Combinatorial Solving with Provably Correct Results

Bart Bogaerts   Ciaran McCreesh   Jakob Nordström

Combinatorial Solving and Optimisation

- Revolution last couple of decades in combinatorial solvers for
  - Boolean satisfiability (SAT) solving [BHvMW21][1]
  - Constraint programming (CP) [RvBW06]
  - Mixed integer linear programming (MIP) [AW13, BR07]
- Solve NP-complete problems (or worse) very successfully in practice!
- Except solvers are sometimes wrong... (Even best commercial ones) [BLB10, CKSW13, AGJ⁺18, GSD19, GS19, BMN22, BBN⁺23]
- Even get feasibility of solutions wrong (though this should be straightforward!)
- And how to check the absence of solutions?
- Or that a solution is optimal? (Even off-by-one mistakes can snowball into large errors if solver used as subroutine)

[1] See end of slides for all references with bibliographic details

What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Inherently can only detect presence of bugs, not absence

What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Inherently can only detect presence of bugs, not absence

- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to this level of complexity

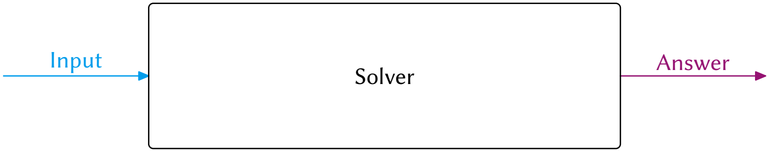
What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Inherently can only detect presence of bugs, not absence

- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to this level of complexity

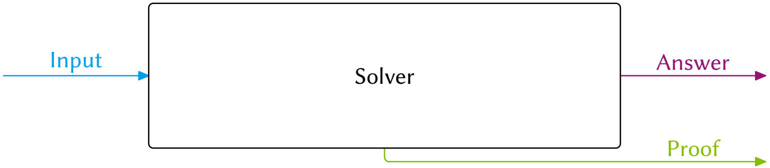
- **Proof logging**
  Make solver certifying [ABM+11, MMNS11] by outputting
  1. not only answer but also
  2. simple, machine-verifiable proof that answer is correct

Proof Logging with Certifying Solvers: Workflow



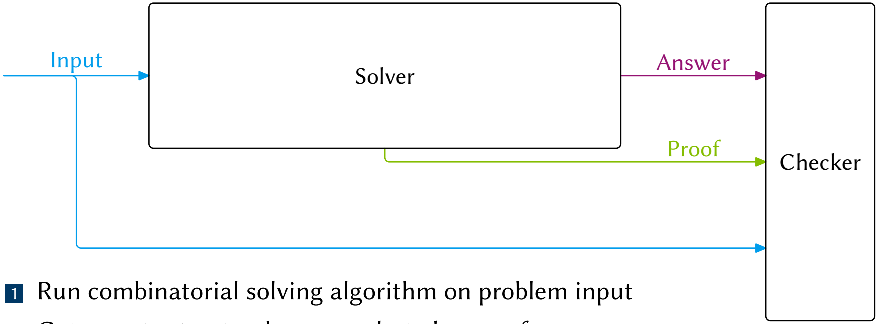
1. Run combinatorial solving algorithm on problem input

Proof Logging with Certifying Solvers: Workflow



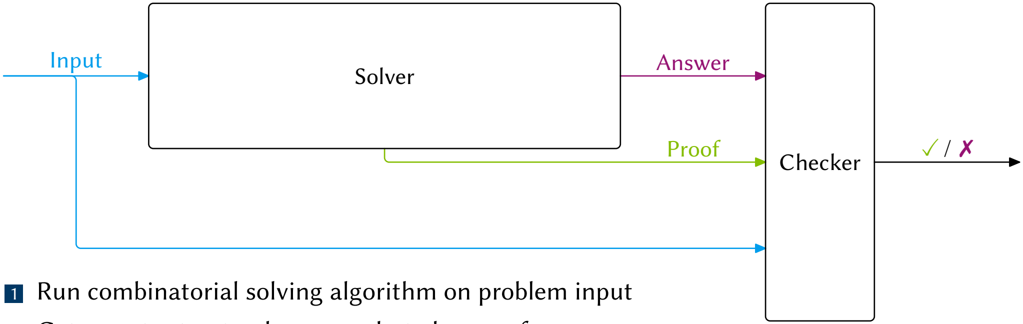
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

Proof Logging with Certifying Solvers: Workflow



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

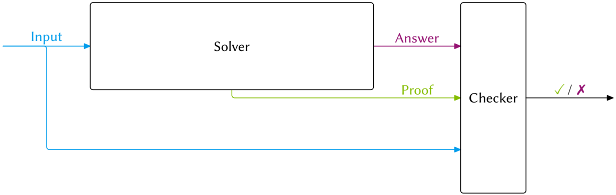
Proof Logging with Certifying Solvers: Workflow



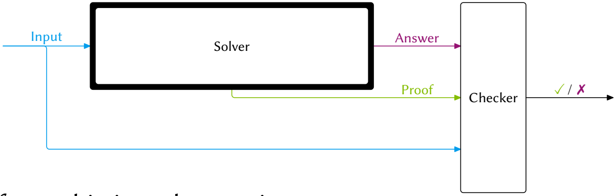
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

Proof Logging Desiderata

Proof format for certifying solver should be



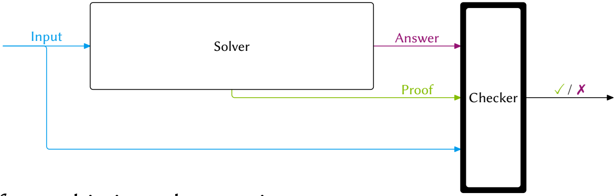
Proof Logging Desiderata



Proof format for certifying solver should be

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

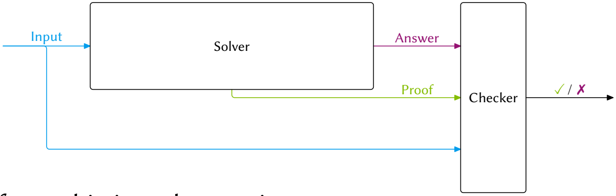
Proof Logging Desiderata



Proof format for certifying solver should be

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

Proof Logging Desiderata

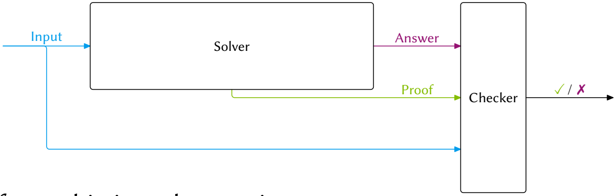


Proof format for certifying solver should be

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

Clear conflict expressivity vs. simplicity!

Proof Logging Desiderata



Proof format for certifying solver should be

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

Clear conflict expressivity vs. simplicity!

Asking for both perhaps a little bit too good to be true?

Take-Away Message from This Tutorial

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

Take-Away Message from This Tutorial

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

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

The Sales Pitch For Proof Logging

1. Certifies correctness of computed results
2. Detects errors even if due to compiler bugs, hardware failures, or cosmic rays
3. Provides debugging support during development [EG21, GMM+20, KM21, BBN+23]
4. Facilitates performance analysis
5. Helps identify potential for further improvements
6. Enables auditability
7. Serves as stepping stone towards explainability

The Rest of This Tutorial

Explain how to use VeriPB to do proof logging for

- SAT solving (including advanced techniques)
- SAT-based optimisation (MaxSAT)
- Subgraph algorithms
- Constraint programming

in a unified way

The SAT Problem

- Variable x: takes value **true** (=1) or **false** (=0)
- Literal ℓ: variable x or its negation $\overline{x}$
- Clause $C = \ell_1 \lor \cdots \lor \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \land \cdots \land C_m$: conjunction of clauses

The SAT Problem

Given a CNF formula F, is it satisfiable?

For instance, what about:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land$$
$$(x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

Proofs for SAT

For satisfiable instances: just specify satisfying assignment

For unsatisfiability: a sequence of clauses (CNF constraints)

- Each clause follows "obviously" from everything we know so far
- Final clause is empty, meaning contradiction (written $\perp$)
- Means original formula must be inconsistent

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

- $p \vee \overline{u}$ propagates $u \mapsto 0$

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

- $p \lor \overline{u}$ propagates $u \mapsto 0$
- $q \lor r$ propagates $r \mapsto 1$

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

- $p \vee \overline{u}$ propagates $u \mapsto 0$
- $q \vee r$ propagates $r \mapsto 1$
- Then $\overline{r} \vee w$ propagates $w \mapsto 1$

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

- $p \lor \overline{u}$ propagates $u \mapsto 0$
- $q \lor r$ propagates $r \mapsto 1$
- Then $\overline{r} \lor w$ propagates $w \mapsto 1$
- No further unit propagations

What Is Obvious? Unit Propagation

Unit Propagation

Clause C unit propagates ℓ under partial assignment ρ if ρ falsifies all literals in C except ℓ

Example: Unit propagate for $\rho = \{p \mapsto 0, q \mapsto 0\}$ on

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

- $p \vee \overline{u}$ propagates $u \mapsto 0$
- $q \vee r$ propagates $r \mapsto 1$
- Then $\overline{r} \vee w$ propagates $w \mapsto 1$
- No further unit propagations

Proof checker should know how to unit propagate until saturation

Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

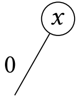
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

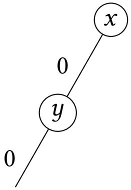


Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

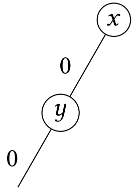


Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



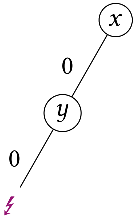
Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

1. $x \lor y$



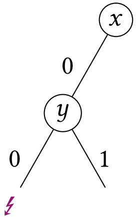
Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

“Proof trace”: when backtracking, write negation of guesses made

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

1. $x \vee y$



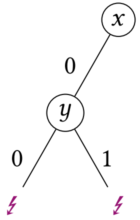
Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

1. $x \vee y$
2. $x \vee \overline{y}$



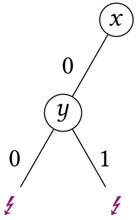
Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

1. $x \vee y$
2. $x \vee \overline{y}$
3. x



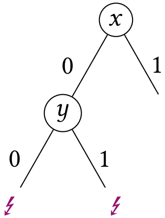
Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

1. $x \lor y$
2. $x \lor \overline{y}$
3. x



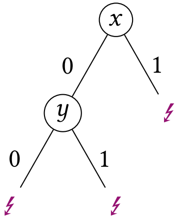
Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

1. $x \lor y$
2. $x \lor \overline{y}$
3. x
4. $\overline{x}$



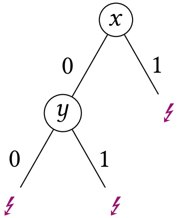
Davis-Putman-Logemann-Loveland (DPLL)

DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

"Proof trace": when backtracking, write negation of guesses made

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

1. $x \lor y$
2. $x \lor \overline{y}$
3. x
4. $\overline{x}$
5. $\perp$



Reverse Unit Propagation (RUP)

To make this a proof, need backtrack clauses to be easily verifiable

Reverse Unit Propagation (RUP)

To make this a proof, need backtrack clauses to be easily verifiable

Reverse unit propagation (RUP) clause [GN03, Van08]

C is a reverse unit propagation (RUP) clause with respect to F if

- assigning C to false
- then unit propagating on F until saturation
- leads to contradiction

If so, F clearly implies C, and this condition is easy to verify efficiently

Reverse Unit Propagation (RUP)

To make this a proof, need backtrack clauses to be easily verifiable

Reverse unit propagation (RUP) clause [GN03, Van08]

C is a reverse unit propagation (RUP) clause with respect to F if

- assigning C to false
- then unit propagating on F until saturation
- leads to contradiction

If so, F clearly implies C, and this condition is easy to verify efficiently

Fact

Backtrack clauses from DPLL solver generate a RUP proof

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

Decision

Free choice to assign value to variable

Notation $p \stackrel{\mathrm{d}}{=} 0$

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \stackrel{\mathrm{d}}{=} 0$

Decision

Free choice to assign value to variable

Notation $p \stackrel{\mathrm{d}}{=} 0$

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \stackrel{\mathrm{d}}{=} 0$

Decision

Free choice to assign value to variable

Notation $p \stackrel{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \stackrel{\mathrm{d}}{=} 0$

$u \stackrel{p \lor \overline{u}}{=} 0$

Decision

Free choice to assign value to variable

Notation $p \stackrel{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \overset{\mathrm{d}}{=} 0$

$u \overset{p \lor \overline{u}}{=} 0$

$q \overset{\mathrm{d}}{=} 0$

Decision

Free choice to assign value to variable

Notation $p \overset{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \overset{\mathrm{d}}{=} 0$

$u \overset{p \lor \overline{u}}{=} 0$

$q \overset{\mathrm{d}}{=} 0$

$r \overset{q \lor r}{=} 1$

Decision

Free choice to assign value to variable

Notation $p \overset{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \overset{\mathrm{d}}{=} 0$

$u \overset{p \lor \overline{u}}{=} 0$

$q \overset{\mathrm{d}}{=} 0$

$r \overset{q \lor r}{=} 1$

$w \overset{\overline{r} \lor w}{=} 1$

Decision

Free choice to assign value to variable

Notation $p \overset{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \stackrel{\mathrm{d}}{=} 0$

$u \stackrel{p \lor \overline{u}}{=} 0$

$q \stackrel{\mathrm{d}}{=} 0$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

$x \stackrel{\mathrm{d}}{=} 0$

Decision

Free choice to assign value to variable

Notation $p \stackrel{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \overset{\mathrm{d}}{=} 0$

$u \overset{p \lor \overline{u}}{=} 0$

$q \overset{\mathrm{d}}{=} 0$

$r \overset{q \lor r}{=} 1$

$w \overset{\overline{r} \lor w}{=} 1$

$x \overset{\mathrm{d}}{=} 0$

$y \overset{u \lor x \lor y}{=} 1$

Decision

Free choice to assign value to variable

Notation $p \overset{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \stackrel{\mathrm{d}}{=} 0$

$u \stackrel{p \lor \overline{u}}{=} 0$

$q \stackrel{\mathrm{d}}{=} 0$

$r \stackrel{q \lor r}{=} 1$

$w \stackrel{\overline{r} \lor w}{=} 1$

$x \stackrel{\mathrm{d}}{=} 0$

$y \stackrel{u \lor x \lor y}{=} 1$

$z \stackrel{x \lor \overline{y} \lor z}{=} 1$

Decision

Free choice to assign value to variable

Notation $p \stackrel{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \stackrel{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \overset{\mathrm{d}}{=} 0$

$u \overset{p \lor \overline{u}}{=} 0$

$q \overset{\mathrm{d}}{=} 0$

$r \overset{q \lor r}{=} 1$

$w \overset{\overline{r} \lor w}{=} 1$

$x \overset{\mathrm{d}}{=} 0$

$y \overset{u \lor x \lor y}{=} 1$

$z \overset{x \lor \overline{y} \lor z}{=} 1$

$\overset{\overline{y} \lor \overline{z}}{\bot}$

Decision

Free choice to assign value to variable

Notation $p \overset{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \lor \overline{u}$ forces $u = 0$

Notation $u \overset{p \lor \overline{u}}{=} 0$ ($p \lor \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

Run CDCL [BS97, MS99, MMZ+01] on our favourite CNF formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

- $p \overset{\mathrm{d}}{=} 0$
- $u \overset{p \vee \overline{u}}{=} 0$

decision level 1

- $q \overset{\mathrm{d}}{=} 0$
- $r \overset{q \vee r}{=} 1$
- $w \overset{\overline{r} \vee w}{=} 1$

decision level 2

- $x \overset{\mathrm{d}}{=} 0$
- $y \overset{u \vee x \vee y}{=} 1$
- $z \overset{x \vee \overline{y} \vee z}{=} 1$
- $\overset{\overline{y} \vee \overline{z}}{\bot}$

decision level 3

Decision

Free choice to assign value to variable

Notation $p \overset{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

decision level 1:
- $p \overset{\mathrm{d}}{=} 0$
- $u \overset{p \vee \overline{u}}{=} 0$

decision level 2:
- $q \overset{\mathrm{d}}{=} 0$
- $r \overset{q \vee r}{=} 1$
- $w \overset{\overline{r} \vee w}{=} 1$

decision level 3:
- $x \overset{\mathrm{d}}{=} 0$
- $y \overset{u \vee x \vee y}{=} 1$
- $z \overset{x \vee \overline{y} \vee z}{=} 1$
- $\overset{\overline{y} \vee \overline{z}}{\bot}$

Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

$p \overset{\mathrm{d}}{=} 0$

$u \overset{p \lor \overline{u}}{=} 0$

decision level 1

$q \overset{\mathrm{d}}{=} 0$

$r \overset{q \lor r}{=} 1$

$w \overset{\overline{r} \lor w}{=} 1$

decision level 2

$x \overset{\mathrm{d}}{=} 0$

$y \overset{u \lor x \lor y}{=} 1$

$z \overset{x \lor \overline{y} \lor z}{=} 1$

$\overset{\overline{y} \lor \overline{z}}{\bot}$

decision level 3

Could backtrack by erasing conflict level & flipping last decision

Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

Trail	
$p \stackrel{\mathrm{d}}{=} 0$	decision level 1
$u \stackrel{p \vee \overline{u}}{=} 0$	
$q \stackrel{\mathrm{d}}{=} 0$	decision level 2
$r \stackrel{q \vee r}{=} 1$	
$w \stackrel{\overline{r} \vee w}{=} 1$	
$x \stackrel{\mathrm{d}}{=} 0$	decision level 3
$y \stackrel{u \vee x \vee y}{=} 1$	
$z \stackrel{x \vee \overline{y} \vee z}{=} 1$	
$\stackrel{\overline{y} \vee \overline{z}}{\bot}$	

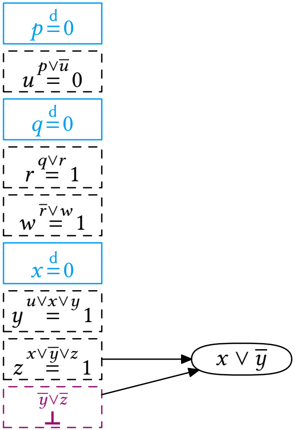
Could backtrack by erasing conflict level & flipping last decision

But want to learn from conflict and cut away as much of search space as possible

Conflict Analysis

Time to analyse this conflict and learn from it!

$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$



Could backtrack by erasing conflict level & flipping last decision

But want to learn from conflict and cut away as much of search space as possible

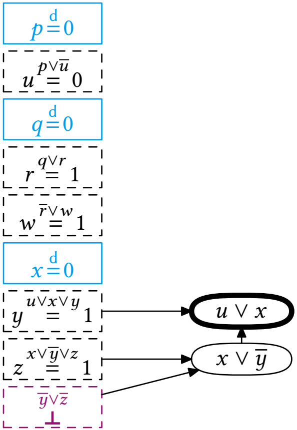
Case analysis over z for last two clauses:

- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \lor \overline{y}$

Conflict Analysis

Time to analyse this conflict and learn from it!

$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$



Could backtrack by erasing conflict level & flipping last decision

But want to learn from conflict and cut away as much of search space as possible

Case analysis over z for last two clauses:

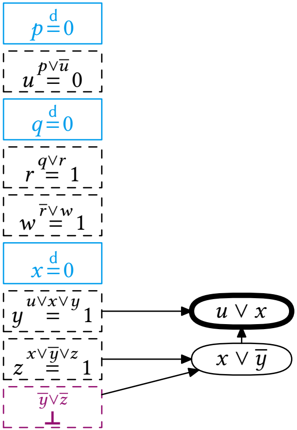
- $x \vee \overline{y} \vee z$ wants $z = 1$
- $\overline{y} \vee \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \vee \overline{y}$

Repeat until UIP clause with only 1 variable at conflict level after last decision — learn and backjump

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

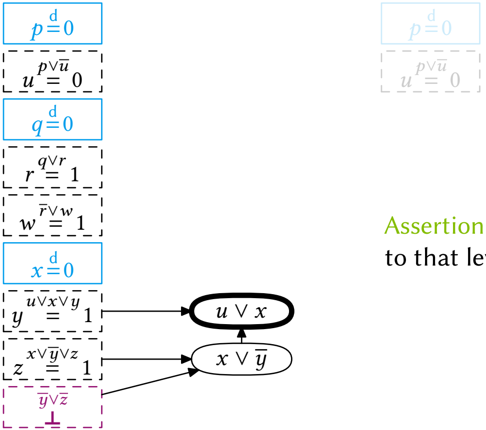
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$

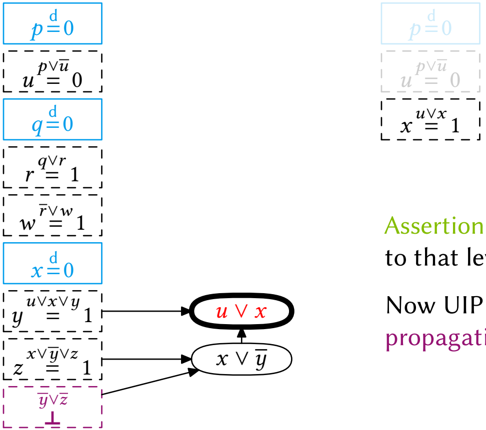


Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$



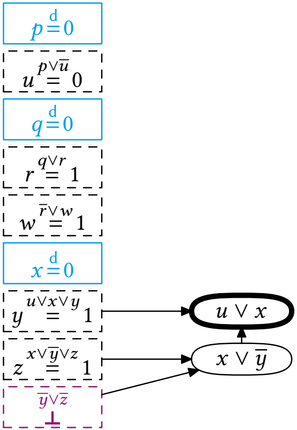
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

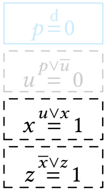
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$





Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

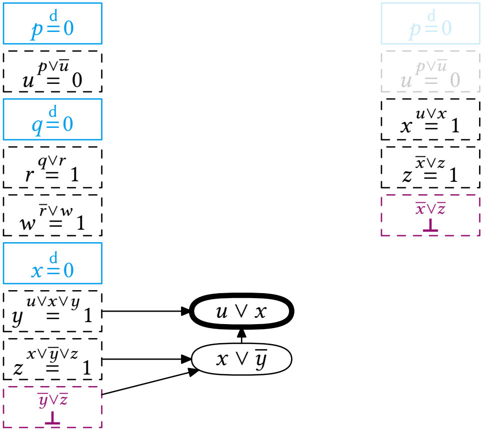
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Then continue as before...

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

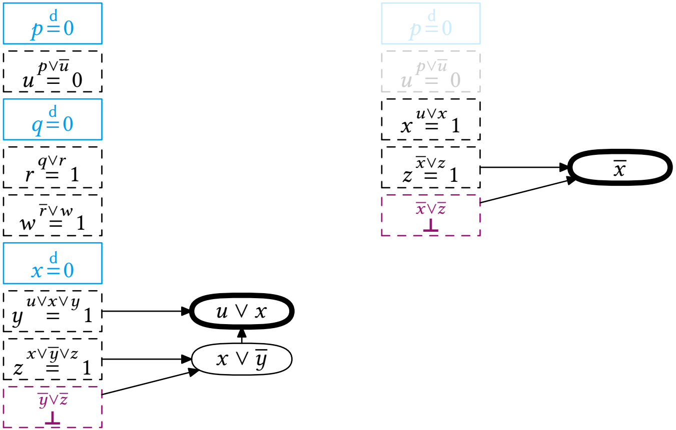
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

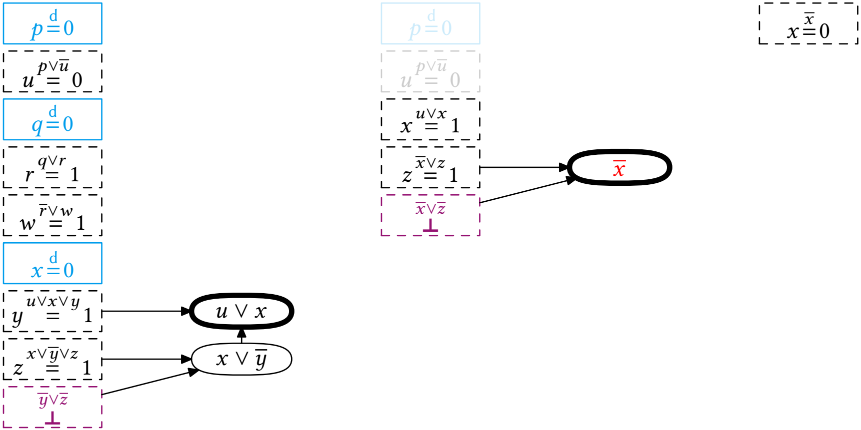
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

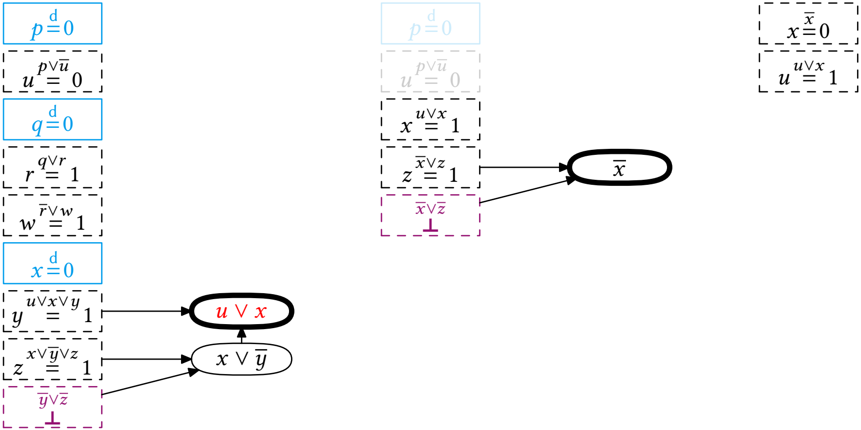
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

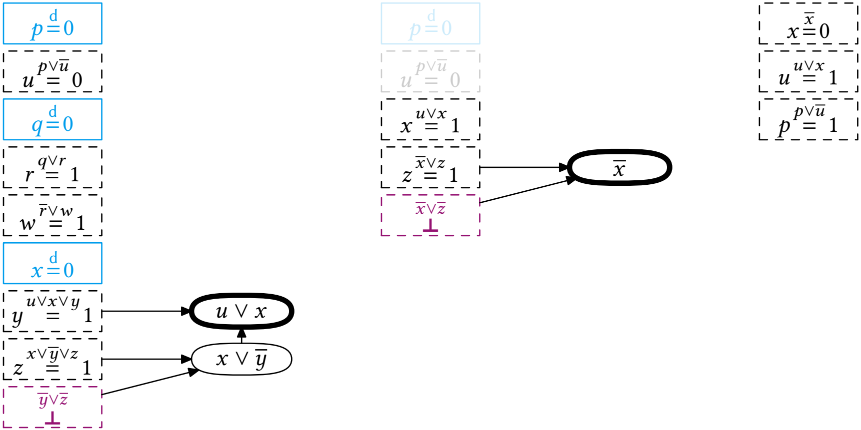
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

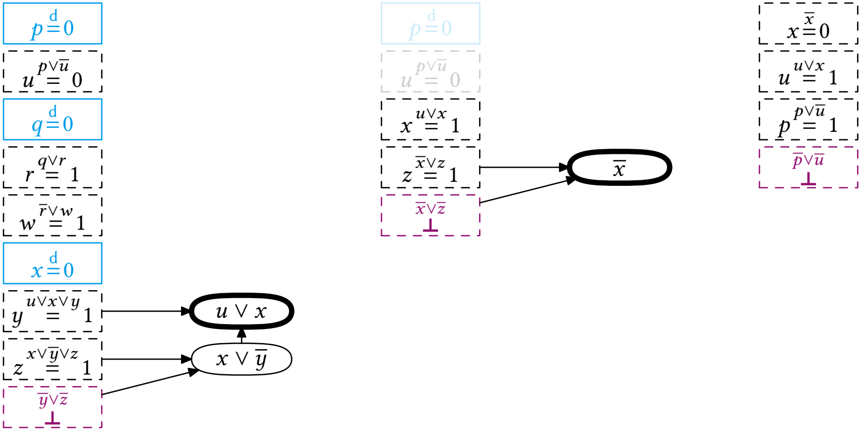
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

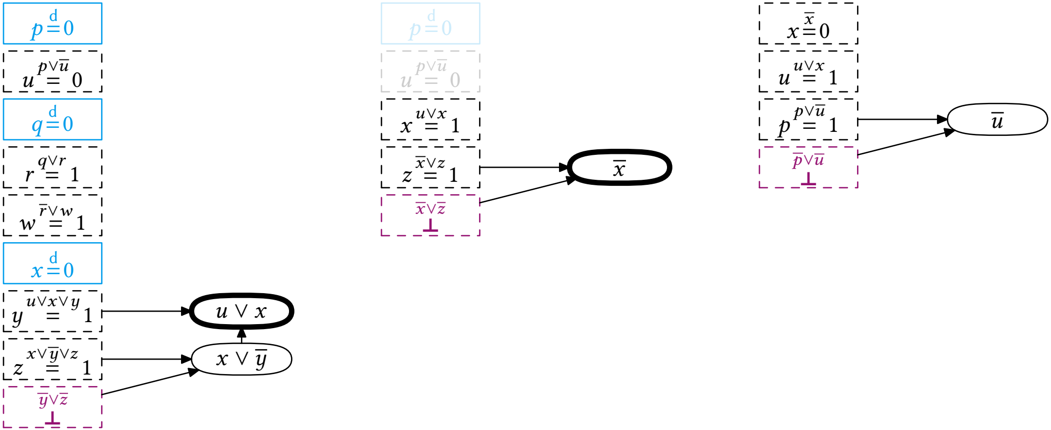
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

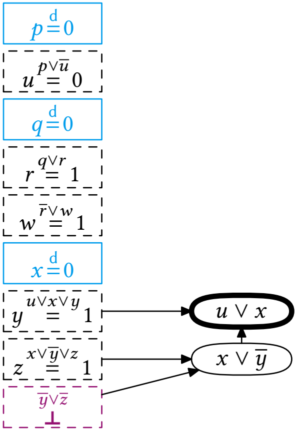
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

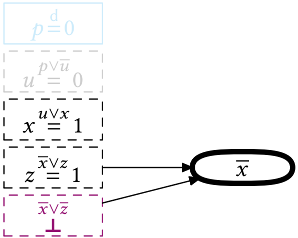


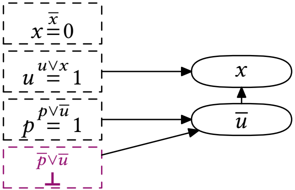
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



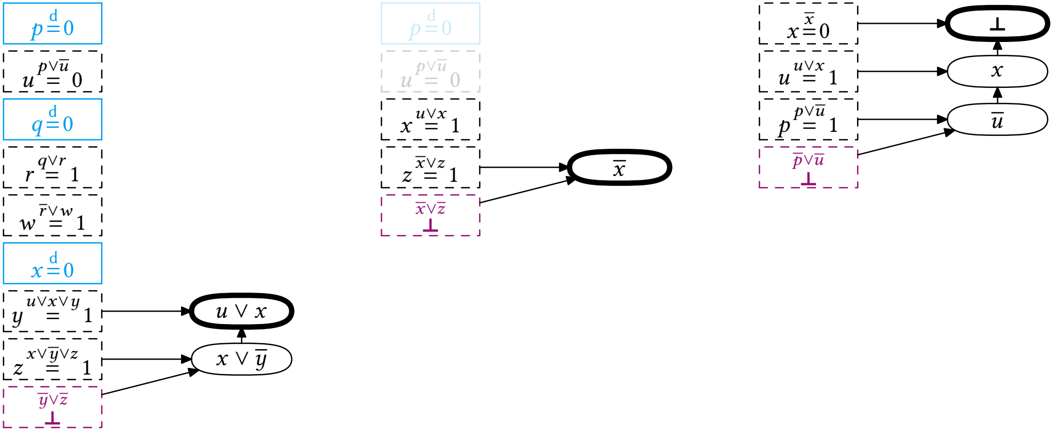




Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$



CDCL Reasoning and the Resolution Proof System

To describe CDCL reasoning, need formal proof system for unsatisfiable formulas

CDCL Reasoning and the Resolution Proof System

To describe CDCL reasoning, need formal proof system for unsatisfiable formulas

Resolution proof system [Bla37, Rob65]

- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}$$

- Done when contradiction $\perp$ in form of empty clause derived

CDCL Reasoning and the Resolution Proof System

To describe CDCL reasoning, need formal proof system for unsatisfiable formulas

Resolution proof system [Bla37, Rob65]

- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

- Done when contradiction $\perp$ in form of empty clause derived

When run on unsatisfiable formula, CDCL generates resolution proof*

CDCL Reasoning and the Resolution Proof System

To describe CDCL reasoning, need formal proof system for unsatisfiable formulas

Resolution proof system [Bla37, Rob65]

- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}$$

- Done when contradiction $\perp$ in form of empty clause derived

When run on unsatisfiable formula, CDCL generates resolution proof*

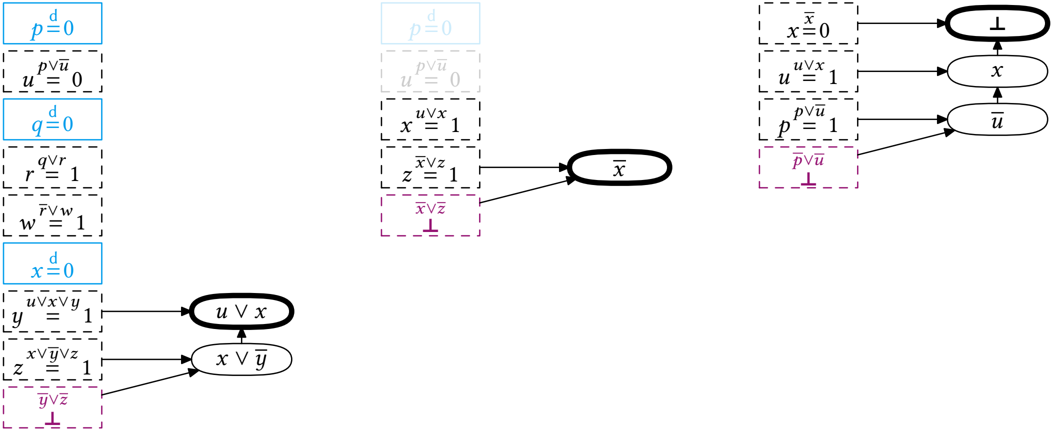
(*) Ignores pre- and inprocessing, but we will get there...

Resolution Proofs from CDCL Executions

Obtain resolution proof...

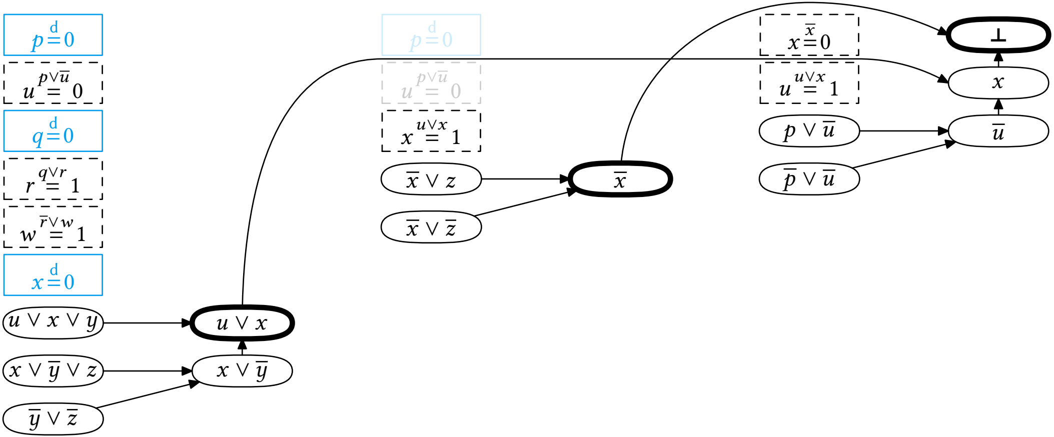
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution…



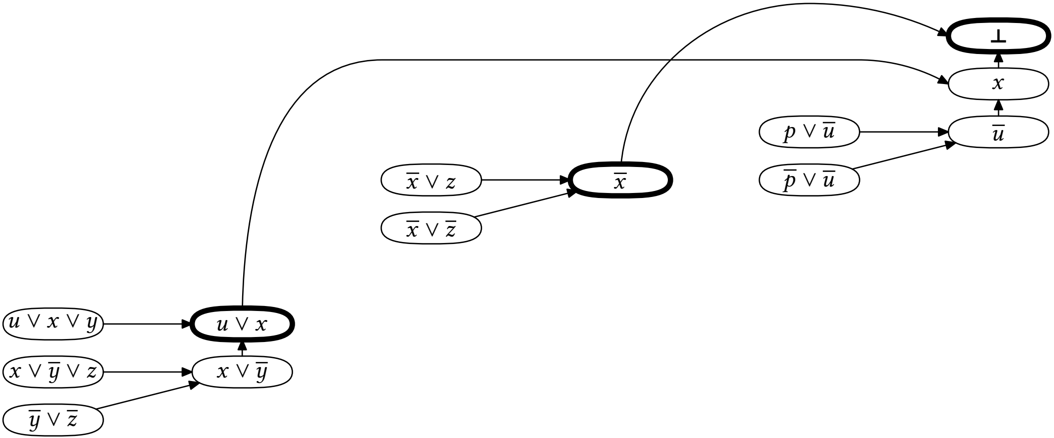
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



RUP Proofs and CDCL

But it turns out we can be lazier. . .

Fact

All learned clauses generated by CDCL solver are RUP clauses

RUP Proofs and CDCL

But it turns out we can be lazier...

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

But it turns out we can be lazier...

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\bot$

RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\bot$

RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

But it turns out we can be lazier...

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

But it turns out we can be lazier...

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\bot$

RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

But it turns out we can be lazier...

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

RUP Proofs and CDCL

But it turns out we can be lazier...

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. $\overline{x}$
3. $\perp$

More Ingredients in Proof Logging for SAT

Fact

RUP proofs can be viewed as shorthand for resolution proofs

See [BN21] for more on this and connections to SAT solving

But RUP and resolution are not enough for preprocessing, inprocessing, and some other kinds of reasoning

Extension Variables, Part 1

Suppose we want a variable a encoding

$$a \Leftrightarrow (x \land y)$$

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

$$a \lor \overline{x} \lor \overline{y} \qquad \overline{a} \lor x \qquad \overline{a} \lor y$$

for fresh variable a (this is fine since a doesn't appear anywhere previously)

Extension Variables, Part 1

Suppose we want a variable a encoding

$$a \Leftrightarrow (x \wedge y)$$

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

$$a \vee \overline{x} \vee \overline{y} \qquad \overline{a} \vee x \qquad \overline{a} \vee y$$

for fresh variable a (this is fine since a doesn't appear anywhere previously)

Fact

Extended resolution (RUP + definition of new variables) is essentially equivalent to the DRAT proof logging system most commonly used for SAT solving

Why Aren't We Done?

Practical limitations of current SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently (even for SAT solving)
- Clausal proofs can't easily reflect what algorithms for other problems do

Why Aren't We Done?

Practical limitations of current SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently (even for SAT solving)
- Clausal proofs can't easily reflect what algorithms for other problems do

Surprising claim: a slight change to 0-1 integer linear inequalities does the job!

- Enables proof logging for advanced SAT techniques so far beyond reach for efficient DRAT proof logging:
  - Cardinality reasoning
  - Gaussian elimination
  - Symmetry breaking
- Supports use of SAT solvers for optimisation problems (MaxSAT)
- Can justify graph reasoning without knowing what a graph is
- Can justify constraint programming inference without knowing what an integer variable is

Pseudo-Boolean Constraints

0–1 integer linear inequalities or (linear) pseudo-Boolean constraints:

$$\sum_i a_i \ell_i \geq A$$

- $a_i, A \in \mathbb{Z}$
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)

Pseudo-Boolean Constraints

0–1 integer linear inequalities or (linear) pseudo-Boolean constraints:

$$\sum_i a_i \ell_i \geq A$$

- $a_i, A \in \mathbb{Z}$
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)

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

Some Types of Pseudo-Boolean Constraints

1. Clauses

$$x_1 \lor \overline{x}_2 \lor x_3 \quad \Leftrightarrow \quad x_1 + \overline{x}_2 + x_3 \geq 1$$

2. Cardinality constraints

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

3. General pseudo-Boolean constraints

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms From the input

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms From the input

Literal axioms $$\frac{}{\ell_i \geq 0}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms From the input

Literal axioms $\dfrac{}{\ell_i \geq 0}$

Addition $\dfrac{\sum_i a_i\ell_i \geq A \qquad \sum_i b_i\ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms From the input

Literal axioms

$$\frac{}{\ell_i \geq 0}$$

Addition

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$$

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

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms — From the input

Literal axioms

$$\frac{}{\ell_i \geq 0}$$

Addition

$$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (a_i + b_i)\ell_i \geq A + B}$$

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

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

Division for any $c \in \mathbb{N}^+$
(assumes normalized form)

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

Cutting Planes Toy Example

$$w + 2x + y \geq 2$$

Cutting Planes Toy Example

$$\text{Multiply by 2} \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$$

Cutting Planes Toy Example

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

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

Multiply by 2
$$\frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$$

Add
$$\frac{2w + 4x + 2y \geq 4 \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9}$$

Multiply by 2
$$\frac{\overline{z} \geq 0}{2\overline{z} \geq 0}$$

Add
$$\frac{3w + 6x + 6y + 2z \geq 9 \qquad 2\overline{z} \geq 0}{3w + 6x + 6y \qquad \geq 7}$$

Cutting Planes Toy Example

Multiply by 2: $\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$

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

Multiply by 2: $\dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}$

Add: $\dfrac{3w + 6x + 6y + 2z \geq 9 \qquad 2\overline{z} \geq 0}{3w + 6x + 6y \geq 7}$

Divide by 3: $\dfrac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 2\frac{1}{3}}$

Cutting Planes Toy Example

Multiply by 2: $\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$

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

Multiply by 2: $\dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}$

Add: $\dfrac{3w + 6x + 6y + 2z \geq 9 \qquad 2\overline{z} \geq 0}{3w + 6x + 6y \geq 7}$

Divide by 3: $\dfrac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 3}$

Cutting Planes Toy Example

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

Naming constraints by integers and literal axioms by the literal involved (with ~ for negation) as

$$
\begin{aligned}
\text{Constraint 1} &\doteq 2x + y + w \geq 2 \\
\text{Constraint 2} &\doteq 2x + 4y + 2z + w \geq 5 \\
\sim z &\doteq \overline{z} \geq 0
\end{aligned}
$$

Cutting Planes Toy Example

Multiply by 2: $\dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}$

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

Multiply by 2: $\dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}$

Add: $\dfrac{3w + 6x + 6y + 2z \geq 9 \qquad 2\overline{z} \geq 0}{3w + 6x + 6y \geq 7}$

Divide by 3: $\dfrac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 3}$

Naming constraints by integers and literal axioms by the literal involved (with ~ for negation) as

$$\text{Constraint 1} \doteq 2x + y + w \geq 2$$
$$\text{Constraint 2} \doteq 2x + 4y + 2z + w \geq 5$$
$$\sim z \doteq \overline{z} \geq 0$$

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

```
pol 1 2 * 2 + ~z 2 * + 3 d
```

Resolution and Cutting Planes

To simulate resolution step such as

$$\frac{\overline{y} \vee \overline{z} \qquad x \vee \overline{y} \vee z}{x \vee \overline{y}}$$

we can perform the cutting planes steps

$$\text{Add}\ \frac{\text{Divide by 2}\ \dfrac{\overline{y} + \overline{z} \geq 1 \qquad x + \overline{y} + z \geq 1}{x + 2\overline{y} \geq 1}}{x + \overline{y} \geq 1}$$

Resolution and Cutting Planes

To simulate resolution step such as

$$\frac{\overline{y} \lor \overline{z} \qquad x \lor \overline{y} \lor z}{x \lor \overline{y}}$$

we can perform the cutting planes steps

$$\text{Divide by 2}\ \dfrac{\text{Add}\ \dfrac{\overline{y} + \overline{z} \geq 1 \qquad x + \overline{y} + z \geq 1}{x + 2\overline{y} \geq 1}}{x + \overline{y} \geq 1}$$

Given that the premises are clauses 7 and 5 in our example CNF formula, using references

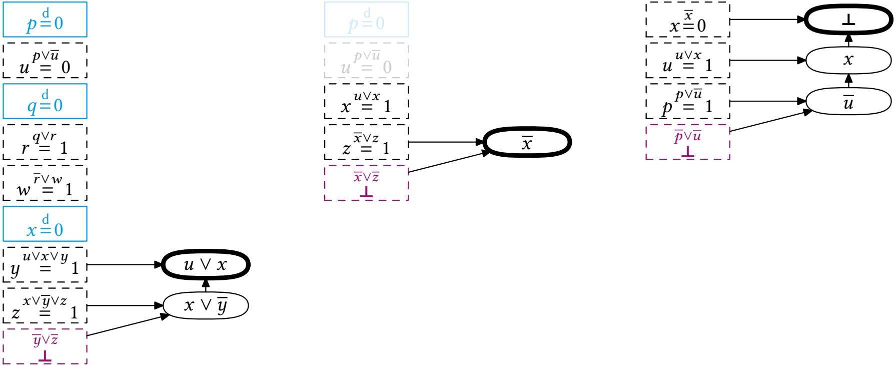
$$\text{Constraint 7} \doteq \overline{y} + \overline{z} \geq 1$$

$$\text{Constraint 5} \doteq x + \overline{y} + z \geq 1$$

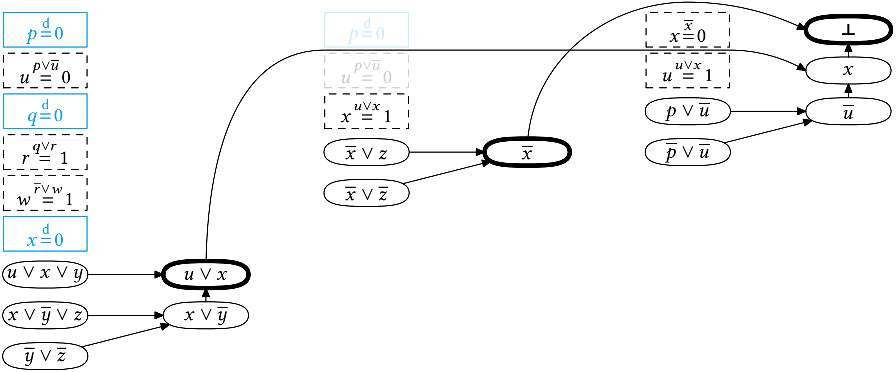
we can write this in the proof log as

```
pol 7 5 + 2 d
```

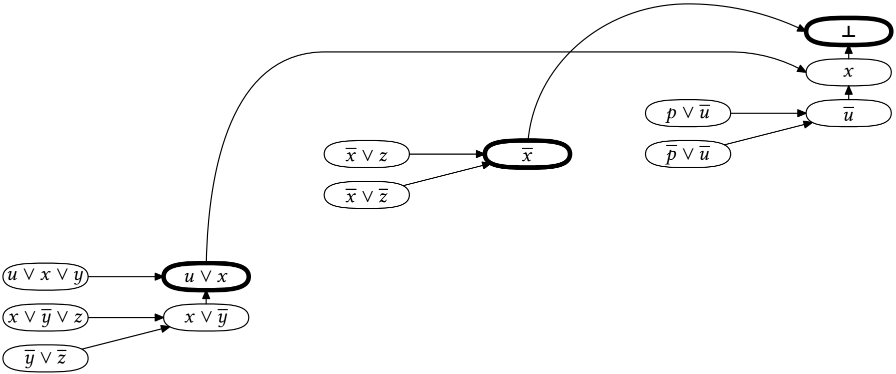
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



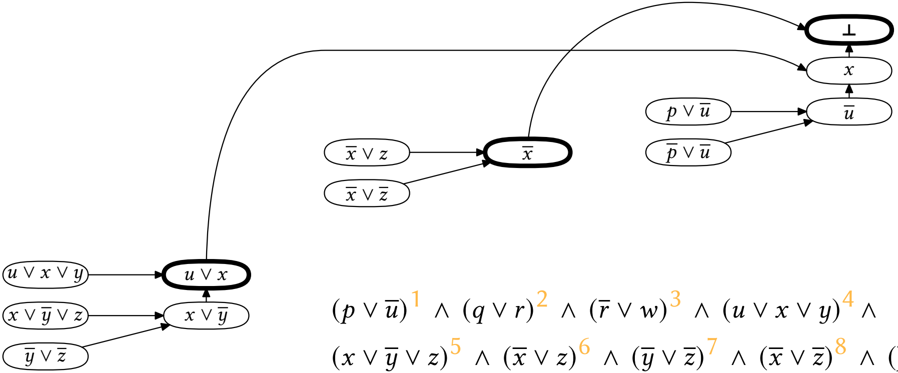
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses

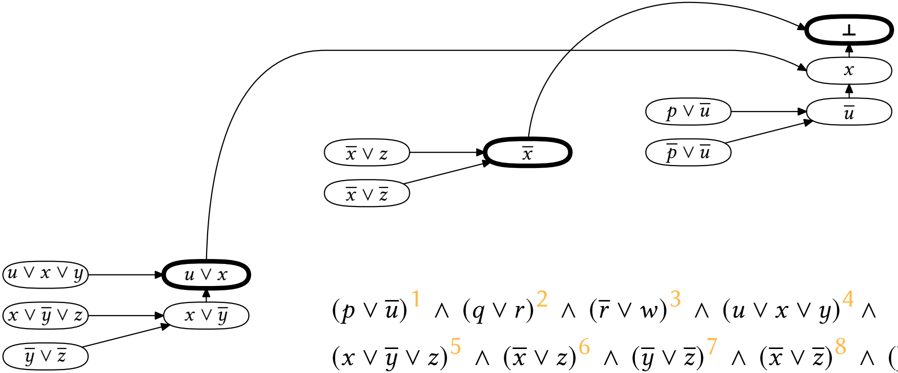


Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



$(p \vee \overline{u})^{1} \wedge (q \vee r)^{2} \wedge (\overline{r} \vee w)^{3} \wedge (u \vee x \vee y)^{4} \wedge$
$(x \vee \overline{y} \vee z)^{5} \wedge (\overline{x} \vee z)^{6} \wedge (\overline{y} \vee \overline{z})^{7} \wedge (\overline{x} \vee \overline{z})^{8} \wedge (\overline{p} \vee \overline{u})^{9}$

Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



$$(p \vee \overline{u})^1 \wedge (q \vee r)^2 \wedge (\overline{r} \vee w)^3 \wedge (u \vee x \vee y)^4 \wedge$$
$$(x \vee \overline{y} \vee z)^5 \wedge (\overline{x} \vee z)^6 \wedge (\overline{y} \vee \overline{z})^7 \wedge (\overline{x} \vee \overline{z})^8 \wedge (\overline{p} \vee \overline{u})^9$$

```
pol 7 5 + 2 d 4 + 2 d
pol 8 6 + 2 d
pol 9 1 + 2 d 10 + 2 d 11 + 2 d
```

$\rightsquigarrow$ Constraint 10 $\doteq u + x \geq 1$

$\rightsquigarrow$ Constraint 11 $\doteq \overline{x} \geq 1$

$\rightsquigarrow$ Constraint 12 $\doteq 0 \geq 1$ ↯

RUP Revisited

Can define (reverse) unit propagation in a pseudo-Boolean setting

Constraint C propagates variable x if setting x to "wrong value" would make C unsatisfiable

E.g., if x_5 is false,

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

would propagate $\overline{x}_4$ (since other coefficients do not add up to 7)

RUP Revisited

Can define (reverse) unit propagation in a pseudo-Boolean setting

Constraint C propagates variable x if setting x to "wrong value" would make C unsatisfiable

E.g., if x_5 is false,

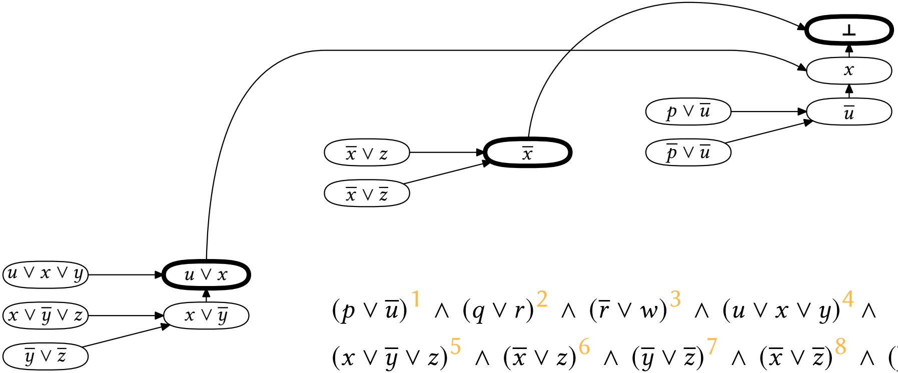
$$x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

would propagate $\overline{x}_4$ (since other coefficients do not add up to 7)

Risk for confusion:

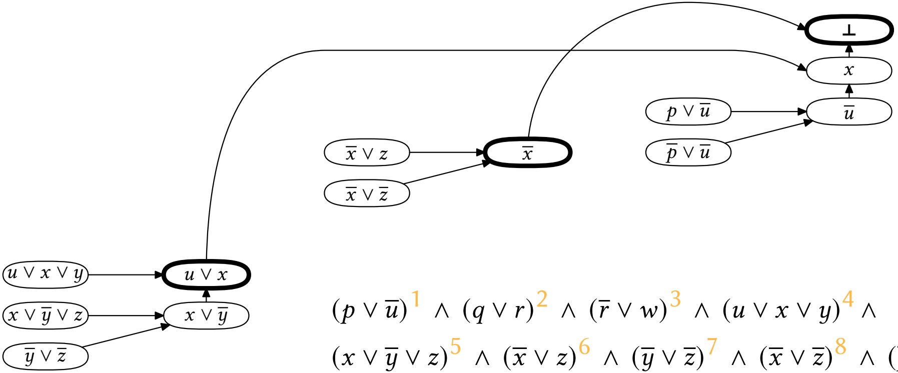
- Constraint programming people might call this (reverse) integer bounds consistency
  - Does the same thing if we're working with clauses
  - More interesting for general pseudo-Boolean constraints
- SAT people beware: constraints can propagate multiple times and multiple variables

Pseudo-Boolean Proof Logging for Example CDCL Execution with RUP



$$(p \lor \overline{u})^{1} \land (q \lor r)^{2} \land (\overline{r} \lor w)^{3} \land (u \lor x \lor y)^{4} \land$$
$$(x \lor \overline{y} \lor z)^{5} \land (\overline{x} \lor z)^{6} \land (\overline{y} \lor \overline{z})^{7} \land (\overline{x} \lor \overline{z})^{8} \land (\overline{p} \lor \overline{u})^{9}$$

Pseudo-Boolean Proof Logging for Example CDCL Execution with RUP



$(p \lor \overline{u})^{1} \land (q \lor r)^{2} \land (\overline{r} \lor w)^{3} \land (u \lor x \lor y)^{4} \land$
$(x \lor \overline{y} \lor z)^{5} \land (\overline{x} \lor z)^{6} \land (\overline{y} \lor \overline{z})^{7} \land (\overline{x} \lor \overline{z})^{8} \land (\overline{p} \lor \overline{u})^{9}$

```
rup 1 u 1 x >= 1 ;
rup 1 ~x >= 1 ;
rup >= 1 ;
```

⇝ Constraint 10 $\doteq u + x \geq 1$
⇝ Constraint 11 $\doteq \overline{x} \geq 1$
⇝ Constraint 12 $\doteq 0 \geq 1$ ↯

Extension Variables, Part 2

Suppose we want new, fresh variable a encoding

$$a \Leftrightarrow (3x + 2y + z + w \geq 3)$$

This time, introduce constraints

$$3\overline{a} + 3x + 2y + z + w \geq 3 \qquad 5a + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5$$

Again, needs support from the proof system

Proof Logs for “Extended Cutting Planes”

For satisfiable instances: just specify a satisfying assignment.

For unsatisfiability: a sequence of pseudo-Boolean constraints in (slight extension of) OPB format [RM16]

- Each constraint follows “obviously” from what is known so far
- Either implicitly, by RUP…
- Or by an explicit cutting planes derivation…
- Or as an extension variable reifying a new constraint*
- Final constraint is $0 \geq 1$

Proof Logs for “Extended Cutting Planes”

For satisfiable instances: just specify a satisfying assignment.

For unsatisfiability: a sequence of pseudo-Boolean constraints in (slight extension of) OPB format [RM16]

- Each constraint follows “obviously” from what is known so far
- Either implicitly, by RUP…
- Or by an explicit cutting planes derivation…
- Or as an extension variable reifying a new constraint*
- Final constraint is $0 \geq 1$

(*) Not actually implemented this way — details in extended version of this tutorial

Deleting Constraints

In practice, important to erase constraints to save memory and time during verification

Fairly straightforward to deal with from the point of view of proof logging

So ignored in this tutorial for simplicity and clarity

Enumeration and Optimisation Problems

Enumeration:

- When a solution is found, can log it
- Introduces a new constraint saying “not this solution”
- So the proof semantics is “infeasible, except for all the solutions I told you about”

Enumeration and Optimisation Problems

Enumeration:

- When a solution is found, can log it
- Introduces a new constraint saying "not this solution"
- So the proof semantics is "infeasible, except for all the solutions I told you about"

For optimisation:

- Define an objective $f = \sum_i w_i \ell_i$, $w_i \in \mathbb{Z}$, to minimise subject to the contraints in the formula
- To maximise, negate objective
- Log a solution α; get an objective-improving constraint $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \alpha(\ell_i)$
- Semantics for proof of optimality: "infeasible to find better solution than best so far"

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0–1 integer linear program (ILP)

- just do proof logging

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0–1 integer linear program (ILP)

- just do proof logging

Otherwise

- do trusted or verified translation to 0–1 ILP
- provide proof logging for 0–1 ILP formulation

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0–1 integer linear program (ILP)

- just do proof logging

Otherwise

- do trusted or verified translation to 0–1 ILP
- provide proof logging for 0–1 ILP formulation

Proof logging philosophy:

- **do not change** input for solver
- **do not change** reasoning in solver
- **only** add print statements (in PB format) here and there

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0–1 integer linear program (ILP)

- just do proof logging

Otherwise

- do trusted or verified translation to 0–1 ILP
- provide proof logging for 0–1 ILP formulation

Proof logging philosophy:

- **do not change** input for solver
- **do not change** reasoning in solver
- **only** add print statements (in PB format) here and there

Goldilocks compromise between expressivity and simplicity:

1. 0–1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments (even for SAT)
3. Efficient reification of constraints

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0–1 integer linear program (ILP)

- just do proof logging

Otherwise

- do trusted or verified translation to 0–1 ILP
- provide proof logging for 0–1 ILP formulation

Proof logging philosophy:

- **do not change** input for solver
- **do not change** reasoning in solver
- **only** add print statements (in PB format) here and there

Goldilocks compromise between expressivity and simplicity:

1. 0–1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments (even for SAT)
3. Efficient reification of constraints — example:

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

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

Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0–1 integer linear program (ILP)

- just do proof logging

Otherwise

- do trusted or verified translation to 0–1 ILP
- provide proof logging for 0–1 ILP formulation

Proof logging philosophy:

- **do not change** input for solver
- **do not change** reasoning in solver
- **only** add print statements (in PB format) here and there

Goldilocks compromise between expressivity and simplicity:

1. 0–1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments (even for SAT)
3. Efficient reification of constraints — example:

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

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

The VeriPB Format and Tool

https://gitlab.com/MIAOresearch/software/VeriPB

Released under MIT Licence

Various features to help development:

- Extended variable name syntax allowing human-readable names
- Proof tracing
- "Trust me" assertions for incremental proof logging

Documentation:

- Description of VeriPB checker [BMM+23] used in SAT 2023 competition (https://satcompetition.github.io/2023/checkers.html)
- Specific details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23]
- Lots of concrete example files at https://gitlab.com/MIAOresearch/software/VeriPB

Parity (XOR) Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Parity (XOR) Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

Parity (XOR) Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Parity (XOR) Reasoning

Given clauses

$$x \vee y \vee z$$
$$x \vee \overline{y} \vee \overline{z}$$
$$\overline{x} \vee y \vee \overline{z}$$
$$\overline{x} \vee \overline{y} \vee z$$

and

$$y \vee z \vee w$$
$$y \vee \overline{z} \vee \overline{w}$$
$$\overline{y} \vee z \vee \overline{w}$$
$$\overline{y} \vee \overline{z} \vee w$$

want to derive

$$x \vee \overline{w}$$
$$\overline{x} \vee w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Exponentially hard for CDCL [Urq87]
But used in *CryptoMiniSat* [Cry]

Parity (XOR) Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Exponentially hard for CDCL [Urq87]
But used in *CryptoMiniSat* [Cry]

DRAT proof logging like [PR16] too inefficient in practice!

Parity (XOR) Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Exponentially hard for CDCL [Urq87]
But used in *CryptoMiniSat* [Cry]

DRAT proof logging like [PR16] too inefficient in practice!

Could add XORs to language, but prefer to keep things super-simple

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Introduce extension variables a, b and derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

(“=” syntactic sugar for “≥” plus “≤”)

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Introduce extension variables a, b and derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("=" syntactic sugar for "≥" plus "≤")
Add to get

$$x + w + 2y + 2z + 2a + 2b = 6$$

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Introduce extension variables a, b and derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("=" syntactic sugar for "≥" plus "≤")
Add to get

$$x + w + 2y + 2z + 2a + 2b = 6$$

From this can extract

$$x + \overline{w} \geq 1$$
$$\overline{x} + w \geq 1$$

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

$$x \lor y \lor z$$
$$x \lor \overline{y} \lor \overline{z}$$
$$\overline{x} \lor y \lor \overline{z}$$
$$\overline{x} \lor \overline{y} \lor z$$

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Introduce extension variables a, b and derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("=" syntactic sugar for "≥" plus "≤")
Add to get

$$x + w + 2y + 2z + 2a + 2b = 6$$

From this can extract

$$x + \overline{w} \geq 1$$
$$\overline{x} + w \geq 1$$

VeriPB can certify XOR reasoning [GN21]

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

- *MiniSat+* [ES06]
- *Open-WBO* [MML14]
- *NaPS* [SN15]

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

- *MiniSat+* [ES06]
- *Open-WBO* [MML14]
- *NaPS* [SN15]

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

- *MiniSat+* [ES06]
- *Open-WBO* [MML14]
- *NaPS* [SN15]

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

$\overline{s}_{1,1} \vee x_1$

$\overline{s}_{2,1} \vee s_{1,1} \vee x_2$

$\overline{s}_{2,2} \vee s_{1,1}$

$\overline{s}_{2,2} \vee x_2$

$\overline{s}_{3,1} \vee s_{2,1} \vee x_3$

$\overline{s}_{3,2} \vee s_{2,1}$

$\overline{s}_{3,2} \vee s_{2,2} \vee x_3$

$\overline{s}_{4,1} \vee s_{3,1} \vee x_4$

$\overline{s}_{4,2} \vee s_{3,1}$

$\overline{s}_{4,2} \vee s_{3,2} \vee x_4$

$s_{4,2}$

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

- *MiniSat+* [ES06]
- *Open-WBO* [MML14]
- *NaPS* [SN15]

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

$\overline{s}_{1,1} \vee x_1$

$\overline{s}_{2,1} \vee s_{1,1} \vee x_2$

$\overline{s}_{2,2} \vee s_{1,1}$

$\overline{s}_{2,2} \vee x_2$

$\overline{s}_{3,1} \vee s_{2,1} \vee x_3$

$\overline{s}_{3,2} \vee s_{2,1}$

$\overline{s}_{3,2} \vee s_{2,2} \vee x_3$

$\overline{s}_{4,1} \vee s_{3,1} \vee x_4$

$\overline{s}_{4,2} \vee s_{3,1}$

$\overline{s}_{4,2} \vee s_{3,2} \vee x_4$

$s_{4,2}$

How to know translation is correct?

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

- *MiniSat+* [ES06]
- *Open-WBO* [MML14]
- *NaPS* [SN15]

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

$$k \cdot \overline{s}_{i,k} + \sum_{j=1}^{i} x_j \geq k$$

$$(i - k + 1) \cdot s_{i,k} + \sum_{j=1}^{i} \overline{x}_j \geq i - k + 1$$

$\overline{s}_{1,1} \lor x_1$

$\overline{s}_{2,1} \lor s_{1,1} \lor x_2$

$\overline{s}_{2,2} \lor s_{1,1}$

$\overline{s}_{2,2} \lor x_2$

$\overline{s}_{3,1} \lor s_{2,1} \lor x_3$

$\overline{s}_{3,2} \lor s_{2,1}$

$\overline{s}_{3,2} \lor s_{2,2} \lor x_3$

$\overline{s}_{4,1} \lor s_{3,1} \lor x_4$

$\overline{s}_{4,2} \lor s_{3,1}$

$\overline{s}_{4,2} \lor s_{3,2} \lor x_4$

$s_{4,2}$

How to know translation is correct?

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

- *MiniSat+* [ES06]
- *Open-WBO* [MML14]
- *NaPS* [SN15]

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

$$k \cdot \overline{s}_{i,k} + \sum_{j=1}^{i} x_j \geq k$$

$$(i - k + 1) \cdot s_{i,k} + \sum_{j=1}^{i} \overline{x}_j \geq i - k + 1$$

$\overline{s}_{1,1} \lor x_1$

$\overline{s}_{2,1} \lor s_{1,1} \lor x_2$

$\overline{s}_{2,2} \lor s_{1,1}$

$\overline{s}_{2,2} \lor x_2$

$\overline{s}_{3,1} \lor s_{2,1} \lor x_3$

$\overline{s}_{3,2} \lor s_{2,1}$

$\overline{s}_{3,2} \lor s_{2,2} \lor x_3$

$\overline{s}_{4,1} \lor s_{3,1} \lor x_4$

$\overline{s}_{4,2} \lor s_{3,1}$

$\overline{s}_{4,2} \lor s_{3,2} \lor x_4$

$s_{4,2}$

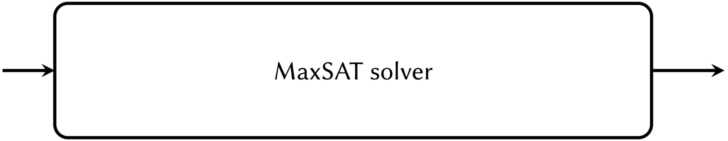
How to know translation is correct?

VeriPB can certify pseudo-Boolean-to-CNF rewriting [GMNO22, VDB22]

Certified Maximum Satisfiability (MaxSAT) Solving

Minimize linear objective subject to satisfying formula in conjunctive normal form (CNF)

$$
\begin{aligned}
\min\ & 2x_1 + x_2 \\
\text{s.t. } & x_1 \vee \overline{z} \\
& z \vee x_2
\end{aligned}
$$



Result:
optimum 1

Many MaxSAT solvers internally make use of SAT solver.

Certified Maximum Satisfiability (MaxSAT) Solving

Minimize linear objective subject to satisfying formula in conjunctive normal form (CNF)

$$\begin{aligned} \min\ & 2x_1 + x_2 \\ \text{s.t.}\ & x_1 \vee \overline{z} \\ & z \vee x_2 \end{aligned}$$

→ MaxSAT solver → Result: optimum 1

Many MaxSAT solvers internally make use of SAT solver. Idea:

- Find optimal solution (checking that it *is* a solution is easy)
- Add clauses claiming a better solution exists
- Use one extra SAT call to get proof of optimality (with standard SAT proof logging)

Certified Maximum Satisfiability (MaxSAT) Solving

Minimize linear objective subject to satisfying formula in conjunctive normal form (CNF)

$$\begin{aligned} \min\ & 2x_1 + x_2 \\ \text{s.t. } & x_1 \lor \overline{z} \\ & z \lor x_2 \end{aligned}$$

→ MaxSAT solver → Result: optimum 1

Many MaxSAT solvers internally make use of SAT solver. Idea:

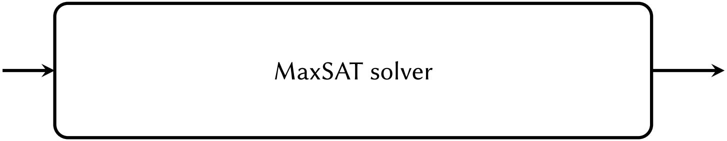
- Find optimal solution (checking that it *is* a solution is easy)
- Add clauses claiming a better solution exists
- Use one extra SAT call to get proof of optimality (with standard SAT proof logging)

Does not work

Certified Maximum Satisfiability (MaxSAT) Solving

Minimize linear objective subject to satisfying formula in conjunctive normal form (CNF)

$$\begin{aligned} \min\ & 2x_1 + x_2 \\ \text{s.t.}\ & x_1 \lor \overline{z} \\ & z \lor x_2 \end{aligned}$$



Result:
optimum 1

Many MaxSAT solvers internally make use of SAT solver. Idea:

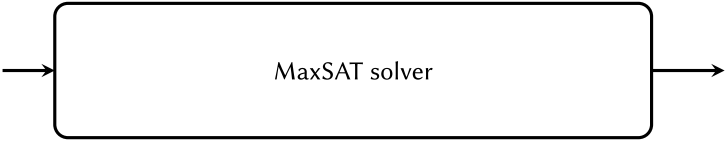
- Find optimal solution (checking that it *is* a solution is easy)
- Add clauses claiming a better solution exists
  Requires proof logging — can be done with VeriPB
- Use one extra SAT call to get proof of optimality (with standard SAT proof logging)
  Causes serious overhead

Does not work

Certified Maximum Satisfiability (MaxSAT) Solving

Minimize linear objective subject to satisfying formula in conjunctive normal form (CNF)

$$\begin{aligned} \min\ & 2x_1 + x_2 \\ \text{s.t. } & x_1 \lor \overline{z} \\ & z \lor x_2 \end{aligned}$$



Result:
optimum 1

Many MaxSAT solvers internally make use of SAT solver. Idea:

- Find optimal solution (checking that it *is* a solution is easy)
- Add clauses claiming a better solution exists
  Requires proof logging — can be done with VeriPB
- Use one extra SAT call to get proof of optimality (with standard SAT proof logging)
  Causes serious overhead

Does not work Only proves answer correct, not reasoning within solver!

MaxSAT Solvers

Three main categories:

- Linear SAT-UNSAT search
  1. Call SAT solver to find some solution
  2. Add clauses encoding "I want a better solution"
  3. Repeat (last found solution is optimal)

MaxSAT Solvers

Three main categories:

- Linear SAT-UNSAT search
  1. Call SAT solver to find some solution
  2. Add clauses encoding "I want a better solution"
  3. Repeat (last found solution is optimal)

  VeriPB-based proof logging available [VDB22, Van23]

MaxSAT Solvers

Three main categories:

- Linear SAT-UNSAT search
  1. Call SAT solver to find some solution
  2. Add clauses encoding “I want a better solution”
  3. Repeat (last found solution is optimal)

  VeriPB-based proof logging available [VDB22, Van23]
- Core-guided search
  1. Call SAT solver to find solution under most optimistic assumptions
  2. If impossible, rewrite objective given output of SAT solver
  3. Repeat (first solution is optimal)

MaxSAT Solvers

Three main categories:

- Linear SAT-UNSAT search
  1. Call SAT solver to find some solution
  2. Add clauses encoding "I want a better solution"
  3. Repeat (last found solution is optimal)

  VeriPB-based proof logging available [VDB22, Van23]
- Core-guided search
  1. Call SAT solver to find solution under most optimistic assumptions
  2. If impossible, rewrite objective given output of SAT solver
  3. Repeat (first solution is optimal)

  VeriPB-based proof logging available [BBN+23]

MaxSAT Solvers

Three main categories:

- Linear SAT-UNSAT search
  1. Call SAT solver to find some solution
  2. Add clauses encoding "I want a better solution"
  3. Repeat (last found solution is optimal)

  VeriPB-based proof logging available [VDB22, Van23]
- Core-guided search
  1. Call SAT solver to find solution under most optimistic assumptions
  2. If impossible, rewrite objective given output of SAT solver
  3. Repeat (first solution is optimal)

  VeriPB-based proof logging available [BBN+23]
- Implicit Hitting Set
  1. Call SAT solver to find solution under most optimistic assumptions
  2. Use hitting set solver (MIP solver) to recompute what most possible optimistic assumptions are
  3. Repeat (first solution is optimal)

MaxSAT Solvers

Three main categories:

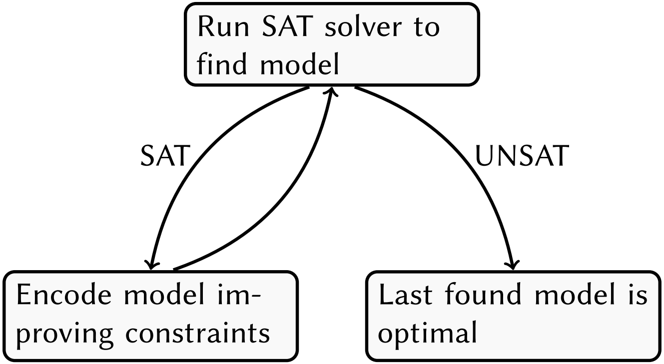
- Linear SAT-UNSAT search
  1. Call SAT solver to find some solution
  2. Add clauses encoding "I want a better solution"
  3. Repeat (last found solution is optimal)

  VeriPB-based proof logging available [VDB22, Van23]
- Core-guided search
  1. Call SAT solver to find solution under most optimistic assumptions
  2. If impossible, rewrite objective given output of SAT solver
  3. Repeat (first solution is optimal)

  VeriPB-based proof logging available [BBN+23]
- Implicit Hitting Set
  1. Call SAT solver to find solution under most optimistic assumptions
  2. Use hitting set solver (MIP solver) to recompute what most possible optimistic assumptions are
  3. Repeat (first solution is optimal)

  No proof logging available yet

Linear SAT-UNSAT Search



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

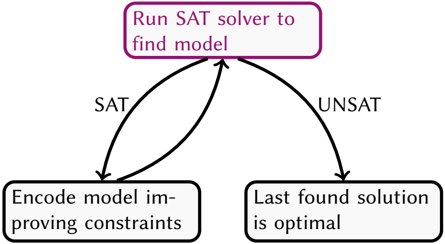
derived	justification

$\overline{x}_1 \lor x_2$ $\quad$ $\overline{x}_1 \lor \overline{x}_2 \lor r_1$

$x_1 \lor \overline{x}_2$ $\quad$ $x_1 \lor x_2 \lor r_2$

$\overline{x}_2 \lor x_3$ $\quad$ $x_2 \lor x_4 \lor r_3$

$\overline{x}_3 \lor x_4$



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification

$\overline{x}_1 \vee x_2$

$x_1 \vee \overline{x}_2$

$\overline{x}_2 \vee x_3$

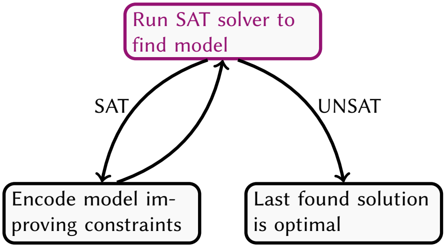
$\overline{x}_3 \vee x_4$

$\overline{x}_1 \vee \overline{x}_2 \vee r_1$

$x_1 \vee x_2 \vee r_2$

$x_2 \vee x_4 \vee r_3$

$x_2 \vee r_2$



Certified LSU Search (Example)

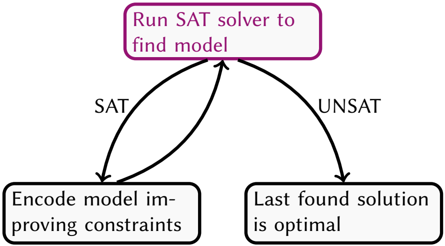
Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation

$\overline{x}_1 \vee x_2$
$x_1 \vee \overline{x}_2$
$\overline{x}_2 \vee x_3$
$\overline{x}_3 \vee x_4$

$\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee x_2 \vee r_2$
$x_2 \vee x_4 \vee r_3$
$x_2 \vee r_2$



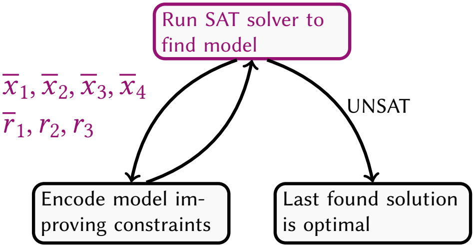
Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation

$\overline{x}_1 \vee x_2$ | $\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee \overline{x}_2$ | $x_1 \vee x_2 \vee r_2$
$\overline{x}_2 \vee x_3$ | $x_2 \vee x_4 \vee r_3$
$\overline{x}_3 \vee x_4$ | $x_2 \vee r_2$



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

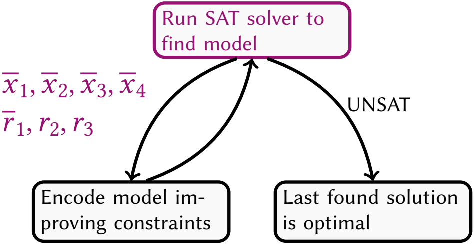
derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$



Certified LSU Search (Example)

Objective: *min* $\sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule

$\overline{x}_1 \lor x_2$

$x_1 \lor \overline{x}_2$

$\overline{x}_2 \lor x_3$

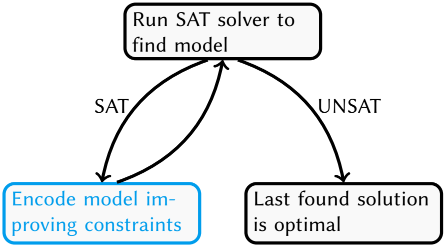
$\overline{x}_3 \lor x_4$

$\overline{x}_1 \lor \overline{x}_2 \lor r_1$

$x_1 \lor x_2 \lor r_2$

$x_2 \lor x_4 \lor r_3$

$x_2 \lor r_2$



Certified LSU Search (Example)

Objective: *min* $\sum_i r_i$

VeriPB proof:

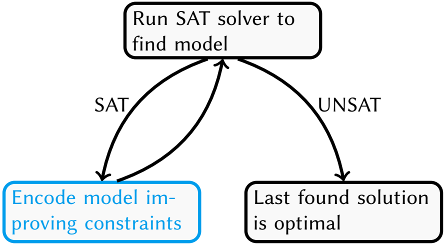
derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$\text{PB}(p_1 \Leftrightarrow (\sum_i r_i \geq 1))$	Fresh variable
$\text{PB}(p_2 \Leftrightarrow (\sum_i r_i \geq 2))$	

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

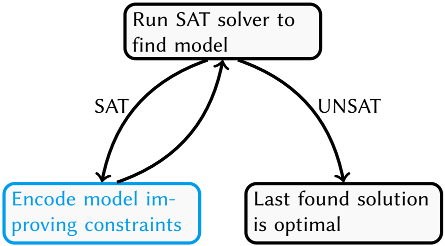
derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	

$\overline{x}_1 \vee x_2$ $\quad \overline{x}_1 \vee \overline{x}_2 \vee r_1$

$x_1 \vee \overline{x}_2$ $\quad x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $\quad x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ $\quad x_2 \vee r_2$



Certified LSU Search (Example)

Objective: *min* $\sum_i r_i$

VeriPB proof:

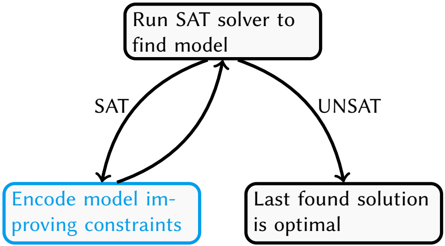
derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\text{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$



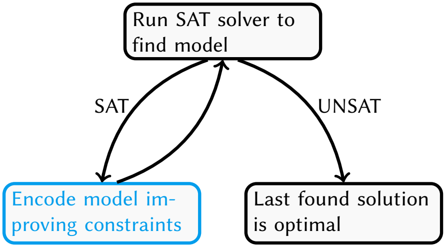
Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$
$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$
$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$



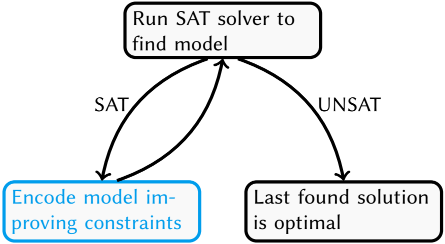
Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$
$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$
$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation

$\overline{x}_1 \vee x_2$ | $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

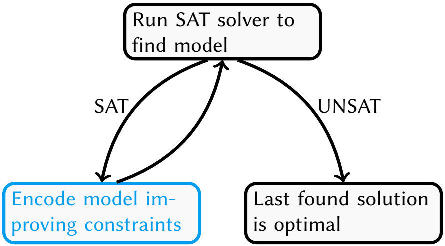
$x_1 \vee \overline{x}_2$ | $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ | $x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ | $x_2 \vee r_2$

$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$

$\overline{p}_2$



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

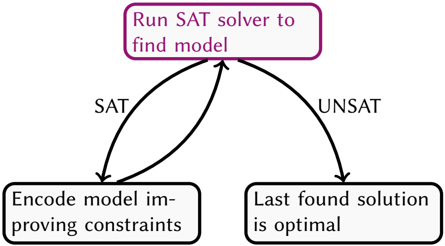
$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$

$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$

$\overline{p}_2$



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation
$x_4 \geq 1$	Reverse Unit Propagation

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

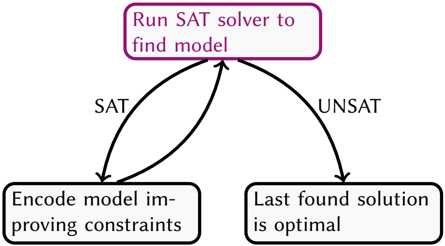
$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$

$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$

$\overline{p}_2$ $\quad$ x_4



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation
$x_4 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{r}_1, r_2, \overline{r}_3\}$	Incumbent solution

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$

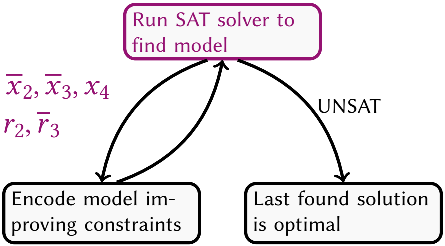
$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$

$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$

$\overline{p}_2$ $\quad$ x_4

$\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4$

$\overline{r}_1, r_2, \overline{r}_3$



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation
$x_4 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{r}_1, r_2, \overline{r}_3\}$	Incumbent solution
$\sum_i r_i \leq 0$	Objective Improvement Rule

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

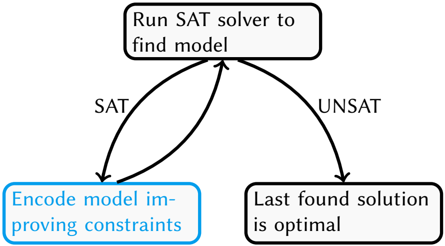
$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$

$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$

$\overline{p}_2$ $\quad$ x_4



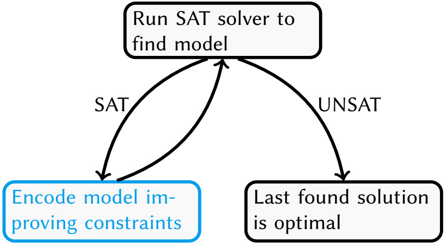
Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation
$x_4 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{r}_1, r_2, \overline{r}_3\}$	Incumbent solution
$\sum_i r_i \leq 0$	Objective Improvement Rule
$\overline{p}_1 \geq 1$	Explicit CP derivation

$\overline{x}_1 \vee x_2$	$\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee \overline{x}_2$	$x_1 \vee x_2 \vee r_2$
$\overline{x}_2 \vee x_3$	$x_2 \vee x_4 \vee r_3$
$\overline{x}_3 \vee x_4$	$x_2 \vee r_2$
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	
$\overline{p}_2$	x_4
$\overline{p}_1$	



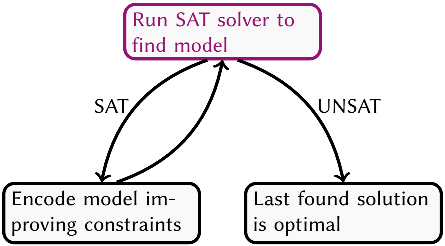
Certified LSU Search (Example)

Objective: $min \sum_i r_i$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation
$x_4 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{r}_1, r_2, \overline{r}_3\}$	Incumbent solution
$\sum_i r_i \leq 0$	Objective Improvement Rule
$\overline{p}_1 \geq 1$	Explicit CP derivation

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$
$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$
$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$
$\overline{p}_2$ $\quad$ x_4
$\overline{p}_1$



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

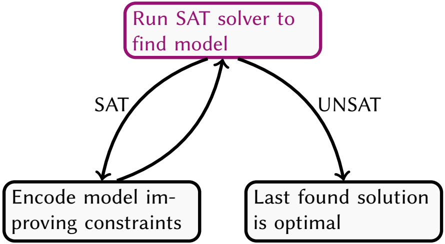
VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation
$x_4 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{r}_1, r_2, \overline{r}_3\}$	Incumbent solution
$\sum_i r_i \leq 0$	Objective Improvement Rule
$\overline{p}_1 \geq 1$	Explicit CP derivation
$0 \geq 1$	Reverse Unit Propagation

$\overline{x}_1 \lor x_2$	$\overline{x}_1 \lor \overline{x}_2 \lor r_1$
$x_1 \lor \overline{x}_2$	$x_1 \lor x_2 \lor r_2$
$\overline{x}_2 \lor x_3$	$x_2 \lor x_4 \lor r_3$
$\overline{x}_3 \lor x_4$	$x_2 \lor r_2$

$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$

$\overline{p}_2$	x_4
$\overline{p}_1$	$\bot$



Certified LSU Search (Example)

Objective: $min \sum_i r_i$

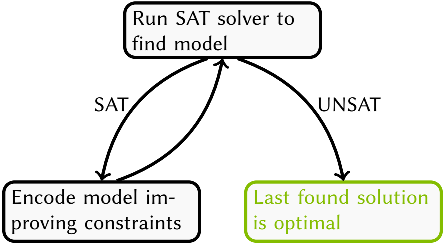
VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\}$	Incumbent solution
$\sum_i r_i \leq 1$	Objective Improvement Rule
$j \cdot \overline{p}_j + \sum_i r_i \geq j$	Fresh variable
$(4-j) \cdot p_j + \sum_i \overline{r}_i \geq 4-j$	
$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$	Explicit CP derivation
$\overline{p}_2 \geq 1$	Explicit CP derivation
$x_4 \geq 1$	Reverse Unit Propagation
$\{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{r}_1, r_2, \overline{r}_3\}$	Incumbent solution
$\sum_i r_i \leq 0$	Objective Improvement Rule
$\overline{p}_1 \geq 1$	Explicit CP derivation
$0 \geq 1$	Reverse Unit Propagation

$\overline{x}_1 \vee x_2$	$\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee \overline{x}_2$	$x_1 \vee x_2 \vee r_2$
$\overline{x}_2 \vee x_3$	$x_2 \vee x_4 \vee r_3$
$\overline{x}_3 \vee x_4$	$x_2 \vee r_2$

$\mathrm{CNF}(p_j \Leftrightarrow (\sum_i r_i \geq j))$

$\overline{p}_2$	x_4
$\overline{p}_1$	$\perp$



LSU Example in VeriPB Syntax

```
pseudo-Boolean proof version 2.0
f 7
* Clauses derived by solver
rup 1 x1 1 r2 >= 1 ;
* Log incumbent solution
soli ~x1 ~x2 ~x3 ~x4 ~r1 r2 r3
* introduce fresh variables
red 2 ~p2 1 r1 1 r2 1 r3  >= 2 ; p2 -> 0 ;
red 2 p2 1 ~r1 1 ~r2 1 ~r3  >= 2; p2 -> 1 ;
red 1 ~p1 1 r1 1 r2 1 r3  >= 1; p1 -> 0 ;
red 3 p1 1 ~r1 1 ~r2 1 ~r3 >= 3; p1 -> 1 ;
* Derive CNF encoding of totalizer
. . .  - coming soon
* Derive counter falsity
pol 9 10 + s
* Clauses derived by solver
rup 1 x4 >= 1 ;
* Log incumbent solution
soli ~x1 ~x2 ~x3 x4 ~r1 r2 ~r3
* Derive counter falsity
pol -1 12 +
* Inconsistency derived by solver
rup >= 1 ;
* Conclusion
output NONE
conclusion BOUNDS 1 1
end pseudo-Boolean proof
```

Certified Encoding of the Model-Improving Constraint

How to encode $p_j \Leftrightarrow \sum_i r_i \geq j$ in CNF?

Certified Encoding of the Model-Improving Constraint

How to encode $p_j \Leftrightarrow \sum_i r_i \geq j$ in CNF?

Different MaxSAT solvers use different PB-to-CNF encodings, e.g.,

- Totalizer Encoding [BB03]
- Binary Adder [War98]
- Modulo-Based Totalizer [OLH+13]
- Sorting Networks [ES06, ANOR09]
- (Dynamic) Polynomial Watchdog [PRB18]

Certified Encoding of the Model-Improving Constraint

How to encode $p_j \Leftrightarrow \sum_i r_i \geq j$ in CNF?

Different MaxSAT solvers use different PB-to-CNF encodings, e.g.,

- Totalizer Encoding [BB03]
- Binary Adder [War98]
- Modulo-Based Totalizer [OLH+13]
- Sorting Networks [ES06, ANOR09]
- (Dynamic) Polynomial Watchdog [PRB18]

Totalizer encoding demonstrated here; ideas generalize to other encodings [Van23]

Totalizer Encoding of Cardinality Constraints

How to encode $p_j^I \Leftrightarrow \sum_{i \in I} r_i \geq j$?

- Totalizer encoding [BB03]

Totalizer Encoding of Cardinality Constraints

How to encode $p^I_j \Leftrightarrow \sum_{i \in I} r_i \geq j$?

- Totalizer encoding [BB03]
- Create binary tree (leaves are the r_i); and introduce counter variables in all nodes

Totalizer Encoding of Cardinality Constraints

How to encode $p_j^I \Leftrightarrow \sum_{i \in I} r_i \geq j$?

- Totalizer encoding [BB03]
- Create binary tree (leaves are the r_i); and introduce counter variables in all nodes
- Example: $I = \{1, \cdots, 8\}$, $I_1 = \{1, \cdots, 4\}$ and $I_2 = \{5, \cdots, 8\}$

$$p_1^I, p_2^I, p_3^I, p_4^I, p_5^I, p_6^I, p_7^I, p_8^I$$

$$p_1^{I_1}, p_2^{I_1}, p_3^{I_1}, p_4^{I_1} \qquad\qquad p_1^{I_2}, p_2^{I_2}, p_3^{I_2}, p_4^{I_2}$$

Totalizer Encoding of Cardinality Constraints

How to encode $p^I_j \Leftrightarrow \sum_{i \in I} r_i \geq j$?

- Totalizer encoding [BB03]
- Create binary tree (leaves are the r_i); and introduce counter variables in all nodes
- Example: $I = \{1, \cdots, 8\}$, $I_1 = \{1, \cdots, 4\}$ and $I_2 = \{5, \cdots, 8\}$

$$p^I_1, p^I_2, p^I_3, p^I_4, p^I_5, p^I_6, p^I_7, p^I_8$$

$$p^{I_1}_1, p^{I_1}_2, p^{I_1}_3, p^{I_1}_4 \qquad p^{I_2}_1, p^{I_2}_2, p^{I_2}_3, p^{I_2}_4$$

Clauses encoding $p^I_6 \Leftarrow \sum_{i \in I} r_i \geq 6$:

$$\left(p^{I_1}_2 \wedge p^{I_2}_4\right) \Rightarrow p^I_6 \qquad \left(p^{I_1}_3 \wedge p^{I_2}_3\right) \Rightarrow p^I_6 \qquad \left(p^{I_1}_4 \wedge p^{I_2}_2\right) \Rightarrow p^I_6$$

Totalizer Encoding of Cardinality Constraints

How to encode $p^I_j \Leftrightarrow \sum_{i \in I} r_i \geq j$?

- Totalizer encoding [BB03]
- Create binary tree (leaves are the r_i); and introduce counter variables in all nodes
- Example: $I = \{1, \cdots, 8\}$, $I_1 = \{1, \cdots, 4\}$ and $I_2 = \{5, \cdots, 8\}$

$$p^I_1, p^I_2, p^I_3, p^I_4, p^I_5, p^I_6, p^I_7, p^I_8$$

$$p^{I_1}_1, p^{I_1}_2, p^{I_1}_3, p^{I_1}_4 \qquad p^{I_2}_1, p^{I_2}_2, p^{I_2}_3, p^{I_2}_4$$

Clauses encoding $p^I_6 \Leftarrow \sum_{i \in I} r_i \geq 6$:

$$\overline{p}^{I_1}_2 \vee \overline{p}^{I_2}_4 \vee p^I_6 \qquad \overline{p}^{I_1}_3 \vee \overline{p}^{I_2}_3 \vee p^I_6 \qquad \overline{p}^{I_1}_4 \vee \overline{p}^{I_2}_2 \vee p^I_6$$

Totalizer Encoding of Cardinality Constraints

How to encode $p^I_j \Leftrightarrow \sum_{i \in I} r_i \geq j$?

- Totalizer encoding [BB03]
- Create binary tree (leaves are the r_i); and introduce counter variables in all nodes
- Example: $I = \{1, \cdots, 8\}$, $I_1 = \{1, \cdots, 4\}$ and $I_2 = \{5, \cdots, 8\}$

$$p^I_1, p^I_2, p^I_3, p^I_4, p^I_5, p^I_6, p^I_7, p^I_8$$

$$p^{I_1}_1, p^{I_1}_2, p^{I_1}_3, p^{I_1}_4 \qquad\qquad p^{I_2}_1, p^{I_2}_2, p^{I_2}_3, p^{I_2}_4$$

Clauses encoding $p^I_6 \Leftarrow \sum_{i \in I} r_i \geq 6$:

$$\overline{p}^{I_1}_2 \vee \overline{p}^{I_2}_4 \vee p^I_6 \qquad \overline{p}^{I_1}_3 \vee \overline{p}^{I_2}_3 \vee p^I_6 \qquad \overline{p}^{I_1}_4 \vee \overline{p}^{I_2}_2 \vee p^I_6$$

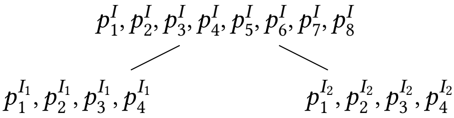
Clauses encoding $p^I_6 \Rightarrow \sum_{i \in I} r_i \geq 6$:

$$\overline{p}^{I_1}_2 \Rightarrow \overline{p}^I_6 \qquad \left(\overline{p}^{I_1}_3 \wedge \overline{p}^{I_2}_4\right) \Rightarrow \overline{p}^I_6 \qquad \left(\overline{p}^{I_1}_4 \wedge \overline{p}^{I_2}_3\right) \Rightarrow \overline{p}^I_6 \qquad \overline{p}^{I_2}_2 \Rightarrow \overline{p}^I_6$$

Totalizer Encoding of Cardinality Constraints

How to encode $p^I_j \Leftrightarrow \sum_{i \in I} r_i \geq j$?

- Totalizer encoding [BB03]
- Create binary tree (leaves are the r_i); and introduce counter variables in all nodes
- Example: $I = \{1, \cdots, 8\}$, $I_1 = \{1, \cdots, 4\}$ and $I_2 = \{5, \cdots, 8\}$



Clauses encoding $p^I_6 \Leftarrow \sum_{i \in I} r_i \geq 6$:

$$\overline{p}^{I_1}_2 \vee \overline{p}^{I_2}_4 \vee p^I_6 \qquad \overline{p}^{I_1}_3 \vee \overline{p}^{I_2}_3 \vee p^I_6 \qquad \overline{p}^{I_1}_4 \vee \overline{p}^{I_2}_2 \vee p^I_6$$

Clauses encoding $p^I_6 \Rightarrow \sum_{i \in I} r_i \geq 6$:

$$p^{I_1}_2 \vee \overline{p}^I_6 \qquad p^{I_1}_3 \vee p^{I_2}_4 \vee \overline{p}^I_6 \qquad p^{I_1}_4 \vee p^{I_2}_3 \vee \overline{p}^I_6 \qquad p^{I_2}_2 \vee \overline{p}^I_6$$

Certifying the Totalizer encoding using cutting planes

- To be derived: $\overline{p}_4^{I_1} \vee \overline{p}_2^{I_2} \vee p_6^{I}$

Certifying the Totalizer encoding using cutting planes

- To be derived: $\overline{p}_4^{I_1} \lor \overline{p}_2^{I_2} \lor p_6^{I}$
- Counting variables introduced using

$$4 \cdot \overline{p}_4^{I_1} + \sum_{i \in I_1} r_i \geq 4$$

$$2 \cdot \overline{p}_2^{I_2} + \sum_{i \in I_2} r_i \geq 2$$

$$3 \cdot p_6^{I} + \sum_{i \in I} \overline{r}_i \geq 3$$

Certifying the Totalizer encoding using cutting planes

- To be derived: $\overline{p}_4^{I_1} \vee \overline{p}_2^{I_2} \vee p_6^{I}$
- Counting variables introduced using

$$4 \cdot \overline{p}_4^{I_1} + \sum_{i \in I_1} r_i \geq 4$$

$$2 \cdot \overline{p}_2^{I_2} + \sum_{i \in I_2} r_i \geq 2$$

$$3 \cdot p_6^{I} + \sum_{i \in I} \overline{r}_i \geq 3$$

- Adding these three constraints yields

$$4 \cdot \overline{p}_4^{I_1} + 2 \cdot \overline{p}_2^{I_2} + 3 \cdot p_6^{I} + 8 \geq 9$$

Certifying the Totalizer encoding using cutting planes

- To be derived: $\overline{p}_4^{I_1} \vee \overline{p}_2^{I_2} \vee p_6^{I}$
- Counting variables introduced using

$$4 \cdot \overline{p}_4^{I_1} + \sum_{i \in I_1} r_i \geq 4$$

$$2 \cdot \overline{p}_2^{I_2} + \sum_{i \in I_2} r_i \geq 2$$

$$3 \cdot p_6^{I} + \sum_{i \in I} \overline{r}_i \geq 3$$

- Adding these three constraints yields

$$4 \cdot \overline{p}_4^{I_1} + 2 \cdot \overline{p}_2^{I_2} + 3 \cdot p_6^{I} + \not{8} \geq \not{9}\ 1$$

Certifying the Totalizer encoding using cutting planes

- To be derived: $\overline{p}_4^{I_1} \vee \overline{p}_2^{I_2} \vee p_6^{I}$
- Counting variables introduced using

$$4 \cdot \overline{p}_4^{I_1} + \sum_{i \in I_1} r_i \geq 4$$

$$2 \cdot \overline{p}_2^{I_2} + \sum_{i \in I_2} r_i \geq 2$$

$$3 \cdot p_6^{I} + \sum_{i \in I} \overline{r}_i \geq 3$$

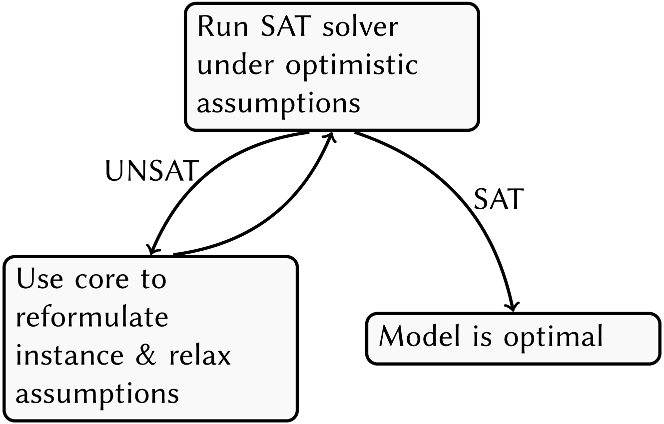
- Adding these three constraints and saturating yields

$$\cancel{4} \cdot \overline{p}_4^{I_1} + \cancel{2} \cdot \overline{p}_2^{I_2} + \cancel{3} \cdot p_6^{I} + \cancel{8} \geq \cancel{9}\ 1$$

Complete LSU Example in VeriPB Syntax

```
pseudo-Boolean proof version 2.0
f 7
* Clauses derived by solver
rup 1 x1 1 r2 >= 1 ;
* Log incumbent solution
soli ~x1 ~x2 ~x3 ~x4 ~r1 r2 r3
* introduce fresh variables
red 2 ~p2 1 r1 1 r2 1 r3  >= 2 ; p2 -> 0 ;
red 2 p2 1 ~r1 1 ~r2 1 ~r3  >= 2; p2 -> 1 ;
red 1 ~p1 1 r1 1 r2 1 r3  >= 1; p1 -> 0 ;
red 3 p1 1 ~r1 1 ~r2 1 ~r3 >= 3; p1 -> 1 ;
* Auxiliary variables for CNF encoding
red 2 ~p_1-2_2 1 r1 1 r2  >= 2 ; p_1-2_2 -> 0 ;
red 1 p_1-2_2 1 ~r1 1 ~r2 >= 1; p_1-2_2 -> 1 ;
red 1 ~p_1-2_1 1 r1 1 r2  >= 1; p_1-2_1 -> 0 ;
red 2 p_1-2_1 1 ~r1 1 ~r2 >= 2; p_1-2_1 -> 1 ;
* Cutting planes derivation of totalizer clauses
pol 10 15 + s
pol 10 17 + ~r3 + s
pol 11 14 + r3 + s
pol 11 16 + s
pol 12 17 + s
pol 13 16 + r3 + s
pol 13 r1 + r2 + s
* Derive counter falsity
pol 9 10 + s
* Clauses derived by solver
rup 1 x4 >= 1 ;
* Log incumbent solution
soli ~x1 ~x2 ~x3 x4 ~r1 r2 ~r3
* Derive counter falsity
pol -1 12 +
* Inconsistency derived by solver
rup >= 1 ;
* Conclusion
output NONE
conclusion BOUNDS 1 1
end pseudo-Boolean proof
```

Core-Guided Search



Certified Core-Guided Search (Example)

$\overline{x}_1 \lor x_2$ $\quad$ $\overline{x}_1 \lor \overline{x}_2 \lor r_1$

$x_1 \lor \overline{x}_2$ $\quad$ $x_1 \lor x_2 \lor r_2$

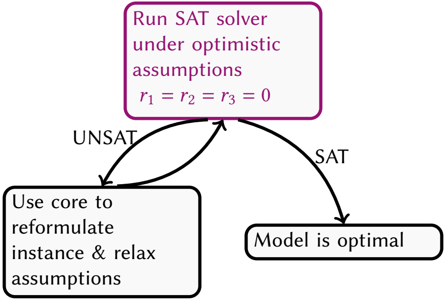
$\overline{x}_2 \lor x_3$ $\quad$ $x_2 \lor x_4 \lor r_3$

$\overline{x}_3 \lor x_4$

Objective (*min*): $r_1 + r_2 + r_3$

VeriPB proof:

derived	justification



Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation

$\overline{x}_1 \lor x_2$

$x_1 \lor \overline{x}_2$

$\overline{x}_2 \lor x_3$

$\overline{x}_3 \lor x_4$

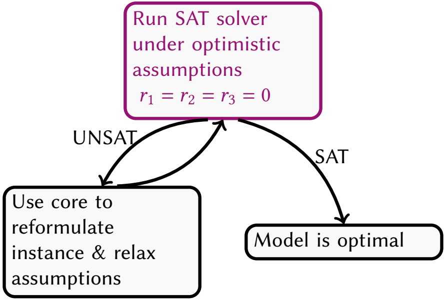
$r_1 \lor r_2$

$\overline{x}_1 \lor \overline{x}_2 \lor r_1$

$x_1 \lor x_2 \lor r_2$

$x_2 \lor x_4 \lor r_3$

$x_2 \lor r_2$



Certified Core-Guided Search (Example)

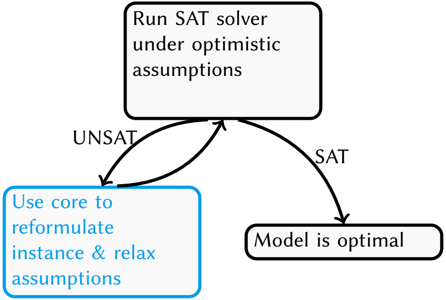
Objective (*min*): $r_1 + r_2 + r_3$

$\overline{x}_1 \vee x_2$
$x_1 \vee \overline{x}_2$
$\overline{x}_2 \vee x_3$
$\overline{x}_3 \vee x_4$
$r_1 \vee r_2$

$\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee x_2 \vee r_2$
$x_2 \vee x_4 \vee r_3$
$x_2 \vee r_2$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$p_2 \Leftrightarrow (r_1 + r_2 \geq 2)$	Fresh variable



Certified Core-Guided Search (Example)

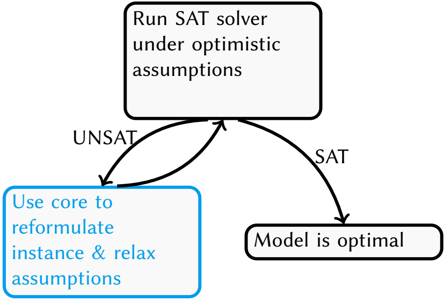
$\overline{x}_1 \vee x_2$
$x_1 \vee \overline{x}_2$
$\overline{x}_2 \vee x_3$
$\overline{x}_3 \vee x_4$
$r_1 \vee r_2$

$\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee x_2 \vee r_2$
$x_2 \vee x_4 \vee r_3$
$x_2 \vee r_2$

Objective (*min*): $r_1 + r_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	



Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation

$\overline{x}_1 \vee x_2$

$x_1 \vee \overline{x}_2$

$\overline{x}_2 \vee x_3$

$\overline{x}_3 \vee x_4$

$r_1 \vee r_2$

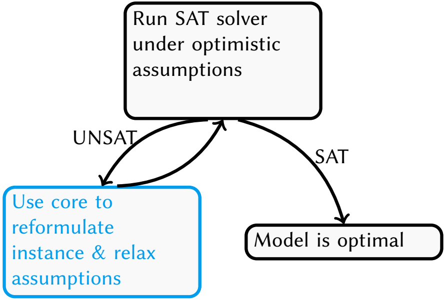
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$

$\overline{x}_1 \vee \overline{x}_2 \vee r_1$

$x_1 \vee x_2 \vee r_2$

$x_2 \vee x_4 \vee r_3$

$x_2 \vee r_2$



Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation

$\overline{x}_1 \vee x_2$

$x_1 \vee \overline{x}_2$

$\overline{x}_2 \vee x_3$

$\overline{x}_3 \vee x_4$

$r_1 \vee r_2$

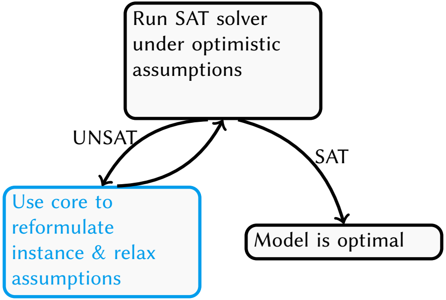
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$

$\overline{x}_1 \vee \overline{x}_2 \vee r_1$

$x_1 \vee x_2 \vee r_2$

$x_2 \vee x_4 \vee r_3$

$x_2 \vee r_2$



Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation

$\overline{x}_1 \vee x_2$ $\quad$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

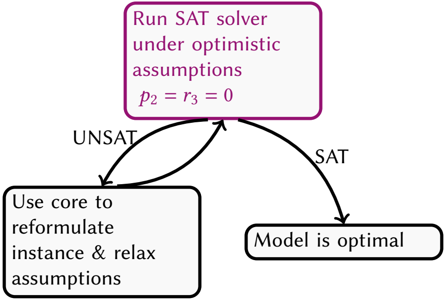
$x_1 \vee \overline{x}_2$ $\quad$ $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $\quad$ $x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ $\quad$ $x_2 \vee r_2$

$r_1 \vee r_2$

$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$



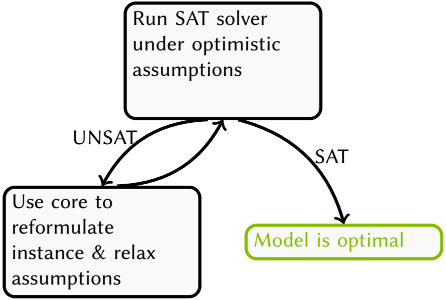
Certified Core-Guided Search (Example)

$\overline{x}_1 \vee x_2$	$\overline{x}_1 \vee \overline{x}_2 \vee r_1$
$x_1 \vee \overline{x}_2$	$x_1 \vee x_2 \vee r_2$
$\overline{x}_2 \vee x_3$	$x_2 \vee x_4 \vee r_3$
$\overline{x}_3 \vee x_4$	$x_2 \vee r_2$
$r_1 \vee r_2$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation
$\{x_1, x_2, x_3, x_4, r_1, \overline{r}_2, \overline{r}_3\}$	Solution



Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation
$\{x_1, x_2, x_3, x_4, r_1, \overline{r}_2, \overline{r}_3\}$	Solution
$\overline{r}_1 + \overline{r}_2 + \overline{r}_3 \geq 3$	Objective Improvement

$\overline{x}_1 \lor x_2$

$x_1 \lor \overline{x}_2$

$\overline{x}_2 \lor x_3$

$\overline{x}_3 \lor x_4$

$r_1 \lor r_2$

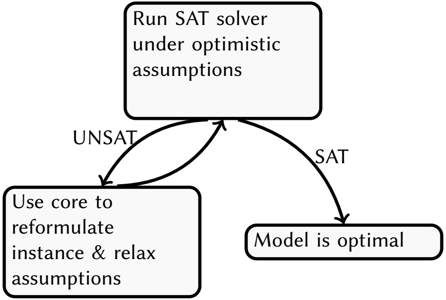
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$

$\overline{x}_1 \lor \overline{x}_2 \lor r_1$

$x_1 \lor x_2 \lor r_2$

$x_2 \lor x_4 \lor r_3$

$x_2 \lor r_2$



Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation
$\{x_1, x_2, x_3, x_4, r_1, \overline{r}_2, \overline{r}_3\}$	Solution
$\overline{r}_1 + \overline{r}_2 + \overline{r}_3 \geq 3$	Objective Improvement
$0 \geq 1$	Explicit CP derivation

$\overline{x}_1 \vee x_2$ $\overline{x}_1 \vee \overline{x}_2 \vee r_1$

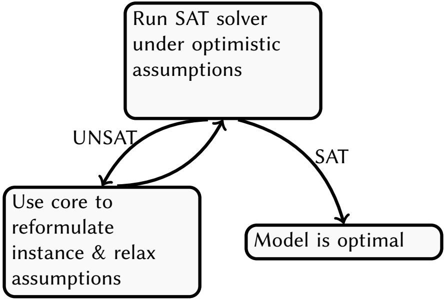
$x_1 \vee \overline{x}_2$ $x_1 \vee x_2 \vee r_2$

$\overline{x}_2 \vee x_3$ $x_2 \vee x_4 \vee r_3$

$\overline{x}_3 \vee x_4$ $x_2 \vee r_2$

$r_1 \vee r_2$

$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$



Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation
$\{x_1, x_2, x_3, x_4, r_1, \overline{r}_2, \overline{r}_3\}$	Solution
$\overline{r}_1 + \overline{r}_2 + \overline{r}_3 \geq 3$	Objective Improvement
$0 \geq 1$	Explicit CP derivation

Explicit CP derivations:

CNF encoding (totalizer): see part on LSU

Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation
$\{x_1, x_2, x_3, x_4, r_1, \overline{r}_2, \overline{r}_3\}$	Solution
$\overline{r}_1 + \overline{r}_2 + \overline{r}_3 \geq 3$	Objective Improvement
$0 \geq 1$	Explicit CP derivation

Explicit CP derivations:

CNF encoding (totalizer): see part on LSU

Adding up definition of p_2 and core constraint yields

$$2 \cdot \overline{p}_2 + 2 \cdot r_1 + 2 \cdot r_2 \geq 3 \;.$$

Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation
$\{x_1, x_2, x_3, x_4, r_1, \overline{r}_2, \overline{r}_3\}$	Solution
$\overline{r}_1 + \overline{r}_2 + \overline{r}_3 \geq 3$	Objective Improvement
$0 \geq 1$	Explicit CP derivation

Explicit CP derivations:

CNF encoding (totalizer): see part on LSU

Adding up definition of p_2 and core constraint and dividing by 2 yields

$$\cancel{2} \cdot \overline{p}_2 + \cancel{2} \cdot r_1 + \cancel{2} \cdot r_2 \geq \cancel{3}2.$$

Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation
$\{x_1, x_2, x_3, x_4, r_1, \overline{r}_2, \overline{r}_3\}$	Solution
$\overline{r}_1 + \overline{r}_2 + \overline{r}_3 \geq 3$	Objective Improvement
$0 \geq 1$	Explicit CP derivation

Explicit CP derivations:

CNF encoding (totalizer): see part on LSU

Adding up definition of p_2 and core constraint and dividing by 2 yields

$$\cancel{2} \cdot \overline{p}_2 + \cancel{2} \cdot r_1 + \cancel{2} \cdot r_2 \geq \cancel{3}2.$$

which is the same as $r_1 + r_2 \geq 1 + p_2$.
Other direction already given

Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = \mathbf{1} + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\mathrm{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation
$\{x_1, x_2, x_3, x_4, r_1, \overline{r}_2, \overline{r}_3\}$	Solution
$\overline{r}_1 + \overline{r}_2 + \overline{r}_3 \geq 3$	Objective Improvement
$0 \geq 1$	Explicit CP derivation

Explicit CP derivations:

CNF encoding (totalizer): see part on LSU

Adding up definition of p_2 and core constraint and dividing by 2 yields

$$\cancel{2} \cdot \overline{p}_2 + \cancel{2} \cdot r_1 + \cancel{2} \cdot r_2 \geq \cancel{3}2.$$

which is the same as $r_1 + r_2 \geq 1 + p_2$. Other direction already given

Previously derived cores guarantee that objective is at least 1:
$r_1 + r_2 \ (+ r_3) \geq 1$

Certified Core-Guided Search (Example)

Objective (*min*): $r_1 + r_2 + r_3 = 1 + p_2 + r_3$

VeriPB proof:

derived	justification
$x_2 + r_2 \geq 1$	Reverse Unit Propagation
Core returned by solver:	
$r_1 + r_2 \geq 1$	Reverse Unit Propagation
$2 \cdot \overline{p}_2 + r_1 + r_2 \geq 2$	Fresh variable
$p_2 + \overline{r}_1 + \overline{r}_2 \geq 1$	
$\text{CNF}(p_2 \Leftrightarrow (r_1 + r_2 \geq 2))$	Explicit CP derivation
$r_1 + r_2 = 1 + p_2$	Explicit CP derivation
$\{x_1, x_2, x_3, x_4, r_1, \overline{r}_2, \overline{r}_3\}$	Solution
$\overline{r}_1 + \overline{r}_2 + \overline{r}_3 \geq 3$	Objective Improvement
$0 \geq 1$	Explicit CP derivation

Explicit CP derivations:

CNF encoding (totalizer): see part on LSU

Adding up definition of p_2 and core constraint and dividing by 2 yields

$$\cancel{2} \cdot \overline{p}_2 + \cancel{2} \cdot r_1 + \cancel{2} \cdot r_2 \geq \cancel{3}2.$$

which is the same as $r_1 + r_2 \geq 1 + p_2$.
Other direction already given

Previously derived cores guarantee that objective is at least 1:
$r_1 + r_2 \ (+ r_3) \geq 1$
Adding this to objective improvement constraint gives contradiction

Complete CG Example in VeriPB Syntax

```
pseudo-Boolean proof version 2.0
f 7
* Clauses derived by solver (inc core)
rup 1 x1 1 r2 >= 1 ;
rup 1 r1 1 r2 >= 1 ;
* Introduce fresh variable
red 2 ~p2 1 r1 1 r2  >= 2 ; p2 -> 0 ;
red 1 p2 1 ~r1 1 ~r2 >= 1; p2 -> 1 ;
* Encode this in CNF
pol 10 ~r1 +
pol 10 ~r2 +
* Rewriting the objective
pol 9 10 + 2 d
* Check that we have indeed
*   derived that r1 + r2 = 1 + p2
e 14 : 1 r1 1 r2 -1 p2 >= 1 ;
e 11 : -1 r1 -1 r2 1 p2 >= -1 ;
```

```
* Solution found
soli x1 x2 x3 x4 r1 ~r2 ~r3
* Prove optimality of solution:
pol -1 9 +
ia -1 : >= 1 ;
* Conclusion
output NONE
conclusion BOUNDS 1 1
end pseudo-Boolean proof
```

Advanced Techniques for Core-Guided MaxSAT

- Important to deal with all state-of-the-art solver techniques

Advanced Techniques for Core-Guided MaxSAT

- Important to deal with all state-of-the-art solver techniques
- Additional techniques that are skipped in this example
  - Intrinsic at-most-one constraints [IMM19]

Advanced Techniques for Core-Guided MaxSAT

- Important to deal with all state-of-the-art solver techniques
- Additional techniques that are skipped in this example
  - Intrinsic at-most-one constraints [IMM19]
  - Hardening [ABGL12]

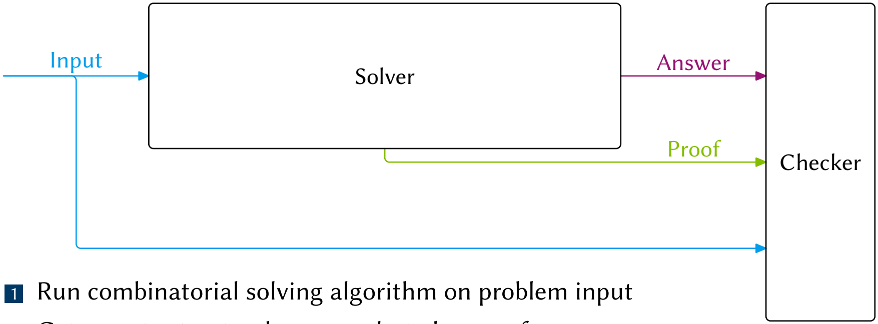
Advanced Techniques for Core-Guided MaxSAT

- Important to deal with all state-of-the-art solver techniques
- Additional techniques that are skipped in this example
  - Intrinsic at-most-one constraints [IMM19]
  - Hardening [ABGL12]
  - Lazy counter variables [MJML14]

Advanced Techniques for Core-Guided MaxSAT

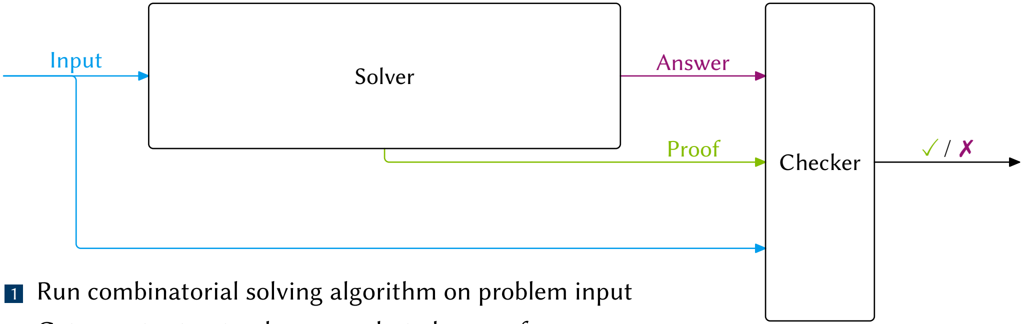
- Important to deal with all state-of-the-art solver techniques
- Additional techniques that are skipped in this example
  - Intrinsic at-most-one constraints [IMM19]
  - Hardening [ABGL12]
  - Lazy counter variables [MJML14]
- VeriPB Proof logging also convenient for these techniques [BBN+23]

Recap (1/2)



1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with input

Recap (1/2)



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

Recap (2/2)

Proof logging implementation

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

Performance goals

Want linear(ish) scaling in terms of solver running time for

- proof size
- proof checking time

Progress So Far

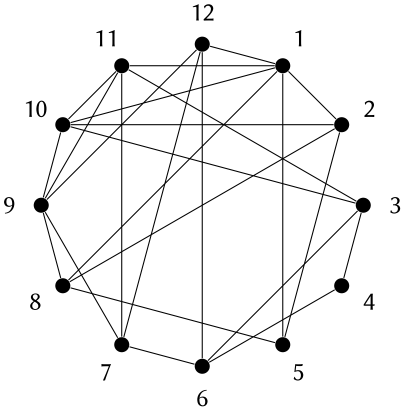
We've seen proof logging, and how it works for SAT

We've learned about

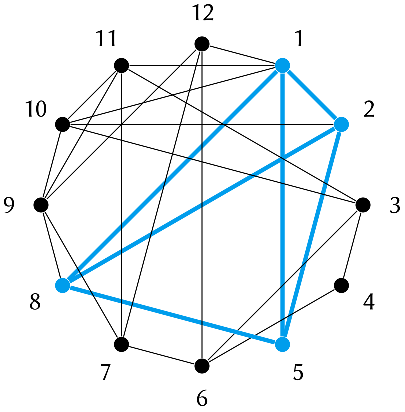
- pseudo-Boolean constraints (0–1 linear inequalities)
- cutting planes reasoning
- VeriPB

Coming next, some worked examples from dedicated graph solvers

The Maximum Clique Problem



The Maximum Clique Problem



Maximum Clique Solvers

There are a lot of dedicated solvers for clique problems

But there are issues:

- “State-of-the-art” solvers have been buggy.
- Often undetected: error rate of around 0.1 [MPP19]

Often used inside other solvers

- An off-by-one result can cause much larger errors

A Brief and Incomplete Guide to Clique Solving (1/4)

Recursive maximum clique algorithm:

- Pick a vertex v
- Either v is in the clique…
  - Throw away every vertex not adjacent to v
  - If vertices remain, recurse
- …or v is not in the clique
  - Throw v away and pick another vertex

A Brief and Incomplete Guide to Clique Solving (2/4)

Key data structures:

- Growing clique C
- Set of potential vertices P
  - All the vertices we haven't thrown away yet
  - Every $v \in P$ is adjacent to every $w \in C$

A Brief and Incomplete Guide to Clique Solving (2/4)

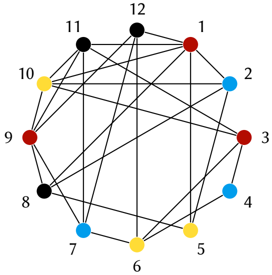
Key data structures:

- Growing clique C
- Set of potential vertices P
  - All the vertices we haven't thrown away yet
  - Every $v \in P$ is adjacent to every $w \in C$

Branch and bound:

- Remember the biggest clique $C^\star$ found so far
- If $|C| + |P| \leq \left|C^\star\right|$, no need to keep going

A Brief and Incomplete Guide to Clique Solving (3/4)



Given a k-colouring of a subgraph, that subgraph cannot have a clique of more than k vertices

We can use $|C| + \#colours(P)$ as a bound, for any colouring

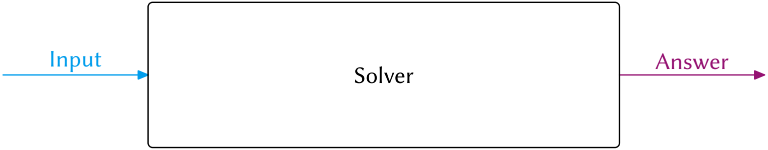
A Brief and Incomplete Guide to Clique Solving (4/4)

- This brings us to 1997
- Many improvements since then
  - better bound functions
  - clever vertex selection heuristics
  - efficient data structures
  - local search
  - ...
- But key ideas for proof logging can be explained without worrying about such things

Making a Proof Logging Clique Solver

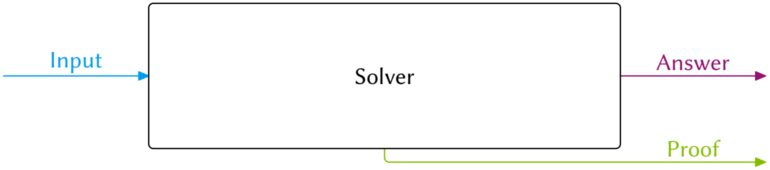
1. Output a pseudo-Boolean encoding of the problem
   - Clique problems have several standard file formats
2. Make the solver log its search tree
   - Output a small header
   - Output something on every backtrack
   - Output something every time a solution is found
   - Output a small footer
3. Figure out how to log the bound function

A Slightly Different Proof Logging Workflow



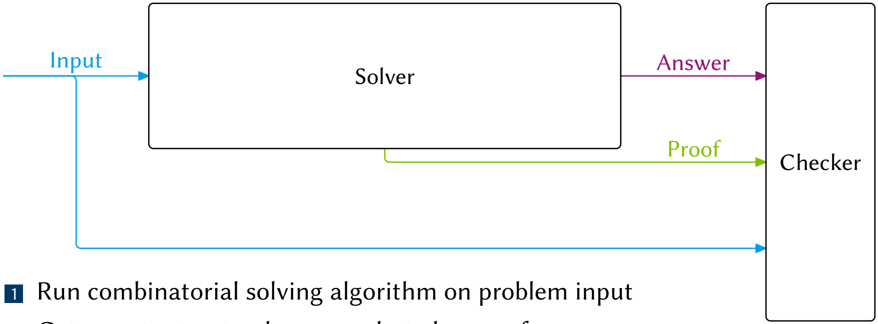
1. Run combinatorial solving algorithm on problem input

A Slightly Different Proof Logging Workflow



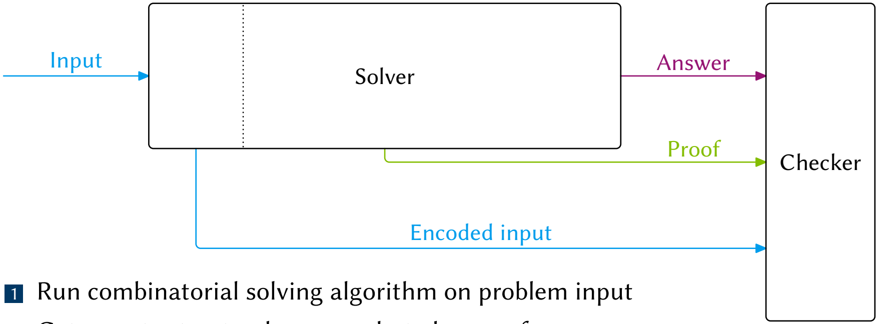
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

A Slightly Different Proof Logging Workflow



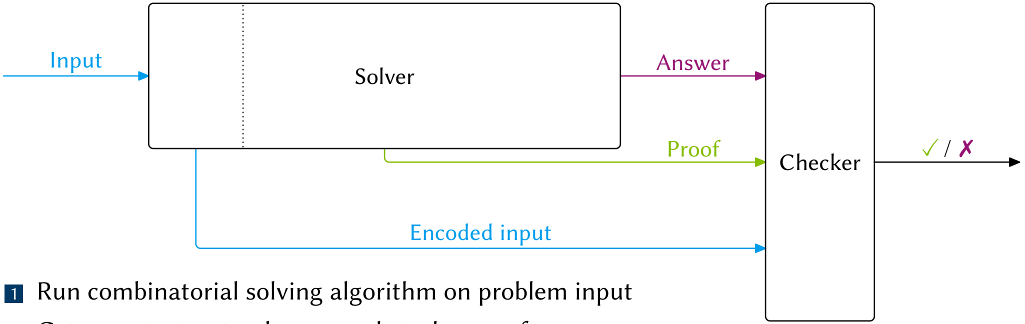
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with input

A Slightly Different Proof Logging Workflow



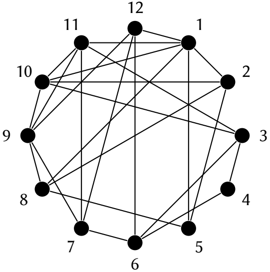
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with 0–1 ILP encoding of input

A Slightly Different Proof Logging Workflow



1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with 0–1 ILP encoding of input
4. Verify that proof checker says answer is correct

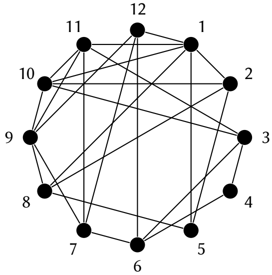
A Pseudo-Boolean Encoding for Clique (in OPB Format)




```
* #variable= 12 #constraint= 41
min: -1 x1 -1 x2 -1 x3 -1 x4 . . . and so on. . . -1 x11 -1 x12 ;
1 ~x3 1 ~x1 >= 1 ;
1 ~x3 1 ~x2 >= 1 ;
1 ~x4 1 ~x1 >= 1 ;
* . . . and a further 38 similar lines for the remaining non-edges
```

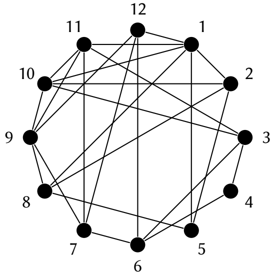
First Attempt at a Proof

```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



First Attempt at a Proof

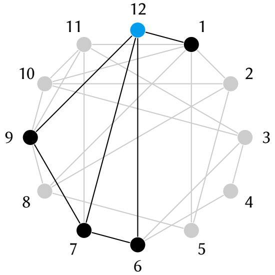
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Start with a header
Load the 41 problem axioms

First Attempt at a Proof

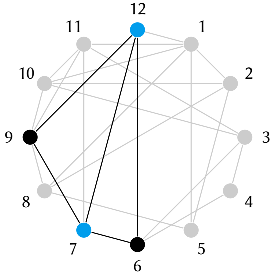
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Branch accepting 12
Throw away non-adjacent vertices

First Attempt at a Proof

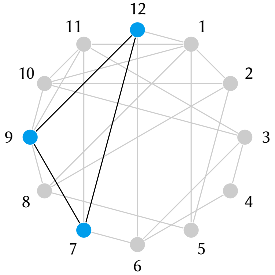
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Branch also accepting 7
Throw away non-adjacent vertices

First Attempt at a Proof

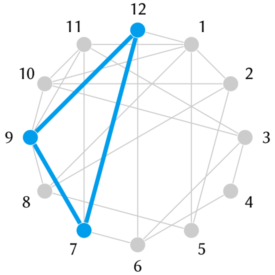
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Branch also accepting 9
Throw away non-adjacent vertices

First Attempt at a Proof

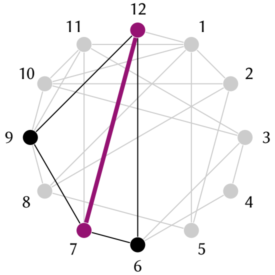
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



We branched on 12, 7, 9
Found a new incumbent
Now looking for a ≥ 4 vertex clique

First Attempt at a Proof

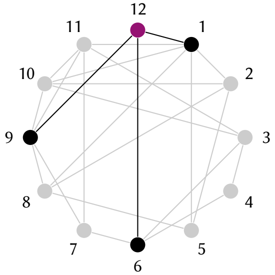
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Backtrack from 12, 7
9 explored already, only 6 feasible
No ≥ 4 vertex clique possible
Effectively this deletes the 7–12 edge

First Attempt at a Proof

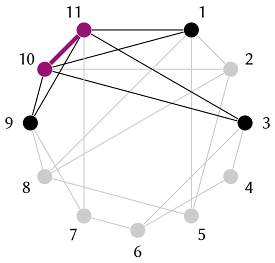
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Backtrack from 12
Only 1, 6 and 9 feasible (1-colourable)
No ≥ 4 vertex clique possible
Effectively this deletes vertex 12

First Attempt at a Proof

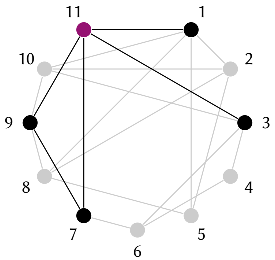
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Branch on 11 then 10
Only 1, 3 and 9 feasible (1-colourable)
No ≥ 4 vertex clique possible
Backtrack, deleting the edge

First Attempt at a Proof

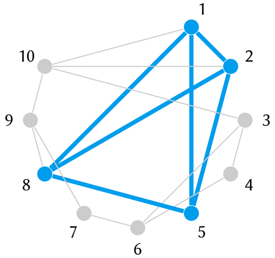
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Backtrack from 11
2-colourable, so no ≥ 4 clique
Delete the vertex

First Attempt at a Proof

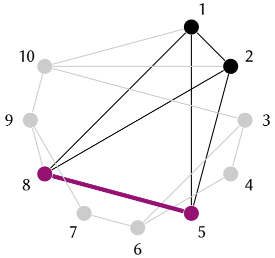
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Branch on 8, 5, 1, 2
Find a new incumbent
Now looking for a ≥ 5 vertex clique

First Attempt at a Proof

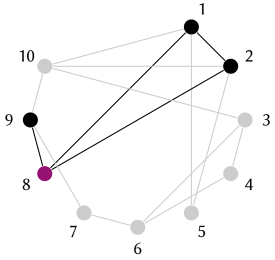
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Backtrack from 8, 5
Only 4 vertices; can't have a ≥ 5 clique
Delete the edge

First Attempt at a Proof

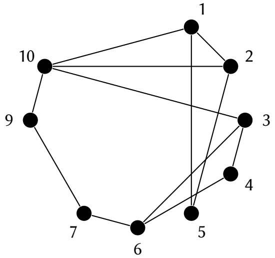
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Backtrack from 8
Still not enough vertices
Delete the vertex

First Attempt at a Proof

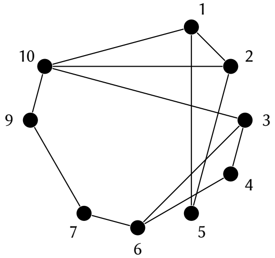
```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



Remaining graph is 3-colourable
Backtrack from root node

First Attempt at a Proof

```
pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
```



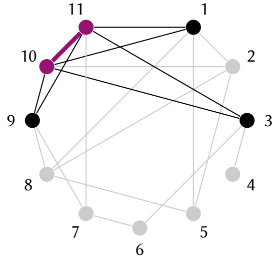
Finish with what we've concluded
We specify a lower and an upper bound
Remember we're minimising $\sum_v -1 \times v$, so a 4-clique has an objective value of -4

Verifying This Proof (Or Not...)

```
$ veripb clique.opb clique-attempt-one.veripb
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
```

Verifying This Proof (Or Not. . .)

```
$ veripb clique.opb clique-attempt-one.veripb
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
```



Verifying This Proof (Or Not…)

```
$ veripb --trace clique.opb clique-attempt-one.veripb
line 002: f 41
  ConstraintId 001: 1 ~x1 1 ~x3 >= 1
  ConstraintId 002: 1 ~x2 1 ~x3 >= 1
...
  ConstraintId 041: 1 ~x11 1 ~x12 >= 1
line 003: soli x7 x9 x12 ~x1 ~x2 ~x3 ~x4 ~x5 ~x6 ~x8 ~x10 ~x11
  ConstraintId 042: 1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 >= 4
line 004: rup 1 ~x12 1 ~x7 >= 1 ;
  ConstraintId 043: 1 ~x7 1 ~x12 >= 1
line 005: rup 1 ~x12 >= 1 ;
  ConstraintId 044: 1 ~x12 >= 1
line 006: rup 1 ~x11 1 ~x10 >= 1 ;
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
```

Dealing With Colourings

The colour bound doesn't follow by RUP…

But we can lazily recover at-most-one constraints for each colour class!

Dealing With Colourings

The colour bound doesn't follow by RUP…

But we can lazily recover at-most-one constraints for each colour class!

$$
\begin{aligned}
&(\overline{x}_1 + \overline{x}_6 \geq 1) && \\
+&(\overline{x}_1 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
+&(\overline{x}_6 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3 \\
&/\,2 && = \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
& && \text{i.e. } x_1 + x_6 + x_9 \leq 1
\end{aligned}
$$

Dealing With Colourings

The colour bound doesn't follow by RUP…

But we can lazily recover at-most-one constraints for each colour class!

$$
\begin{aligned}
(\overline{x}_1 + \overline{x}_6 \geq 1) & \\
+ (\overline{x}_1 + \overline{x}_9 \geq 1) & \quad = 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
+ (\overline{x}_6 + \overline{x}_9 \geq 1) & \quad = 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3 \\
/\, 2 & \quad = \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
& \quad \text{i.e. } x_1 + x_6 + x_9 \leq 1
\end{aligned}
$$

This generalises to colour classes of any size v

- Each non-edge is used exactly once, $v(v-1)$ additions
- $v-3$ multiplications and $v-2$ divisions

Solvers don't need to "understand" cutting planes to write this derivation to proof log

What This Looks Like in the Proof Log

pseudo-Boolean proof version 2.0
f 41
soli x12 x7 x9
rup 1 ~x12 1 ~x7 >= 1 ;
* bound, colour classes [x1 x6 x9]
pol $7_{1 \not\sim 6}$ $19_{1 \not\sim 9}$ + $24_{6 \not\sim 9}$ + 2 d
pol 42_{obj} -1 +
rup 1 ~x12 >= 1 ;
* bound, colour classes [x1 x3 x9]
pol $1_{1 \not\sim 3}$ $19_{1 \not\sim 9}$ + $21_{3 \not\sim 9}$ + 2 d
pol 42_{obj} -1 +
rup 1 ~x11 1 ~x10 >= 1 ;
* bound, colour classes [x1 x3 x7]
* [x9]
pol $1_{1 \not\sim 3}$ $10_{1 \not\sim 7}$ + $12_{3 \not\sim 7}$ + 2 d
pol 42_{obj} -1 +
rup 1 ~x11 >= 1 ;
soli x8 x5 x2 x1
rup 1 ~x8 1 ~x5 >= 1 ;
* bound, colour classes [x1 x9] [x2]
pol 53_{obj} $19_{1 \not\sim 9}$ +
rup 1 ~x8 >= 1 ;
* bound, colour classes [x1 x3 x7]
* [x2 x4 x9] [x5 x6 x10]
pol $1_{1 \not\sim 3}$ $10_{1 \not\sim 7}$ + $12_{3 \not\sim 7}$ + 2 d
pol 53_{obj} -1 +
pol $4_{2 \not\sim 4}$ $20_{2 \not\sim 9}$ + $22_{4 \not\sim 9}$ + 2 d
pol 53_{obj} -3 + -1 +
pol $9_{5 \not\sim 6}$ $26_{5 \not\sim 10}$ + $27_{6 \not\sim 10}$ + 2 d
pol 53_{obj} -5 + -3 + -1 +
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

Verifying This Proof (For Real, This Time)

```
$ veripb --trace clique.opb clique-attempt-two.veripb
=== begin trace ===
line 002: f 41
  ConstraintId 001: 1 ~x1 1 ~x3 >= 1
  ConstraintId 002: 1 ~x2 1 ~x3 >= 1
...
  ConstraintId 041: 1 ~x11 1 ~x12 >= 1
line 003: soli x7 x9 x12 ~x1 ~x2 ~x3 ~x4 ~x5 ~x6 ~x8 ~x10 ~x11
  ConstraintId 042: 1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 >= 4
line 004: rup 1 ~x12 1 ~x7 >= 1 ;
  ConstraintId 043: 1 ~x7 1 ~x12 >= 1
line 005: * bound, colour classes [ x1 x6 x9 ]
line 006: pol 7 19 + 24 + 2 d
  ConstraintId 044: 1 ~x1 1 ~x6 1 ~x9 >= 2
line 007: pol 42 43 +
  ConstraintId 045: 1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x8 1 x9 1 x10 1 x11 >= 3
...
  ConstraintId 061: 1 ~x5 1 ~x6 1 ~x10 >= 2
line 028: pol 53 57 + 59 + 61 +
  ConstraintId 062: 1 x8 1 x11 1 x12 >= 2
line 029: rup >= 1 ;
  ConstraintId 063: >= 1
line 030: output NONE
line 031: conclusion BOUNDS -4 -4
line 032: end pseudo-Boolean proof
=== end trace ===

Verification succeeded.
```

Different Clique Algorithms

Different search orders?

- ✓ Irrelevant for proof logging

Using local search to initialise?

- ✓ Just log the incumbent

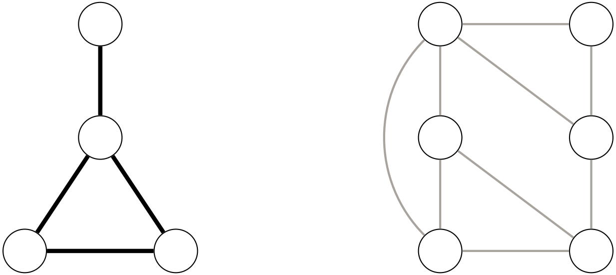
Different bound functions?

- Is cutting planes strong enough to justify every useful bound function ever invented?
- So far, seems like it…

Weighted cliques?

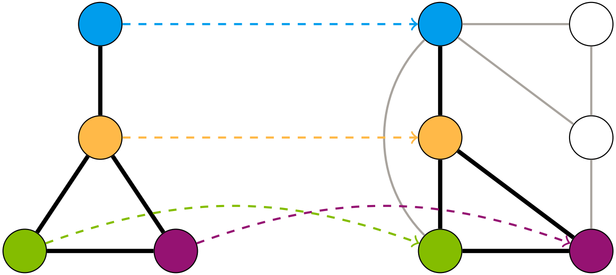
- ✓ Multiply a colour class by its largest weight
- ✓ Also works for vertices "split between colour classes"

Subgraph Isomorphism



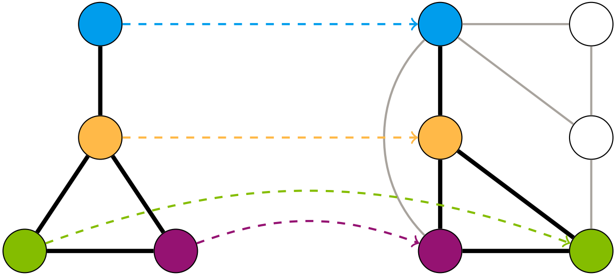
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

Subgraph Isomorphism



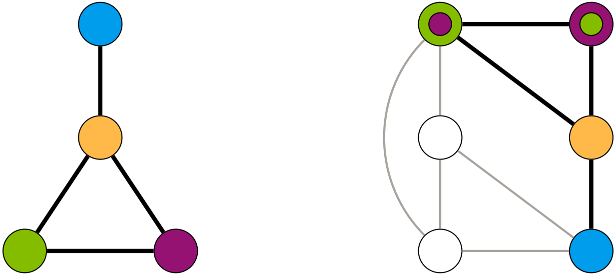
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

Subgraph Isomorphism



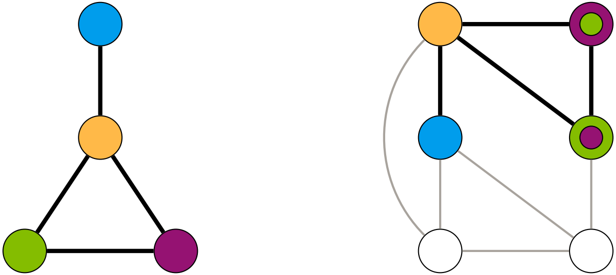
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches

Subgraph Isomorphism



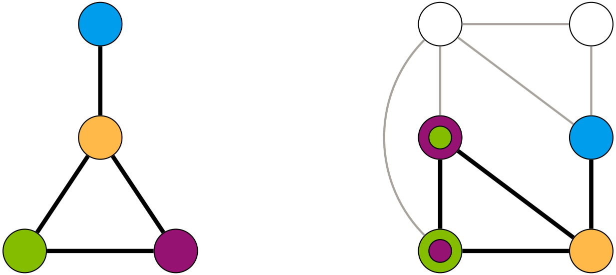
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

Subgraph Isomorphism



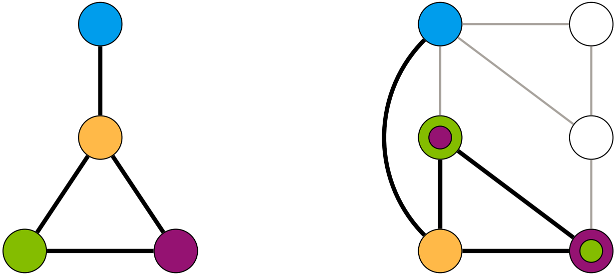
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

Subgraph Isomorphism



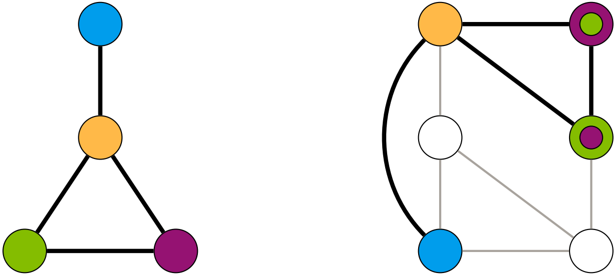
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

Subgraph Isomorphism



- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

Subgraph Isomorphism



- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a target vertex:

$$\sum_{t \in \mathrm{V}(T)} x_{p,t} = 1 \qquad\qquad p \in \mathrm{V}(P)$$

Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a target vertex:

$$\sum_{t \in \mathrm{V}(T)} x_{p,t} = 1 \qquad\qquad p \in \mathrm{V}(P)$$

Each target vertex may be used at most once:

$$\sum_{p \in \mathrm{V}(P)} -x_{p,t} \geq -1 \qquad\qquad t \in \mathrm{V}(T)$$

Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a target vertex:

$$\sum_{t \in \mathrm{V}(T)} x_{p,t} = 1 \qquad\qquad p \in \mathrm{V}(P)$$

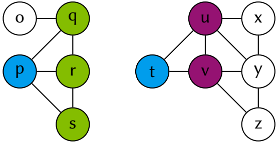
Each target vertex may be used at most once:

$$\sum_{p \in \mathrm{V}(P)} -x_{p,t} \geq -1 \qquad\qquad t \in \mathrm{V}(T)$$

Adjacency constraints, if p is mapped to t, then p's neighbours must be mapped to t's neighbours:

$$\overline{x}_{p,t} + \sum_{u \in \mathrm{N}(t)} x_{q,u} \geq 1 \qquad\qquad p \in \mathrm{V}(P),\ q \in \mathrm{N}(p),\ t \in \mathrm{V}(T)$$

Degree Reasoning in Cutting Planes

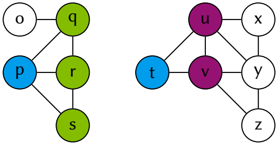


Pattern vertex p of degree $\deg(p)$ can never be mapped to target vertex t of degree $< \deg(p)$ in any subgraph isomorphism

Observe $N(p) = \{q, r, s\}$ and $N(t) = \{u, v\}$

We wish to derive $\overline{x}_{p,t} \geq 1$

Degree Reasoning in Cutting Planes



Adjacency:

$$\overline{x}_{p,t} + x_{q,u} + x_{q,v} \geq 1$$
$$\overline{x}_{p,t} + x_{r,u} + x_{r,v} \geq 1$$
$$\overline{x}_{p,t} + x_{s,u} + x_{s,v} \geq 1$$

Injectivity:

$$-x_{o,u} + -x_{p,u} + -x_{q,u} + -x_{r,u} + -x_{s,u} \geq -1$$
$$-x_{o,v} + -x_{p,v} + -x_{q,v} + -x_{r,v} + -x_{s,v} \geq -1$$

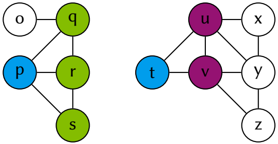
Literal axioms:

$$x_{o,u} \geq 0$$
$$x_{o,v} \geq 0$$
$$x_{p,u} \geq 0$$
$$x_{p,v} \geq 0$$

Add these together …

$$3 \cdot \overline{x}_{p,t} \geq 1$$

Degree Reasoning in Cutting Planes



Adjacency:

$$\overline{x}_{p,t} + x_{q,u} + x_{q,v} \geq 1$$
$$\overline{x}_{p,t} + x_{r,u} + x_{r,v} \geq 1$$
$$\overline{x}_{p,t} + x_{s,u} + x_{s,v} \geq 1$$

Injectivity:

$$-x_{o,u} + -x_{p,u} + -x_{q,u} + -x_{r,u} + -x_{s,u} \geq -1$$
$$-x_{o,v} + -x_{p,v} + -x_{q,v} + -x_{r,v} + -x_{s,v} \geq -1$$

Literal axioms:

$$x_{o,u} \geq 0$$
$$x_{o,v} \geq 0$$
$$x_{p,u} \geq 0$$
$$x_{p,v} \geq 0$$

Add these together and divide by 3 to get

$$\overline{x}_{p,t} \geq 1$$

Degree Reasoning in VeriPB

```
pol 18_{p~t:q} 19_{p~t:r} + 20_{p~t:s} +    * sum adjacency constraints
  12_{inj(u)} + 13_{inj(v)} +                * sum injectivity constraints
  xo_u + xo_v +                              * cancel stray xo_*
  xp_u + xp_v +                              * cancel stray xp_*
  3 d                                        * divide, and we're done
```

Or we can ask VeriPB to do the last bit of simplification automatically:

```
pol 18_{p~t:q} 19_{p~t:r} + 20_{p~t:s} +    * sum adjacency constraints
  12_{inj(u)} + 13_{inj(v)} +                * sum injectivity constraints
ia -1 : 1 ~xp_t >= 1 ;                       * desired conclusion is implied
```

Other Forms of Reasoning

We can also log all of the other things state of the art subgraph solvers do:

- Injectivity reasoning and filtering
- Distance filtering
- Neighbourhood degree sequences
- Path filtering
- Supplemental graphs

Other Forms of Reasoning

We can also log all of the other things state of the art subgraph solvers do:

- Injectivity reasoning and filtering
- Distance filtering
- Neighbourhood degree sequences
- Path filtering
- Supplemental graphs

Proof steps are "efficient" using cutting planes

- Length of proof $\approx$ time complexity of the reasoning algorithms
- Most proof steps require only trivial additional computations

Limitations

Why trust the encoding?

- Correctness of encoding can be formally verified! Work in progress…

Limitations

Why trust the encoding?

- Correctness of encoding can be formally verified! Work in progress…

Proof logging can introduce large slowdowns

- Writing to disk is much slower than bit-parallel algorithms

Limitations

Why trust the encoding?

- Correctness of encoding can be formally verified! Work in progress…

Proof logging can introduce large slowdowns

- Writing to disk is much slower than bit-parallel algorithms

Verification can be even slower

- Unit propagation is much slower than bit-parallel algorithms

Limitations

Why trust the encoding?

- Correctness of encoding can be formally verified! Work in progress...

Proof logging can introduce large slowdowns

- Writing to disk is much slower than bit-parallel algorithms

Verification can be even slower

- Unit propagation is much slower than bit-parallel algorithms

Works up to moderately-sized hard instances

- Even an $O(n^3)$ encoding is painful
- Particularly bad when the pseudo-Boolean encoding talks about "non-edges" but large sparse graphs are "easy"

Code for Proof Logging Subgraph Solver

`https://github.com/ciaranm/glasgow-subgraph-solver`

Released under MIT Licence

What About Constraint Programming?

Non-Boolean variables?

Constraints?

- Encoding constraints in pseudo-Boolean form?
- Justifying inferences?

Reformulations?

Compiling CP Variables (1/2)

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

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

Compiling CP Variables (1/2)

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

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

This doesn't work for large domains...

Compiling CP Variables (1/2)

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

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

This doesn't work for large domains...

We could use a binary encoding:

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

This doesn't propagate much, but that isn't a problem for proof logging

Compiling CP Variables (1/2)

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

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

This doesn't work for large domains…

We could use a binary encoding:

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

This doesn't propagate much, but that isn't a problem for proof logging

Convention in what follows:

- Upper-case A, B, C are CP variables;
- Lower-case a, b, c are corresponding Boolean variables in PB encoding

Compiling CP Variables (2/2)

We can mix binary and an order encoding! Where needed, define:

$$a_{\geq 4} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4$$
$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$
$$a_{=4} \Leftrightarrow a_{\geq 4} \land \overline{a}_{\geq 5}$$

Compiling CP Variables (2/2)

We can mix binary and an order encoding! Where needed, define:

$$a_{\geq 4} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4$$
$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$
$$a_{=4} \Leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}$$

When creating $a_{\geq i}$, also introduce pseudo-Boolean constraints encoding

$$a_{\geq i} \Rightarrow a_{\geq j} \quad \text{and} \quad a_{\geq h} \Rightarrow a_{\geq i}$$

for the closest values $j < i < h$ that already exist

Compiling CP Variables (2/2)

We can mix binary and an order encoding! Where needed, define:

$$a_{\geq 4} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4$$
$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$
$$a_{=4} \Leftrightarrow a_{\geq 4} \land \overline{a}_{\geq 5}$$

When creating $a_{\geq i}$, also introduce pseudo-Boolean constraints encoding

$$a_{\geq i} \Rightarrow a_{\geq j} \quad \text{and} \quad a_{\geq h} \Rightarrow a_{\geq i}$$

for the closest values $j < i < h$ that already exist

We can do this:

- Inside the pseudo-Boolean model, where needed
- Otherwise lazily during proof logging

Compiling Constraints

- Also need to compile every constraint to pseudo-Boolean form
- Doesn't need to be a propagating encoding
- Can use additional variables

Compiling Linear Inequalities

Given inequality

$$2A + 3B + 4C \geq 42$$

where $A, B, C \in \{-3 \ldots 9\}$

Compiling Linear Inequalities

Given inequality

$$2A + 3B + 4C \geq 42$$

where $A, B, C \in \{-3 \ldots 9\}$

Encode in pseudo-Boolean form as

$$\begin{aligned} &-32a_{\text{neg}} + 2a_{\text{b0}} + 4a_{\text{b1}} + 8a_{\text{b2}} + 16a_{\text{b3}} \\ &+ -48b_{\text{neg}} + 3b_{\text{b0}} + 6b_{\text{b1}} + 12b_{\text{b2}} + 24b_{\text{b3}} \\ &+ -64c_{\text{neg}} + 4c_{\text{b0}} + 8c_{\text{b1}} + 16c_{\text{b2}} + 32c_{\text{b3}} \geq 42 \end{aligned}$$

Compiling Table Constraints

Constraints can be specified extensionally as list of feasible tuples, called a table
Variable assignments must match some row in table

Compiling Table Constraints

Constraints can be specified extensionally as list of feasible tuples, called a table
Variable assignments must match some row in table

Given table constraint

$$(A, B, C) \in [(1, 2, 3), (1, 3, 4), (2, 2, 5)]$$

define

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

using tuple selector variables

$$t_1 + t_2 + t_3 = 1$$

Encoding Constraint Definitions

Already know how to do it for any constraint with a sane encoding using some combination of

- CNF
- Integer linear inequalities
- Table constraints
- Auxiliary variables

Simplicity is important, propagation strength isn’t

Justifying Search

Mostly this works as in earlier examples

Restarts are easy

No need to justify guesses or decisions — only justify backtracking

Justifying Inference

Key idea

Anything the constraint programming solver knows must follow from unit propagation of guessed assignments on constraints in proof log

Justifying Inference

Key idea

Anything the constraint programming solver knows must follow from unit propagation of guessed assignments on constraints in proof log

If it follows from unit propagation on the encoding, nothing needed

Some propagators and encodings need RUP steps for inferences

- A lot of propagators are effectively "doing a little bit of lookahead" but in an efficient way

Justifying Inference

Key idea

Anything the constraint programming solver knows must follow from unit propagation of guessed assignments on constraints in proof log

If it follows from unit propagation on the encoding, nothing needed

Some propagators and encodings need RUP steps for inferences

- A lot of propagators are effectively "doing a little bit of lookahead" but in an efficient way

A few need explicit cutting planes justifications written to the proof log

- Linear inequalities just need to multiply and add
- All-different needs a bit more

Justifying All-Different Failures

$$
\begin{array}{rl}
V \in & \{\, 1 \quad \quad \quad 4 \quad 5 \,\} \\
W \in & \{\, 1 \quad 2 \quad 3 \quad \quad \,\} \\
X \in & \{\, \quad 2 \quad 3 \quad \quad \,\} \\
Y \in & \{\, 1 \quad \quad 3 \quad \quad \,\} \\
Z \in & \{\, 1 \quad \quad 3 \quad \quad \,\}
\end{array}
$$

Justifying All-Different Failures

$$V \in \{1 \quad \quad 4 \quad 5\}$$
$$W \in \{1 \quad 2 \quad 3 \quad \quad \}$$
$$X \in \{ \quad 2 \quad 3 \quad \quad \}$$
$$Y \in \{1 \quad \quad 3 \quad \quad \}$$
$$Z \in \{1 \quad \quad 3 \quad \quad \}$$

Justifying All-Different Failures

$V \in \{1 \quad 4 \quad 5\}$

$W \in \{1 \quad 2 \quad 3\}$ $\quad w_{=1} + w_{=2} + w_{=3} \geq 1$ $\quad$ [W takes some value]

$X \in \{2 \quad 3\}$

$Y \in \{1 \quad 3\}$

$Z \in \{1 \quad 3\}$

Justifying All-Different Failures

$$V \in \{1, 4, 5\}$$

$$W \in \{1, 2, 3\} \qquad w_{=1} + w_{=2} + w_{=3} \geq 1 \quad [\text{ } W \text{ takes some value }]$$

$$X \in \{2, 3\} \qquad x_{=2} + x_{=3} \geq 1 \quad [\text{ } X \text{ takes some value }]$$

$$Y \in \{1, 3\} \qquad y_{=1} + y_{=3} \geq 1 \quad [\text{ } Y \text{ takes some value }]$$

$$Z \in \{1, 3\} \qquad z_{=1} + z_{=3} \geq 1 \quad [\text{ } Z \text{ takes some value }]$$

Justifying All-Different Failures

$$V \in \{1 \quad 4 \quad 5\}$$

$W \in \{1\ 2\ 3\}$	$w_{=1} + w_{=2} + w_{=3}$	≥ 1	[W takes some value]
$X \in \{2\ 3\}$	$x_{=2} + x_{=3}$	≥ 1	[X takes some value]
$Y \in \{1\ 3\}$	$y_{=1} + y_{=3}$	≥ 1	[Y takes some value]
$Z \in \{1\ 3\}$	$z_{=1} + z_{=3}$	≥ 1	[Z takes some value]
$\rightarrow$	$-v_{=1} + -w_{=1} + -y_{=1} + -z_{=1}$	≥ -1	[At most one variable = 1]
$\rightarrow$	$-w_{=2} + -x_{=2}$	≥ -1	[At most one variable = 2]
$\rightarrow$	$-w_{=3} + -x_{=3} + -y_{=3} + -z_{=3}$	≥ -1	[At most one variable = 3]

Justifying All-Different Failures

$V \in \{1 \quad \quad 4 \quad 5\}$

$W \in \{1\ 2\ 3\}$	$w_{=1} + w_{=2} + w_{=3}$	≥ 1	[W takes some value]
$X \in \{2\ 3\}$	$x_{=2} + x_{=3}$	≥ 1	[X takes some value]
$Y \in \{1\ 3\}$	$y_{=1} + y_{=3}$	≥ 1	[Y takes some value]
$Z \in \{1\ 3\}$	$z_{=1} + z_{=3}$	≥ 1	[Z takes some value]
$\rightarrow$	$-v_{=1} + -w_{=1} + -y_{=1} + -z_{=1}$	≥ -1	[At most one variable = 1]
$\rightarrow$	$-w_{=2} + -x_{=2}$	≥ -1	[At most one variable = 2]
$\rightarrow$	$-w_{=3} + -x_{=3} + -y_{=3} + -z_{=3}$	≥ -1	[At most one variable = 3]
	$-v_{=1}$	≥ 1	[Sum all constraints so far]

Justifying All-Different Failures

$$
\begin{array}{rlll}
V \in \{1, 4, 5\} & & & \\
W \in \{1, 2, 3\} & w_{=1} + w_{=2} + w_{=3} & \geq 1 & \text{[} W \text{ takes some value]} \\
X \in \{2, 3\} & x_{=2} + x_{=3} & \geq 1 & \text{[} X \text{ takes some value]} \\
Y \in \{1, 3\} & y_{=1} + y_{=3} & \geq 1 & \text{[} Y \text{ takes some value]} \\
Z \in \{1, 3\} & z_{=1} + z_{=3} & \geq 1 & \text{[} Z \text{ takes some value]} \\
\rightarrow & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} & \geq -1 & \text{[At most one variable = 1]} \\
\rightarrow & -w_{=2} + -x_{=2} & \geq -1 & \text{[At most one variable = 2]} \\
\rightarrow & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} & \geq -1 & \text{[At most one variable = 3]} \\
 & -v_{=1} & \geq 1 & \text{[Sum all constraints so far]} \\
 & v_{=1} & \geq 0 & \text{[Variable } v_{=1} \text{ non-negative]}
\end{array}
$$

Justifying All-Different Failures

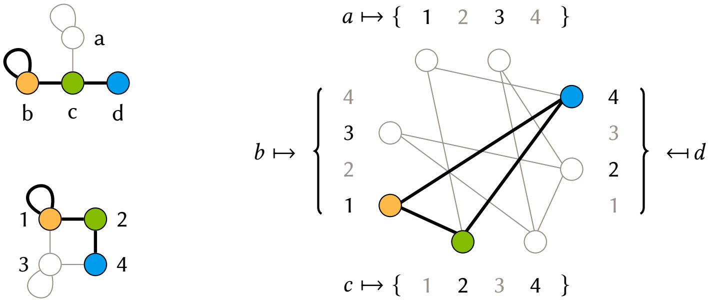
$V \in \{1 \quad 4 \quad 5\}$			
$W \in \{1 \quad 2 \quad 3\}$	$w_{=1} + w_{=2} + w_{=3}$	≥ 1	[W takes some value]
$X \in \{2 \quad 3\}$	$x_{=2} + x_{=3}$	≥ 1	[X takes some value]
$Y \in \{1 \quad 3\}$	$y_{=1} + y_{=3}$	≥ 1	[Y takes some value]
$Z \in \{1 \quad 3\}$	$z_{=1} + z_{=3}$	≥ 1	[Z takes some value]
$\rightarrow$	$-v_{=1} + -w_{=1} + -y_{=1} + -z_{=1}$	≥ -1	[At most one variable = 1]
$\rightarrow$	$-w_{=2} + -x_{=2}$	≥ -1	[At most one variable = 2]
$\rightarrow$	$-w_{=3} + -x_{=3} + -y_{=3} + -z_{=3}$	≥ -1	[At most one variable = 3]
	$-v_{=1}$	≥ 1	[Sum all constraints so far]
	$v_{=1}$	≥ 0	[Variable $v_{=1}$ non-negative]
	0	≥ 1	[Sum above two constraints]

Reformulation

Auto-tabulation is possible

- Heavy use of extension variables

Can re-encode maximum common subgraph as a clique problem, without changing pseudo-Boolean encoding



High Level Modelling Languages?

High level modelling languages like MiniZinc and Essence have complicated compilers

How do we know we're giving a proof for the problem the user actually specified?

This would need a modelling language with formally specified semantics...

Code

https://github.com/ciaranm/glasgow-constraint-solver

Released under MIT Licence

Supports proof logging for global constraints including:

- All-different
- Integer linear inequality (including for very large domains)
- Smart table and regular
- Minimum / maximum of an array
- Element
- Absolute value
- (Hamiltonian) Circuit

Details in [EGMN20, GMN22, MM23]

Strengthening Rules (And Truth About Extension Variables)

When is it allowed to derive a new constraint? If it is (clear that it is) implied?

Strengthening Rules (And Truth About Extension Variables)

When is it allowed to derive a new constraint? If it is (clear that it is) implied?

Sometimes weaker criterion needed — recall that to get variable a encoding

$$a \Leftrightarrow (3x + 2y + z + w \geq 3)$$

we introduced pseudo-Boolean constraints

$$3\overline{a} + 3x + 2y + z + w \geq 3 \qquad 5a + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5$$

Cutting planes method inherently cannot certify such constraints — they are not implied!

Strengthening Rules (And Truth About Extension Variables)

When is it allowed to derive a new constraint? If it is (clear that it is) implied?

Sometimes weaker criterion needed — recall that to get variable a encoding

$$a \Leftrightarrow (3x + 2y + z + w \geq 3)$$

we introduced pseudo-Boolean constraints

$$3\overline{a} + 3x + 2y + z + w \geq 3 \qquad 5a + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5$$

Cutting planes method inherently cannot certify such constraints — they are not implied!

Wish to allow without-loss-of-generality arguments that can derive non-implied constraints

Strengthening Rules (and Symmetry)

VeriPB supports different forms of strengthening rules that enable such w.l.o.g. arguments

Strengthening Rules (and Symmetry)

VeriPB supports different forms of strengthening rules that enable such w.l.o.g. arguments

Care is needed in combination with deletion

Strengthening Rules (and Symmetry)

VeriPB supports different forms of strengthening rules that enable such w.l.o.g. arguments

Care is needed in combination with deletion

Can be very powerful: VeriPB can certify automatic symmetry breaking for SAT

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging

- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker *(work in progress [BMM+23])*

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging

- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker *(work in progress [BMM+23])*

Proof logging for other combinatorial problems and techniques

- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(some work on SCIP in [CGS17, EG21])*
- Satisfiability modulo theories (SMT) solving *(some work by Bjørner and others)*
- High-level modelling languages

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging

- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker *(work in progress [BMM+23])*

Proof logging for other combinatorial problems and techniques

- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(some work on SCIP in [CGS17, EG21])*
- Satisfiability modulo theories (SMT) solving *(some work by Bjørner and others)*
- High-level modelling languages

And more…

- Use proof logs for algorithm analysis or explainability purposes
- Lots of other challenging problems and interesting ideas

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging

- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker *(work in progress [BMM+23])*

Proof logging for other combinatorial problems and techniques

- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(some work on SCIP in [CGS17, EG21])*
- Satisfiability modulo theories (SMT) solving *(some work by Bjørner and others)*
- High-level modelling languages

And more...

- Use proof logs for algorithm analysis or explainability purposes
- Lots of other challenging problems and interesting ideas
- Talk to us if you want to join the proof logging revolution! ☺
  We're happy to collaborate, and we're hiring

Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity

Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **Action point:** What problems can VeriPB solve for you?

  Come talk to us. We're hiring and open to collaboration!

Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **Action point:** What problems can VeriPB solve for you?

  Come talk to us. We're hiring and open to collaboration!

The end.

Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **Action point:** What problems can VeriPB solve for you?

  Come talk to us. We're hiring and open to collaboration!

The end. Or rather, the beginning!

References for Getting Started with VeriPB

https://gitlab.com/MIAOresearch/software/VeriPB

Released under MIT Licence

Various features to help development:

- Extended variable name syntax allowing human-readable names
- Proof tracing
- "Trust me" assertions for incremental proof logging

Documentation:

- Description of VeriPB checker [BMM+23] used in SAT 2023 competition (https://satcompetition.github.io/2023/checkers.html)
- Specific details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23]
- Lots of concrete example files at https://gitlab.com/MIAOresearch/software/VeriPB

References I

[ABGL12] Carlos Ansótegui, María Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving SAT-based weighted MaxSAT solvers. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 86–101. Springer, October 2012.

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

[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.

[ANOR09] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 167–180. Springer, 2009.

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

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

[BB03] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP '03)*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, September 2003.

References II

[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.

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

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

[Bla37] Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

[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.

[BMM+23] Bart Bogaerts, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. Documentation of VeriPB and CakePB for the SAT competition 2023. Available at `https://satcompetition.github.io/2023/checkers.html`, March 2023.

[BMN22] Bart Bogaerts, Ciaran McCreesh, and Jakob Nordström. Solving with provably correct results: Beyond satisfiability, and towards constraint programming. Tutorial at the *28th International Conference on Principles and Practice of Constraint Programming*. Slides available at `http://www.jakobnordstrom.se/presentations/`, August 2022.

[BN21] Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Biere et al. [BHvMW21], chapter 7, pages 233–350.

References III

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

[Bre] BreakID. `https://bitbucket.org/krr/breakid`.

[BS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

[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.

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

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

[Cry] CryptoMiniSat SAT solver. `https://github.com/msoos/cryptominisat/`.

References IV

[DBBD16] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Improved static symmetry breaking for SAT. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *Lecture Notes in Computer Science*, pages 104–122. Springer, July 2016.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[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 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, February 2020.

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, March 2006.

[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.

References V

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

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

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

[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.

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

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

References VI

[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.

[HHW13a] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD ’13)*, pages 181–188, October 2013.

[HHW13b] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, June 2013.

[IMM19] Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019.

[JMM15] Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-Boolean constraints. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP ’15)*, volume 9255 of *Lecture Notes in Computer Science*, pages 200–209. Springer, August-September 2015.

[KM21] Sonja Kraiczy and Ciaran McCreesh. Solving graph homomorphism and subgraph isomorphism problems faster through clique neighbourhood constraints. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI ’21)*, pages 1396–1402, August 2021.

[MJML14] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for MaxSAT. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP ’14)*, volume 8656 of *Lecture Notes in Computer Science*, pages 531–548. Springer, September 2014.

References VII

[MM23] Matthew McIlree and Ciaran McCreesh. Proof logging for smart extensional constraints. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP '23)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:17, August 2023.

[MML14] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. 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 438–445. Springer, July 2014.

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

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[MPP19] Ciaran McCreesh, William Pettersson, and Patrick Prosser. Understanding the empirical hardness of random optimisation problems. 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 333–349. Springer, September 2019.

[MS99] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

[MSH21] Laurent D. Michel, Pierre Schaus, and Pascal Van Hentenryck. MiniCP: a lightweight solver for constraint programming. *Mathematical Programming Computation*, 13(1):133–184, February 2021.

References VIII

[OLH+13] Toru Ogawa, Yangyang Liu, Ryuzo Hasegawa, Miyuki Koshimura, and Hiroshi Fujita. Modulo based CNF encoding of cardinality constraints and its application to maxsat solvers. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013*, pages 9–17. IEEE Computer Society, 2013.

[PR16] Tobias Philipp and Adrián Rebola-Pardo. DRAT proofs for XOR reasoning. In *Proceedings of the 15th European Conference on Logics in Artificial Intelligence (JELIA '16)*, volume 10021 of *Lecture Notes in Computer Science*, pages 415–429. Springer, November 2016.

[PRB18] Tobias Paxian, Sven Reimer, and Bernd Becker. Dynamic polynomial watchdog encoding for solving weighted MaxSAT. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 37–53. Springer, July 2018.

[RM16] Olivier Roussel and Vasco M. Manquinho. Input/output format and solver requirements for the competitions of pseudo-Boolean solvers. Revision 2324. Available at http://www.cril.univ-artois.fr/PB16/format.pdf, January 2016.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

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

[Sin05] Carsten Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP '05)*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, October 2005.

[SN15] Masahiko Sakai and Hidetomo Nabeshima. Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. *IEICE Transactions on Information and Systems*, 98-D(6):1121–1127, June 2015.

References IX

[Tse68] Grigori Tseitin. On the complexity of derivation in propositional calculus. In A. O. Silenko, editor, *Structures in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Consultants Bureau, New York-London, 1968.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.

[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`.

[Van23] Dieter Vandesande. Towards certified MaxSAT solving — certified MaxSAT solving with SAT oracles and encodings of pseudo-Boolean constraints. Master's thesis, Vrije Universiteit Brussel, 2023. To appear.

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

[War98] Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, October 1998.

[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.

Parity Reasoning: Experimental Evaluation

Implemented parity reasoning and PB proof logging engine[2]

Also DRAT proof logging for XOR constraints as described in [PR16]

Experiments with MiniSat[3]

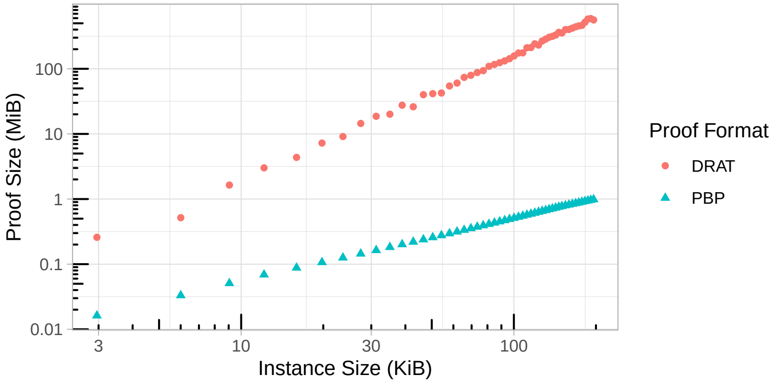
Set-up:[4]

- Intel Core i5-1145G7 @2.60GHz $\times$ 4
- Memory limit 8GiB
- Disk write speed roughly 200 MiB/s
- Read speed of 2 GiB/s

---

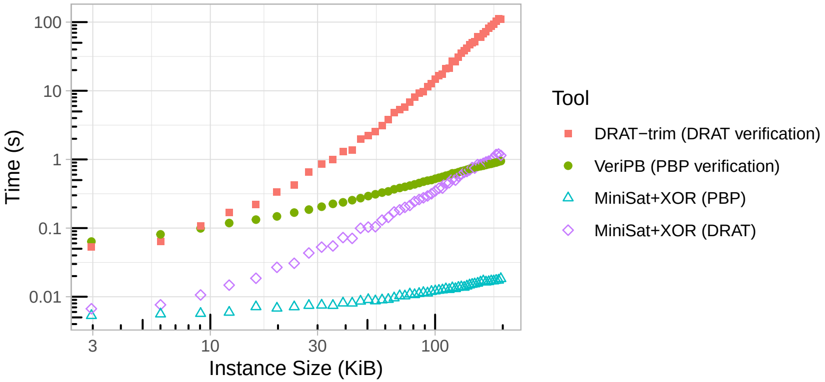
[2] https://gitlab.com/MIAOresearch/tools-and-utlities/xorengine
[3] http://minisat.se/
[4] Tools, benchmarks, data and evaluation scripts available at https://doi.org/10.5281/zenodo.7083485

Parity Reasoning: Proof Size for DRAT and PB Proof Logging



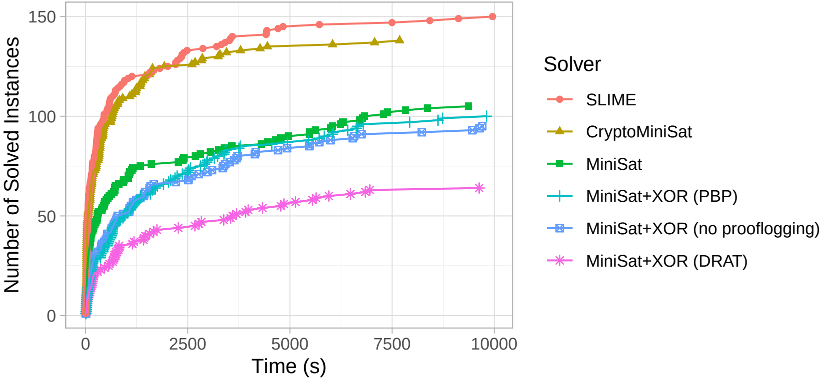
Proof sizes for Tseitin formulas using DRAT and pseudo-Boolean proof logging

Parity Reasoning: Solving and Proof Checking Time



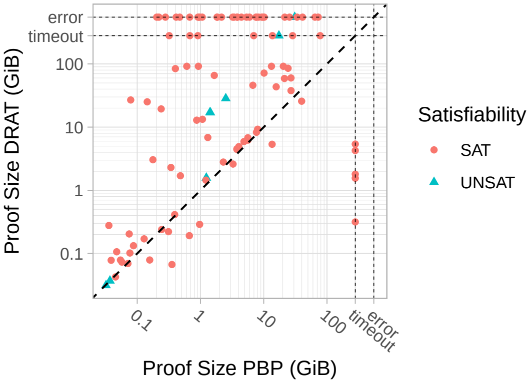
Solving and proof checking time for Tseitin formulas using DRAT and PB proof logging

Parity Reasoning: Crypto Track of SAT 2021 Competition



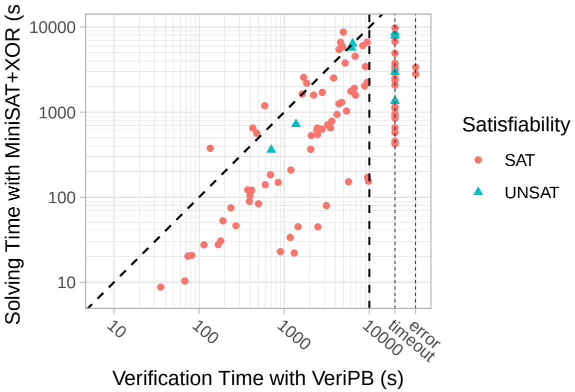
Cumulative plot for the crypto track of the SAT Competition 2021

Parity Reasoning: Crypto Track Proof Size



DRAT and PB proof sizes for crypto track of SAT Competition 2021

Parity Reasoning: Crypto Track Solving & Proof Checking Time



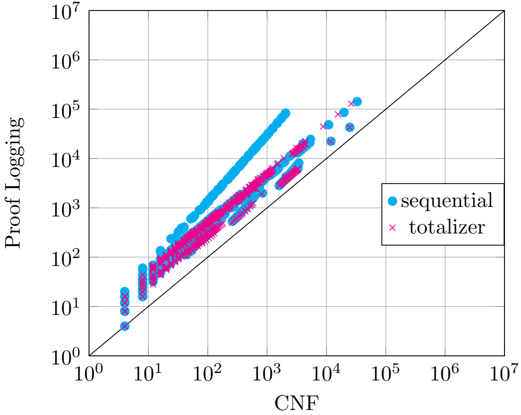
Time required for solving and proof checking for cryptographic instances

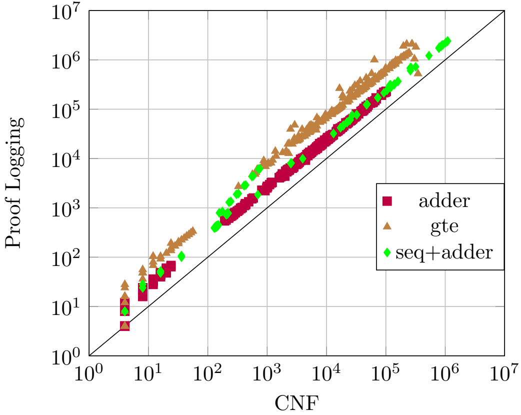
PB-to-CNF Translation: Experimental Evaluation

- Certified translations for CNF encodings with *VeritasPBLib*[5]
  - Sequential counter [Sin05]
  - Totalizer [BB03]
  - Generalized totalizer [JMM15]
  - Adder network [ES06]
- Proofs verified by proof checker VeriPB
- Formulas solved with fork of Kissat[6] syntactically modified to output VeriPB proofs
- Benchmarks from PB 2016 Evaluation[7] in 3 categories
  - Only cardinality constraints (sequential counter, totalizer)
  - Only general 0-1 ILP constraints (generalized totalizer, adder network)
  - Mixed cardinality & general 0-1 ILP constraints (sequential counter + adder network)

[5] https://github.com/forge-lab/VeritasPBLib
[6] https://gitlab.com/MIAOresearch/tools-and-utilities/kissat_fork
[7] http://www.cril.univ-artois.fr/PB16/

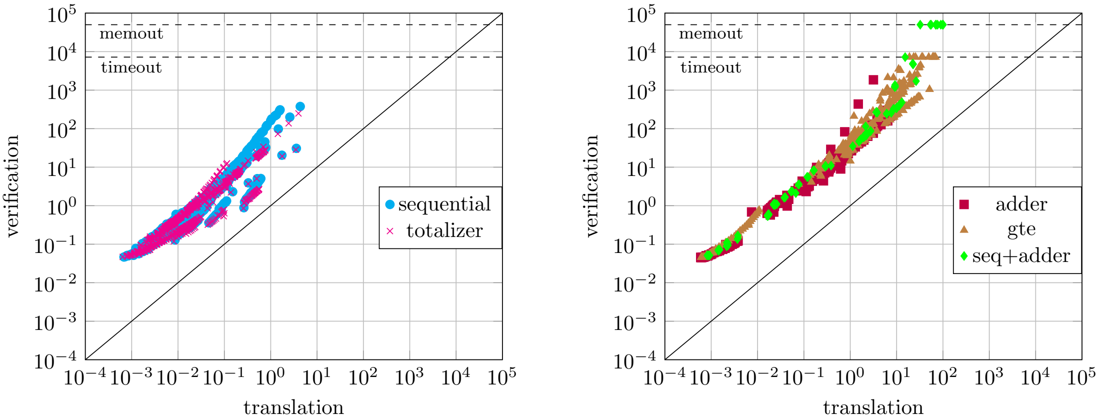
PB-to-CNF: CNF Size vs Proof Size in KiB





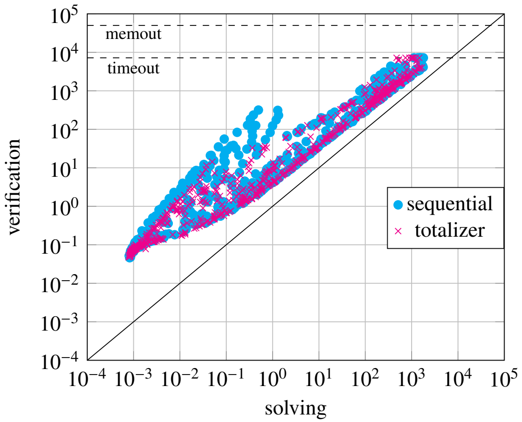
- Nice scaling for proof size in terms of original CNF formula size
- Except for some sequential encoding cases (which is not such a great encoding anyway)

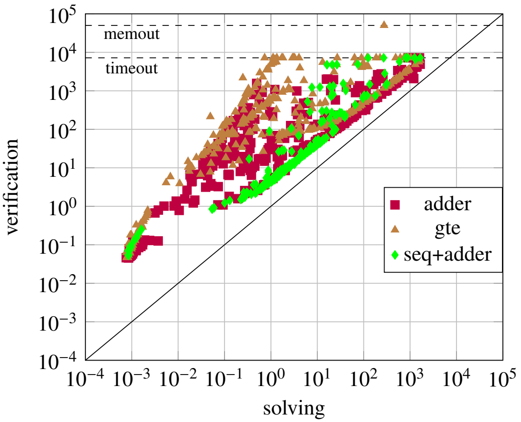
PB-to-CNF: Translation Time vs Proof Checking Time in Seconds



- Translation faster — only has to generate clauses and proof
- Proof checking slower — has to verify full proof

PB-to-CNF: Solving Time vs Proof Checking Time in Seconds





- Room for improvement of end-to-end proof checking process
- But even first proof-of-concept implementation shows our approach is viable

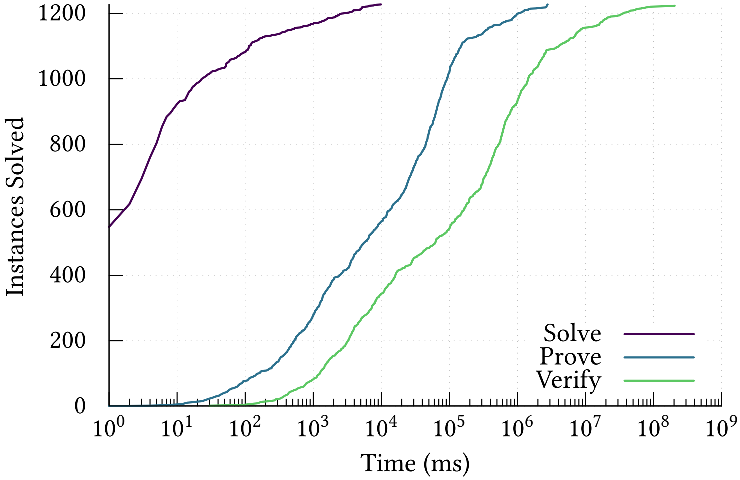
Clique Solving: Experimental Evaluation

- Implemented in the *Glasgow Subgraph Solver*
  - Bit-parallel, can perform a colouring and recursive call in under a microsecond
- 59 of the 80 DIMACS instances take under 1,000 seconds to solve without logging
- Produced and verified proofs for 57 of these 59 instances (the other two reached 1TByte disk space)
- Mean slowdown from proof logging is 80.1 (due to disk I/O)
- Mean verification slowdown a further 10.1
- Approximate implementation effort: one Masters student

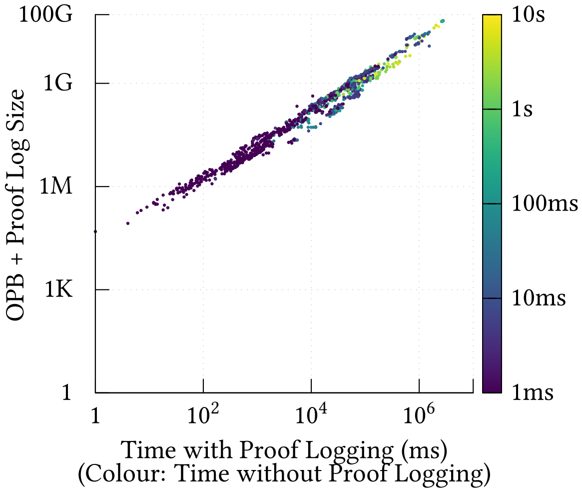
Subgraph Isomorphism Solving: Experimental Evaluation (1/3)

- The Pseudo-Boolean models can be large: had to restrict to instances with no more than 260 vertices in the target graph
- Took enumeration instances which could be solved without proof logging in under ten seconds
- 1,227 instances from Solnon's benchmark collection:
  - 789 unsatisfiable, up to 50,635,140 solutions in the rest
  - 498 instances solved without guessing
  - Hardest solved satisfiable and unsatisfiable instances required 53,605,482 and 2,074,386 recursive calls

Subgraph Isomorphism Solving: Experimental Evaluation (2/3)



Subgraph Isomorphism Solving: Experimental Evaluation (3/3)



Constraint Programming: How Expensive is Proof Logging? (1/2)

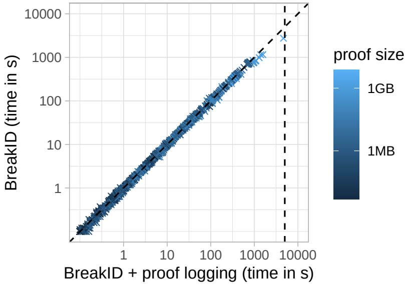
- Laurent D. Michel, Pierre Schaus, Pascal Van Hentenryck: *MiniCP: A Lightweight Solver for Constraint Programming* [MSH21]
- Five benchmark problems allowing comparison of solvers “doing the same thing”:
  - Simple models
  - Fixed search order and well-defined propagation consistency levels
  - Few global constraints
- Probably close to the worst case for proof logging performance
- Also: Crystal Maze and World’s Hardest Sudoku

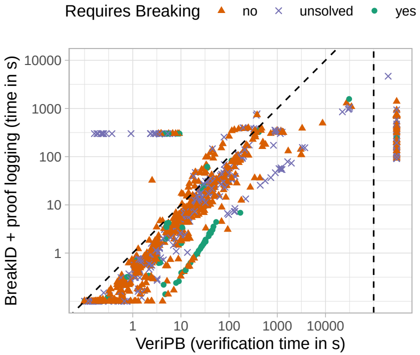
Constraint Programming: How Expensive is Proof Logging? (2/2)

- Our solver: faster than the fastest of *MiniCP*, *OscaR*, and *Choco*
- Proof logging slowdown: between 8.4 and 61.1 factor
  - 800,000 to 3,000,000 inferences per second
  - Proof logs can be hundreds of GBytes
  - No effort put into making the proof-writing code run fast
- Verification slowdown: a further factor 10 to 100
  - Probably possible to reduce this substantially if we are prepared to put more care into writing proofs

SAT Symmetry Breaking: Experimental Evaluation

- Evaluated on SAT competition benchmarks
- *BreakID* [DBBD16, Bre] used to find and break symmetries





- Proof logging overhead negligible
- Proof checking at most 20 times slower than solving for 95% of instances