't have efficient DRAT proof logging
  - parity reasoning
  - symmetry breaking
  - symmetric explanation learning
- not using these techniques $\Rightarrow$ exponential loss in reasoning power / performance
- How about practical proof logging for other solving paradigms?
  - MaxSAT solving
  - constraint programming (CP)
  - mixed integer programming (MIP)
  - algebraic reasoning / Gröbner basis computations
  - pseudo-Boolean satisfiability and optimization

Need to look beyond DRAT!

New Proof Systems are Being Developed

many new proof systems

- 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]
- Max-SAT resolution [PCH21]
- **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
- rest of this talk will explain and refine these concepts

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`

Running Example — Matching

- bipartite graph $G = (U \cup V, E)$
- find maximum matching $M \subseteq E$
- such that no node is incident to two edges in M

Basics — Pseudo-Boolean Problems

- Boolean variable x is 0 (false) or 1 (true)
- Literal: x or its negation $\overline{x} = 1 - x$
- pseudo-Boolean constraint: linear inequality over literals
  e.g., $\overline{x_1} + \overline{x_2} \geq 1$ or $x_1 + 2x_2 + \overline{x_3} \geq 2$
- formula F: set of constraints
- objective function f to be minimized
- Clause: at-least-one constraint, e.g., $\overline{x_1} + \overline{x_2} \geq 1$
- Contradiction: $\perp$ or $0 \geq 1$ is constraint that can't be satisfied

Goal: find assignment minimizing objective and satisfying all constraints

2

Literal Axioms

$$\frac{}{x \geq 0} \qquad \frac{}{\overline{x} \geq 0}$$

- can add variable bound
- rule is annotated by literal

3

Addition Rule

$$\frac{\overline{x}_1 + \overline{x}_2 \geq 1 \qquad \overline{x}_2 + \overline{x}_3 \geq 1}{\overline{x}_1 + 2\overline{x}_2 + \overline{x}_3 \geq 2}$$

- can add two pseudo-Boolean constraints
- rule is annotated by (reference to) the constraints to be added

4

Multiplication Rule

$$\frac{\overline{x}_1 + \overline{x}_2 \geq 1}{2\overline{x}_1 + 2\overline{x}_2 \geq 2}$$

- can multiply constraint by positive number
- rule is annotated by (reference to) the constraint and used factor

Division Rule

$$\frac{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_2 \geq 3}{\overline{x}_1 + \overline{x}_2 + \overline{x}_2 \geq 2}$$

- can divide constraint by positive number and round up
- rule is annotated by (reference to) the constraint and used divisor

- rules so far are known as the cutting planes proof system [CCT87]

5

Saturation Rule

$$\frac{4\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_2 \geq 3}{3\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_2 \geq 3}$$

- can reduce too large coefficients (assuming all coefficients are positive)
- rule is annotated by (reference to) the constraint

Basics — Manipulating Constraints

- can negate constraint

$$C: \quad x + y + z \geq 2$$
$$\neg C: \quad \overline{x} + \overline{y} + \overline{z} \geq 2$$

Basics — Manipulating Constraints

- can negate constraint

$$C: \quad x + y + z \geq 2$$
$$\neg C: \quad \overline{x} + \overline{y} + \overline{z} \geq 2$$

- substitution ω replaces variables by literals or $0, 1$

$$\omega = \{ x \mapsto z, y \mapsto 1 \}$$
$$C{\restriction_\omega}: \quad z + 1 + z \geq 2$$
$$2z \geq 1$$

Basics — Manipulating Constraints

- can negate constraint

$$C: \quad x + y + z \geq 2$$
$$\neg C: \quad \overline{x} + \overline{y} + \overline{z} \geq 2$$

- substitution ω replaces variables by literals or $0, 1$

$$\omega = \{\, x \mapsto z, y \mapsto 1 \,\}$$
$$C_{\restriction\omega}: \quad z + 1 + z \geq 2$$
$$2z \geq 1$$

- (total) assignment ρ is substitution setting all variables to 0 or 1

$$\rho = \{\, x \mapsto 1, y \mapsto 1, z \mapsto 0 \,\}$$
$$C_{\restriction\rho}: \quad 1 + 1 + 0 \geq 2$$

Basics — Manipulating Constraints

- can negate constraint

$$\begin{aligned} C:&\quad x+y+z\geq 2\\ \neg C:&\quad \overline{x}+\overline{y}+\overline{z}\geq 2\end{aligned}$$

- substitution ω replaces variables by literals or $0,1$

$$\omega = \{\, x \mapsto z, y \mapsto 1 \,\}$$

$$\begin{aligned} C_{\restriction\omega}:&\quad z+1+z\geq 2\\ &\quad 2z\geq 1\end{aligned}$$

- (total) assignment ρ is substitution setting all variables to 0 or 1

$$\rho = \{\, x \mapsto 1, y \mapsto 1, z \mapsto 0 \,\}$$

$$C_{\restriction\rho}:\quad 1+1+0\geq 2$$

- compose substitutions $\rho \circ \omega(x) = \rho(\omega(x))$

Basics — Manipulating Constraints

- can negate constraint

$$\begin{aligned} C: &\quad x + y + z \geq 2 \\ \neg C: &\quad \overline{x} + \overline{y} + \overline{z} \geq 2 \end{aligned}$$

- substitution ω replaces variables by literals or $0, 1$

$$\omega = \{\, x \mapsto z, y \mapsto 1 \,\}$$

$$\begin{aligned} C_{\restriction\omega}: &\quad z + 1 + z \geq 2 \\ &\quad 2z \geq 1 \end{aligned}$$

- (total) assignment ρ is substitution setting all variables to 0 or 1

$$\rho = \{\, x \mapsto 1, y \mapsto 1, z \mapsto 0 \,\}$$

$$C_{\restriction\rho}: \quad 1 + 1 + 0 \geq 2$$

- compose substitutions $\rho \circ \omega(x) = \rho(\omega(x))$
- Implication: $F \models C$ if every assignment satisfying F also satisfies C

Constraints that Remove Solutions

- so far, any solution satisfying F also satisfies added constraints (rules are implicational)
- however, only need to guarantee that solutions with minimal objective f remain

6

Constraints that Remove Solutions

- so far, any solution satisfying F also satisfies added constraints (rules are implicational)
- however, only need to guarantee that solutions with minimal objective f remain

6

- initial idea:
  - assume we want to add C
  - but there is assignment α satisfying F but falsifying C

Constraints that Remove Solutions

- so far, any solution satisfying F also satisfies added constraints (rules are implicational)
- however, only need to guarantee that solutions with minimal objective f remain

6

- initial idea:
  - assume we want to add C
  - but there is assignment α satisfying F but falsifying C
  - if we find assignment α' satisfying F and $f_{\restriction\alpha'} < f_{\restriction\alpha}$

Constraints that Remove Solutions

- so far, any solution satisfying F also satisfies added constraints (rules are implicational)
- however, only need to guarantee that solutions with minimal objective f remain

✎ 6

- initial idea:
  - assume we want to add C
  - but there is assignment α satisfying F but falsifying C
  - if we find assignment α' satisfying F and $f_{\upharpoonright\alpha'} < f_{\upharpoonright\alpha}$
  - then α wasn't optimal $\rightarrow$ OK that α falsifies C
  - save to add C if for all α falsifying C we find such an α'

Verifying Constraints that Remove Solutions

- initial idea: add C if for all α falsifying C there is α' satisfying F and $f_{\upharpoonright\alpha'} < f_{\upharpoonright\alpha}$
- problem:
  - needs to be efficiently verifiable
  - listing all α' would be prohibitive

Verifying Constraints that Remove Solutions

- initial idea: add C if for all α falsifying C there is α' satisfying F and $f_{\restriction\alpha'} < f_{\restriction\alpha}$
- problem:
  - needs to be efficiently verifiable
  - listing all α' would be prohibitive
- solution: only provide instruction (substitution ω) how to alter α, check that

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

Verifying Constraints that Remove Solutions

- initial idea: add C if for all α falsifying C there is α' satisfying F and $f_{\restriction \alpha'} < f_{\restriction \alpha}$
- problem:
  - needs to be efficiently verifiable
  - listing all α' would be prohibitive
- solution: only provide instruction (substitution ω) how to alter α, check that

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

  - assume α satisfies $F \cup \{\, \neg C \,\}$ (i.e., α satisfies F and falsifies C)
  - by implication above, α satisfies $F_{\restriction \omega} \cup \{\, f_{\restriction \omega} < f \,\}$
  - hence $\alpha' = \alpha \circ \omega$ satisfies F and has better objective value:

$$\begin{aligned} & (F_{\restriction \omega} \cup \{f_{\restriction \omega} < f\})_{\restriction \alpha} \\ = \; & F_{\restriction \alpha \circ \omega} \cup \{f_{\restriction \alpha \circ \omega} < f_{\restriction \alpha}\} \\ = \; & F_{\restriction \alpha'} \cup \{f_{\restriction \alpha'} < f_{\restriction \alpha}\} \end{aligned}$$

Verifying the Condition

- only provide instruction (substitution ω) how to alter α, check that

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

- problem: implication hard to check in general

Verifying the Condition

- only provide instruction (substitution ω) how to alter α, check that

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

- problem: implication hard to check in general
- solution: provide proof (using previous rules), showing

$$F \cup \{ \neg C, \neg D \} \models \bot \text{ for } D \in F_{\restriction\omega} \cup \{ f_{\restriction\omega} < f \}$$

Dominance Rule (simplified)

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

- rule is annotated by:
  - used substitution ω
  - for each $D \in F{\restriction_\omega} \cup \{f{\restriction_\omega} < f\}$ a proof showing $F \cup \{\neg C, \neg D\} \models \bot$

6

Redundance Rule (simplified)

- idea: (generalize redundancy from SAT [HKB17, BT19] to PB and optimization)
  - don't need to improve objective strictly
  - sufficient if one optimal solution remains

7

Redundance Rule (simplified)

- idea: (generalize redundancy from SAT [HKB17, BT19] to PB and optimization)
  - don't need to improve objective strictly
  - sufficient if one optimal solution remains
  - let G_i, be set of constraints added so far ($G_i = F \cup \{ D_1, \ldots, D_{i-1} \}$)

7

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

- rule is annotated by:
  - used substitution ω
  - for each $C \in (G_i \cup D_i)_{\restriction\omega} \cup \{ f_{\restriction\omega} \leq f \}$ a proof showing $G_i \cup \{ \neg D_i, \neg C \} \models \bot$

Proving Soundness of Proof System

Remember:

- rules need to guarantee that at least one (optimal) solution satisfies all constraints (given that original problem has solution)

Proving Soundness of Proof System

Remember:

- rules need to guarantee that at least one (optimal) solution satisfies all constraints (given that original problem has solution)

Rules presented so far maintain invariant:

- if there is optimal assignment ρ satisfying $F \cup \{ D_1, \ldots, D_{i-1} \}$,
- then there is assignment ρ' satisfying $F \cup \{ D_1, \ldots, D_i \}$,
- and $f_{\upharpoonright\rho'} = f_{\upharpoonright\rho}$

8

Proving Soundness of Proof System

Remember:

- rules need to guarantee that at least one (optimal) solution satisfies all constraints (given that original problem has solution)

Rules presented so far maintain invariant:

- if there is optimal assignment ρ satisfying $F \cup \{ D_1, \ldots, D_{i-1} \}$,
- then there is assignment ρ' satisfying $F \cup \{ D_1, \ldots, D_i \}$,
- and $f_{\restriction \rho'} = f_{\restriction \rho}$

8

Rule for Objective Bound Update

idea:

- incremental solver finds solution ρ to F
- now only looking for better solution
- finished if no better solution can be found

Rule for Objective Bound Update

idea:

- incremental solver finds solution ρ to F
- now only looking for better solution
- finished if no better solution can be found

$$\frac{\rho \text{ satisfies } F}{f < f\restriction_\rho}$$

rule is annotated by:

- solution ρ

9

Rule for Objective Bound Update

idea:

- incremental solver finds solution ρ to F
- now only looking for better solution
- finished if no better solution can be found

$$\frac{\rho \text{ satisfies } F}{f < f_{\restriction\rho}}$$

rule is annotated by:

- solution ρ

9

note:

- can terminate all proofs with contradiction $(0 \geq 1)$

Deletion Rule

deleting constraints . . .

- important for performance and memory efficiency
- only makes problem more satisfiable
  (except in connection with dominance — explanation in second part)

Dealing with Lazy Programmers ;-)

- goal: proof system should be easy to use
- problem: often "obvious" that adding constraint is OK, but tedious to write down
- solution: let verifier take care of "obvious" cases

Omitting Obvious Steps

from dominance rule:

- for each $D \in F_{\upharpoonright\omega} \cup \{ f_{\upharpoonright\omega} < f \}$ a proof showing $F \cup \{ \neg C, \neg D \} \models \bot$

can omitt proof if

- $\neg D = \bot$
- $D \in F$ (because $\neg D + D = \bot$)
- $\neg C + \neg D = \bot$

Reverse Unit Propagation [GN03, Van08]

- assume we have

$$C_1 : x + y >= 1$$
$$C_2 : \overline{y} + z >= 1$$

- and want to add

$$C_3 : x + z >= 1$$

Reverse Unit Propagation [GN03, Van08]

- assume we have

$$C_1 : x + y >= 1$$
$$C_2 : \overline{y} + z >= 1$$

- and want to add

$$C_3 : x + z >= 1$$

- simply claim there is no solution satisfying C_1, C_2 but falsifying C_3

Reverse Unit Propagation [GN03, Van08]

- assume we have

$$C_1 : x + y >= 1$$
$$C_2 : \overline{y} + z >= 1$$

- and want to add

$$C_3 : x + z >= 1$$

- simply claim there is no solution satisfying C_1, C_2 but falsifying C_3
- easy to check by propagation (setting forced variables):
  - C_3 only false if $\rho(x) = \rho(z) = 0$

Reverse Unit Propagation [GN03, Van08]

- assume we have

$$C_1 : x + y >= 1$$
$$C_2 : \overline{y} + z >= 1$$

- and want to add

$$C_3 : x + z >= 1$$

- simply claim there is no solution satisfying C_1, C_2 but falsifying C_3
- easy to check by propagation (setting forced variables):
  - C_3 only false if $\rho(x) = \rho(z) = 0$
  - but then C_1 only true if $\rho(y) = 1$

Reverse Unit Propagation [GN03, Van08]

- assume we have

$$C_1 : x + y >= 1$$
$$C_2 : \overline{y} + z >= 1$$

- and want to add

$$C_3 : x + z >= 1$$

- simply claim there is no solution satisfying C_1, C_2 but falsifying C_3
- easy to check by propagation (setting forced variables):
  - C_3 only false if $\rho(x) = \rho(z) = 0$
  - but then C_1 only true if $\rho(y) = 1$
  - but now C_2 falsified by ρ

Reverse Unit Propagation [GN03, Van08]

- assume we have

$$C_1 : x + y >= 1$$
$$C_2 : \overline{y} + z >= 1$$

- and want to add

$$C_3 : x + z >= 1$$

- simply claim there is no solution satisfying C_1, C_2 but falsifying C_3
- easy to check by propagation (setting forced variables):
  - C_3 only false if $\rho(x) = \rho(z) = 0$
  - but then C_1 only true if $\rho(y) = 1$
  - but now C_2 falsified by ρ
  - $\Rightarrow$ no assignment ρ satisfies C_1, C_2 and $\neg C_3$

Future Work

improve performance:

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

Future Work

improve performance:

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

increase trustworthiness:

- formally verified verifier

Future Work

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:

- MaxSAT (optimization for SAT)
- more propagators in constraint programming
- symmetric explanation learning
- integer programming

Conclusion

- proof logging is well-established standard for SAT solving
- so far not usable for
  - some techniques in SAT (e.g. symmetry breaking)
  - richer problem formalisms including optimization

2

Conclusion

- proof logging is well-established standard for SAT solving
- so far not usable for
  - some techniques in SAT (e.g. symmetry breaking)
  - richer problem formalisms including optimization

our work: proof logging via pseudo-Boolean proofs + verification (VeriPB[2])

- simple to implement + efficient proof checking
- applicable to wide range of combinatorial problems / algorithms
- resolve open problems for proof logging in SAT

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

Menu for second part

- live demo
- translating DRAT proofs
- full form of dominance / redundance
- deletion and dominance

Translating DRAT Proof to Pseudo-Boolean Proof

- step in DRAT proof is clausal form of redundance rule
- witness ω implicitly set by first literal
- literals represented by numbers

```
c DRAT PROOF
c loads formula implicitly
1 2 -3 0
```

```
pseudo-Boolean proof version 1.3
* load formula explicitly
f
red 1 x1 1 x2 1 ~x3 >= 1 ; x1 -> 1
```

Dominance Rule (explanation in second part)

in [BGMN22] rule is more general:

- allow to improve arbitrary preorder $\preceq$ instead of objective
- define preorder as set of constraints $\mathcal{O}_{\preceq}(\vec{u}, \vec{v})$
- proof file shows that $\mathcal{O}_{\preceq}(\vec{u}, \vec{v})$ defines preorder
- $\mathscr{C}$ is set of constraints last time preorder was changed
- $\mathscr{D}$ is set of constraints added after last time preorder was changed
- allows to use constraints in $\mathscr{D}$
- check $\alpha' \preceq \alpha$

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \models \mathscr{C}{\upharpoonright_\omega} \cup \mathcal{O}_{\preceq}(\vec{x}{\upharpoonright_\omega}, \vec{x}) \cup \{f{\upharpoonright_\omega} \leq f\}$$

- check $\alpha \not\preceq \alpha'$

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \cup \mathcal{O}_{\preceq}(\vec{x}, \vec{x}{\upharpoonright_\omega}) \models \bot$$

Redundance Rule (explanation in second part)

in [BGMN22] rule is more general:

- preorder defined through $\mathcal{O}_{\preceq}(\vec{u}, \vec{v})$
- $\mathscr{C}$ is set of constraints last time preorder was changed
- $\mathscr{D}$ is set of constraints added after last time preorder was changed

$$\frac{\mathscr{C} \cup \mathscr{D} \cup \{\, \neg C \,\} \models (\mathscr{C} \cup \mathscr{D} \cup \{\, C \,\})_{\restriction\omega} \cup \mathcal{O}_{\preceq}(\vec{x}_{\restriction\omega}, \vec{x}) \cup \{\, f_{\restriction\omega} \leq f \,\}}{C}$$

References I

[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, MarchApril 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.

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

References II

[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.

[CCT87] William Cook, Collette Rene Coullard, and György Turán.
On the Complexity of Cutting-Plane Proofs.
Discrete Applied Mathematics, 18(1):25–38, November 1987.

[CGS17] Kevin K. H. Cheung, Ambros M. Gleixner, and Daniel E. Steffy.
Verifying Integer Programming Results.
In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO '17)*, volume 10328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, June 2017.

[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.

References III

[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.

[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.

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, 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.

[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.

References V

[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 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.

[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.

[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 VI

[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.

[PCH21] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet.
A Proof Builder for Max-SAT.
In *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 488–498, 2021.

[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.

[Van08] Allen Van Gelder.
Verifying RUP Proofs of Propositional Unsatisfiability.
In *10th International Symposium on Artificial Intelligence and Mathematics (ISAIM '08)*, 2008.
Available at `http://isaim2008.unl.edu/index.php?page=proceedings`.

References VII

[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 Internatjuional Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, July 2014.