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]

---

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

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 problems (or worse) very successfully in practice!

---

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

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 problems (or worse) very successfully in practice!
- Except solvers are sometimes wrong… (Even best commercial ones) [BLB10, CKSW13, AGJ$^+$18, GSD19, GS19]

---

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

The Controversial Slide

In the 2021 constraint programming MiniZinc challenge: for 1.28% of instances, wrong solutions were claimed.

- False claims of unsatisfiability.
- False claims of optimality.
- Infeasible solutions produced.

The Controversial Slide

In the 2021 constraint programming MiniZinc challenge: for 1.28% of instances, wrong solutions were claimed.

- False claims of unsatisfiability.
- False claims of optimality.
- Infeasible solutions produced.

This problem is worth taking seriously.

- Not limited to a single solver, problem, or constraint.
- Not even consistent — same solver on same hardware and same instance can give different results on different runs.

The Controversial Slide

In the 2021 constraint programming MiniZinc challenge: for 1.28% of instances, wrong solutions were claimed.

- False claims of unsatisfiability.
- False claims of optimality.
- Infeasible solutions produced.

This problem is worth taking seriously.

- Not limited to a single solver, problem, or constraint.
- Not even consistent — same solver on same hardware and same instance can give different results on different runs.

Obviously, *your* solver doesn't have this problem, but how do you convince others of this?

Testing?

Various domain-specific testing methods [BLB10, AGJ+18, GSD19].

Definitely better than nothing, but is it enough?

Testing?

Various domain-specific testing methods [BLB10, AGJ$^+$18, GSD19].

Definitely better than nothing, but is it enough?

- Clearly not: bugs are found in thoroughly tested solvers as well.
- Testing can only reveal the presence of bugs, not their absence.

Formal Methods?

Prove that solver implementation adheres to formal specification.

Current techniques cannot scale to level of complexity in modern solvers.

- In SAT solver competition, formally verified solvers are far behind in terms of performance (and available techniques).
- In constraint programming, even an inefficient implementation of all-different is pushing the limits [Dub20].

A Simple but Crucial Change of Perspective

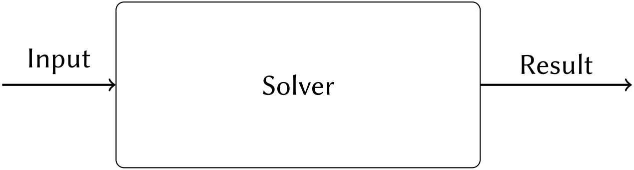
State-of-the-art SAT solvers instead use proof logging.

- Make solvers certifying [ABM⁺11, MMNS11].
- Output proof of correctness in standard format that is independently verified.

A variety of proof logging formats introduced, including

- DRAT [HHW13a, HHW13b, WHH14]
- GRIT [CMS17]
- LRAT [CHH⁺17]
- ...

Proof Logging Workflow



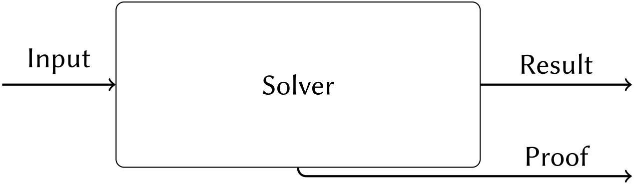
Input

Solver

Result

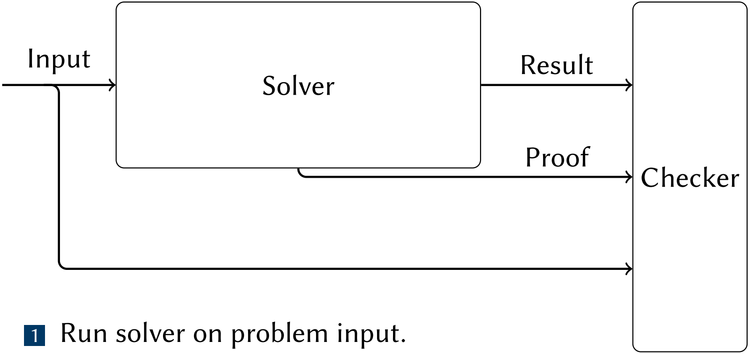
1. Run solver on problem input.

Proof Logging Workflow



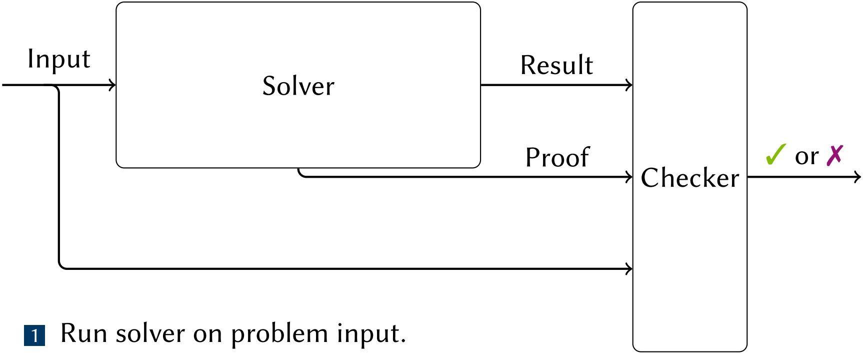
1. Run solver on problem input.
2. Get as output not only result but also proof.

Proof Logging Workflow



1. Run solver on problem input.
2. Get as output not only result but also proof.
3. Feed input + result + proof to proof checker.

Proof Logging Workflow



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

Requirements

Proofs produced by certifying solver should:

- Be powerful enough for proof logging to incur minimal overhead.
- Be based on very simple rules.
- Not require knowledge of inner workings of solver.
- Allow verification by stand-alone proof checker.

Requirements

Proofs produced by certifying solver should:

- Be powerful enough for proof logging to incur minimal overhead.
- Be based on very simple rules.
- Not require knowledge of inner workings of solver.
- Allow verification by stand-alone proof checker.

Much easier to trust a small, simple checker than a full solver.

- Should even be simple enough to be formally verified.

Does not prove solver correct, but proves solution correct.

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].
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

VeriPB (`https://gitlab.com/MIAOresearch/software/VeriPB`)

Versatile proof logging system that can handle

- Subgraph algorithms
- Constraint programming
- Symmetry and dominance reasoning

in a unified way.

The Rest of This Tutorial

VeriPB (`https://gitlab.com/MIAOresearch/software/VeriPB`)

Versatile proof logging system that can handle

- Subgraph algorithms
- Constraint programming
- Symmetry and dominance reasoning

in a unified way.

But first we need to tell you about:

- Proof logging for SAT.
- Pseudo-Boolean reasoning and cutting planes.

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 a 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

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

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

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

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 $u \mapsto 0$.

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 $r \mapsto 1$.

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

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

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

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.

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.

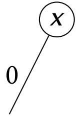
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

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.

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

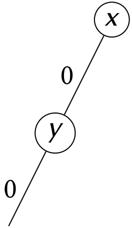


Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

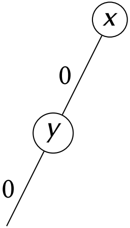


Davis-Putman-Logemann-Loveland (DPLL)

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

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

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



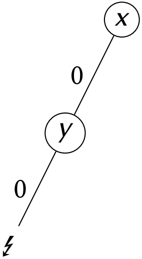
Davis-Putman-Logemann-Loveland (DPLL)

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 $x \lor y$



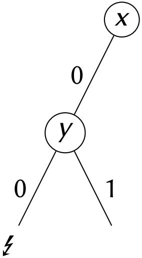
Davis-Putman-Logemann-Loveland (DPLL)

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. $x \vee y$



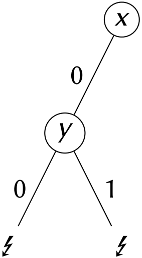
Davis-Putman-Logemann-Loveland (DPLL)

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. $x \vee \overline{y}$



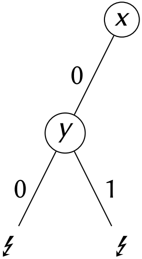
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



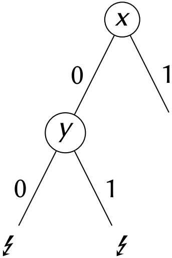
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



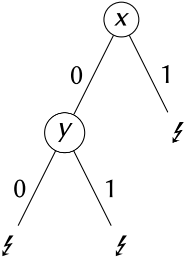
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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

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

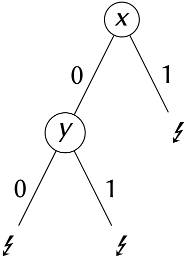
1 $x \vee y$

2 $x \vee \overline{y}$

3 x

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 condition 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 condition 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:

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

Decision

Free choice to assign value to variable

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

What About Conflict-Driven Clause Learning (CDCL)?

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

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

$\boxed{p \overset{\mathsf{d}}{=} 0}$

Decision

Free choice to assign value to variable

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

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

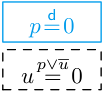
Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

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

What About Conflict-Driven Clause Learning (CDCL)?

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

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



$\boxed{p \overset{\mathsf{d}}{=} 0}$

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

Decision

Free choice to assign value to variable

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

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)

What About Conflict-Driven Clause Learning (CDCL)?

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

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

$\boxed{p \overset{\mathsf{d}}{=} 0}$

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

$\boxed{q \overset{\mathsf{d}}{=} 0}$

Decision

Free choice to assign value to variable

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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

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

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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

$\boxed{p \overset{\mathsf{d}}{=} 0}$

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

$\boxed{q \overset{\mathsf{d}}{=} 0}$

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

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

Decision

Free choice to assign value to variable

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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

$$\boxed{p \overset{\mathsf{d}}{=} 0}$$
$$u \overset{p \lor \overline{u}}{=} 0$$
$$\boxed{q \overset{\mathsf{d}}{=} 0}$$
$$r \overset{q \lor r}{=} 1$$
$$w \overset{\overline{r} \lor w}{=} 1$$
$$\boxed{x \overset{\mathsf{d}}{=} 0}$$

Decision

Free choice to assign value to variable

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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

$\boxed{p \overset{\mathsf{d}}{=} 0}$

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

$\boxed{q \overset{\mathsf{d}}{=} 0}$

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

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

$\boxed{x \overset{\mathsf{d}}{=} 0}$

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

Decision

Free choice to assign value to variable

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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

$\boxed{p \overset{\mathsf{d}}{=} 0}$

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

$\boxed{q \overset{\mathsf{d}}{=} 0}$

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

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

$\boxed{x \overset{\mathsf{d}}{=} 0}$

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

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

Decision

Free choice to assign value to variable

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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

$\boxed{p \overset{\mathsf{d}}{=} 0}$

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

$\boxed{q \overset{\mathsf{d}}{=} 0}$

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

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

$\boxed{x \overset{\mathsf{d}}{=} 0}$

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

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

$\overset{\overline{y}\vee\overline{z}}{\perp}$

Decision

Free choice to assign value to variable

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

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, else decide

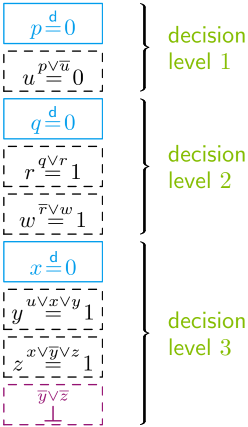
Add to assignment trail

Until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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



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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

decision level 3

Decision

Free choice to assign value to variable

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

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, else decide

Add to assignment trail

Until satisfying assignment or conflict

Conflict Analysis

Time to analyse this conflict and learn from it!

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

$$\left.\begin{array}{c} \boxed{p \overset{\mathsf{d}}{=} 0} \\ u \overset{p \vee \overline{u}}{=} 0 \end{array}\right\} \text{decision level } 1$$

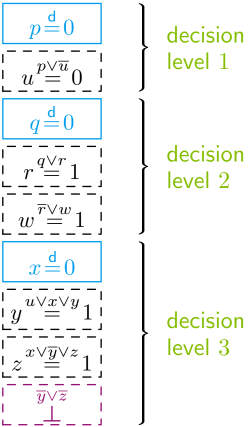
$$\left.\begin{array}{c} \boxed{q \overset{\mathsf{d}}{=} 0} \\ r \overset{q \vee r}{=} 1 \\ w \overset{\overline{r} \vee w}{=} 1 \end{array}\right\} \text{decision level } 2$$

$$\left.\begin{array}{c} \boxed{x \overset{\mathsf{d}}{=} 0} \\ y \overset{u \vee x \vee y}{=} 1 \\ z \overset{x \vee \overline{y} \vee z}{=} 1 \\ \overset{\overline{y} \vee \overline{z}}{\bot} \end{array}\right\} \text{decision level } 3$$

Conflict Analysis

Time to analyse this conflict and learn from it!

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



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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

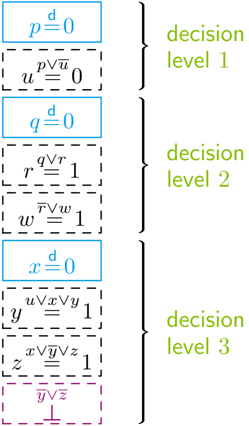
decision level 3

Could backtrack by flipping last decision

Conflict Analysis

Time to analyse this conflict and learn from it!

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



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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

decision level 3

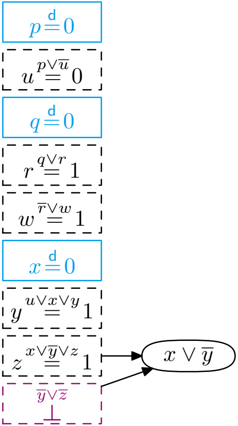
Could backtrack by flipping last decision

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!

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



Could backtrack by flipping last decision

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

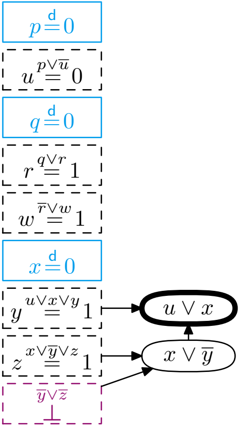
Case analysis over z for last two clauses:

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

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 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:

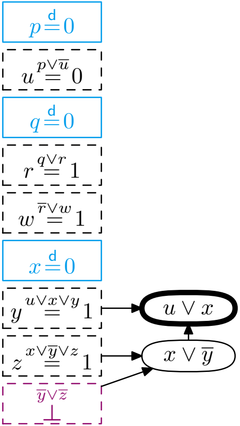
- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \lor \overline{y}$

Repeat til UIP clause with only 1 variable at conflict level — learn and backjump

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

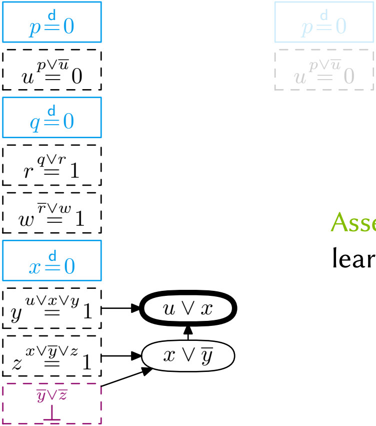
$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



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

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

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

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

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

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

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

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

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

$u \lor x$

$x \lor \overline{y}$

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

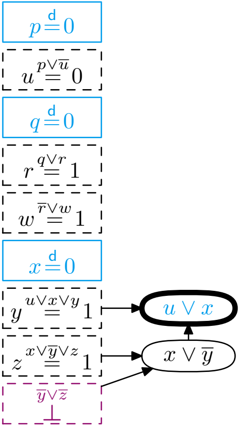
$u \overset{p \lor \overline{u}}{=} 0$

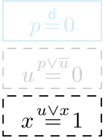
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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





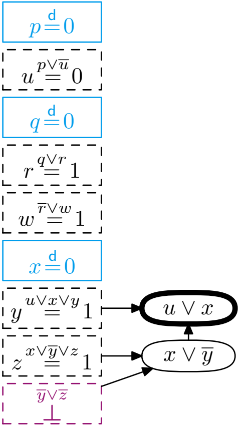
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

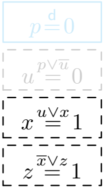
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

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

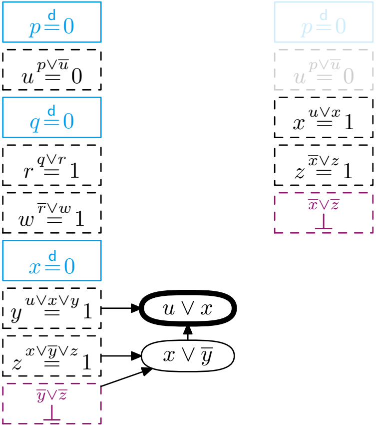
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Then continue as before…

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

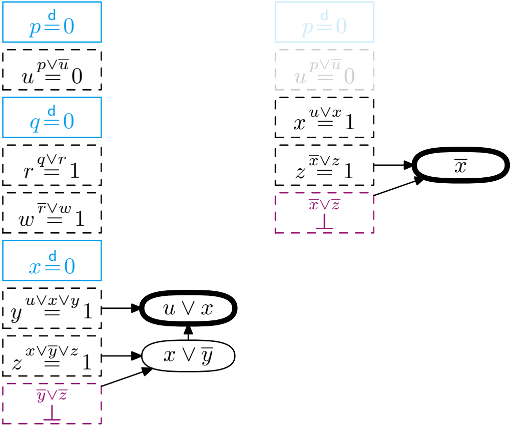
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

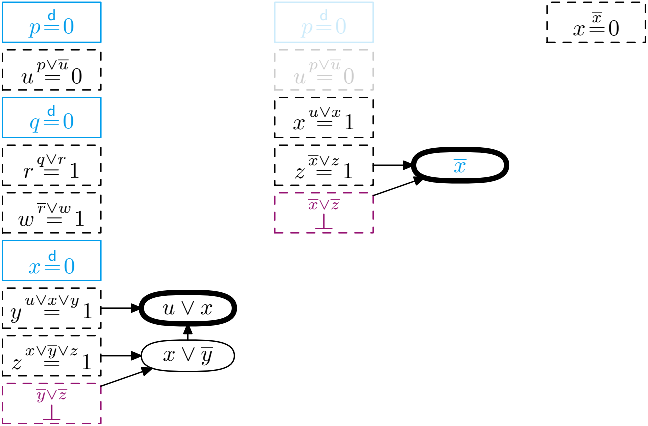
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



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

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

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

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

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

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

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

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

$\overset{\overline{y} \vee \overline{z}}{\perp}$

$u \vee x$

$x \vee \overline{y}$

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

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

$x \overset{u \vee x}{=} 1$

$z \overset{\overline{x} \vee z}{=} 1$

$\overset{\overline{x} \vee \overline{z}}{\perp}$

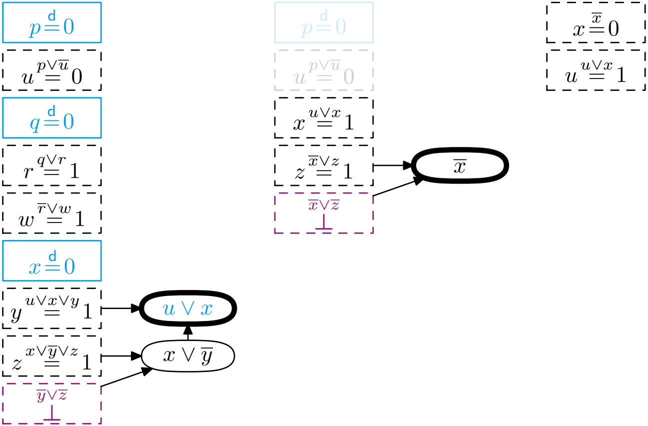
$\overline{x}$

$x \overset{\overline{x}}{=} 0$

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

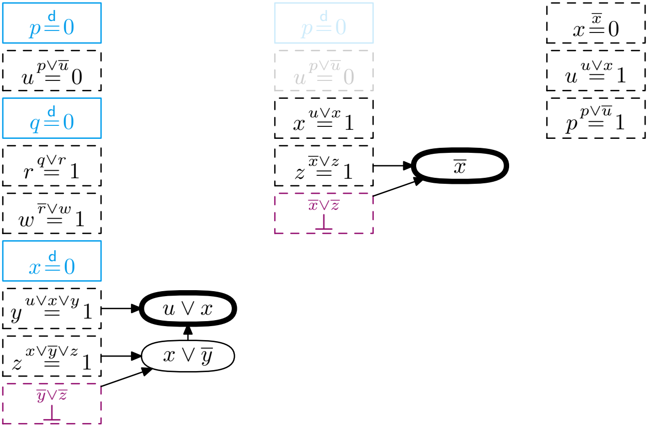
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

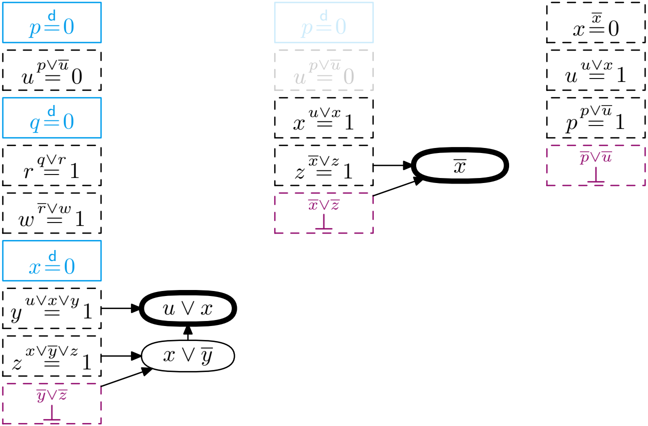
$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



$p \overset{\mathsf{d}}{=} 0$
$u \overset{p \vee \overline{u}}{=} 0$
$q \overset{\mathsf{d}}{=} 0$
$r \overset{q \vee r}{=} 1$
$w \overset{\overline{r} \vee w}{=} 1$
$x \overset{\mathsf{d}}{=} 0$
$y \overset{u \vee x \vee y}{=} 1$
$z \overset{x \vee \overline{y} \vee z}{=} 1$
$\overset{\overline{y} \vee \overline{z}}{\bot}$
$u \vee x$
$x \vee \overline{y}$

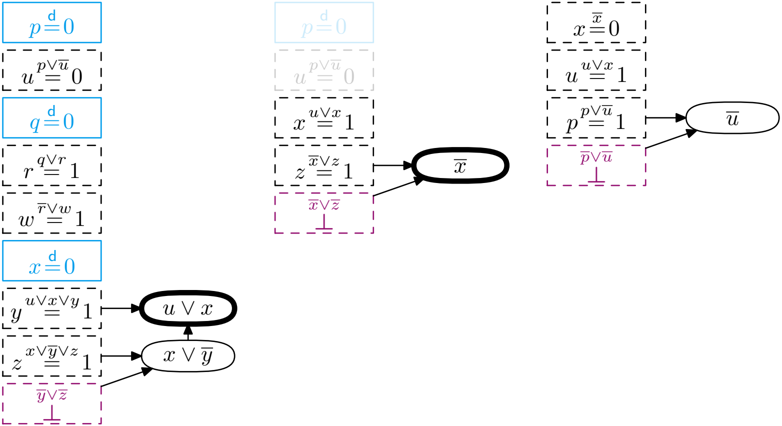
$p \overset{\mathsf{d}}{=} 0$
$u \overset{p \vee \overline{u}}{=} 0$
$x \overset{u \vee x}{=} 1$
$z \overset{\overline{x} \vee z}{=} 1$
$\overset{\overline{x} \vee \overline{z}}{\bot}$
$\overline{x}$

$x \overset{\overline{x}}{=} 0$
$u \overset{u \vee x}{=} 1$
$p \overset{p \vee \overline{u}}{=} 1$
$\overset{\overline{p} \vee \overline{u}}{\bot}$

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



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

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

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

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

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

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

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

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

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

$u \lor x$

$x \lor \overline{y}$

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

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

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

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

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

$\overline{x}$

$x \overset{\overline{x}}{=} 0$

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

$p \overset{p \lor \overline{u}}{=} 1$

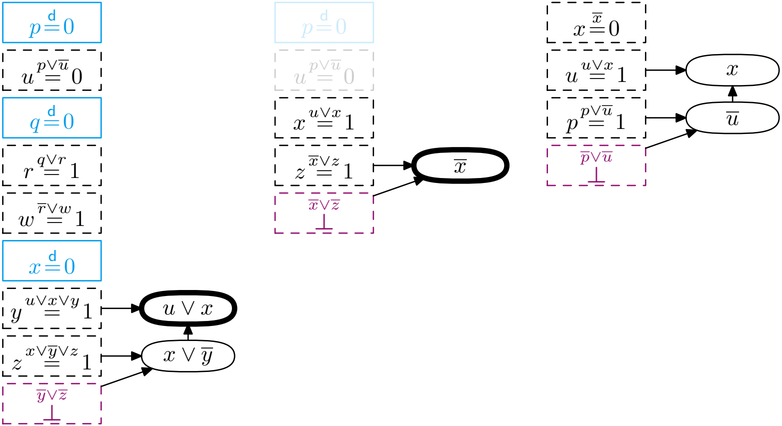
$\overset{\overline{p} \lor \overline{u}}{\bot}$

$\overline{u}$

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



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

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

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

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

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

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

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

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

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

$u \vee x$

$x \vee \overline{y}$

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

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

$x \overset{u\vee x}{=} 1$

$z \overset{\overline{x}\vee z}{=} 1$

$\overset{\overline{x}\vee\overline{z}}{\bot}$

$\overline{x}$

$x \overset{\overline{x}}{=} 0$

$u \overset{u\vee x}{=} 1$

$p \overset{p\vee\overline{u}}{=} 1$

$\overset{\overline{p}\vee\overline{u}}{\bot}$

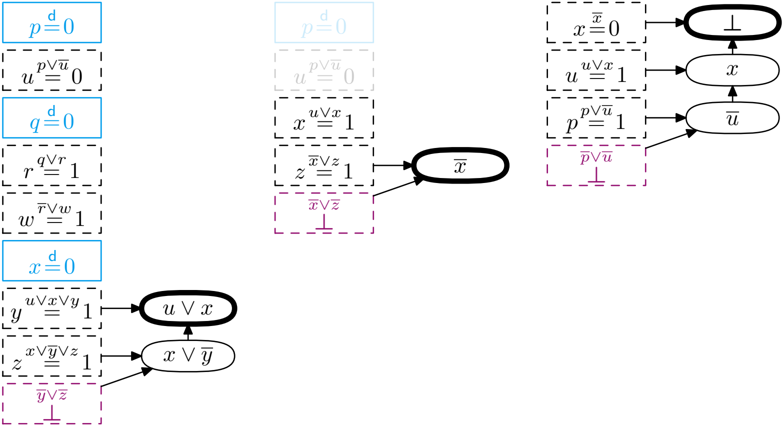
x

$\overline{u}$

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



$p \overset{\mathsf{d}}{=} 0$
$u \overset{p \lor \overline{u}}{=} 0$
$q \overset{\mathsf{d}}{=} 0$
$r \overset{q \lor r}{=} 1$
$w \overset{\overline{r} \lor w}{=} 1$
$x \overset{\mathsf{d}}{=} 0$
$y \overset{u \lor x \lor y}{=} 1$
$z \overset{x \lor \overline{y} \lor z}{=} 1$
$\overset{\overline{y} \lor \overline{z}}{\bot}$
$u \lor x$
$x \lor \overline{y}$

$p \overset{\mathsf{d}}{=} 0$
$u \overset{p \lor \overline{u}}{=} 0$
$x \overset{u \lor x}{=} 1$
$z \overset{\overline{x} \lor z}{=} 1$
$\overset{\overline{x} \lor \overline{z}}{\bot}$
$\overline{x}$

$x \overset{\overline{x}}{=} 0$
$u \overset{u \lor x}{=} 1$
$p \overset{p \lor \overline{u}}{=} 1$
$\overset{\overline{p} \lor \overline{u}}{\bot}$
$\bot$
x
$\overline{u}$

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So 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. $\perp$

Writing Proofs in the DRAT Format

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

Writing Proofs in the DRAT Format

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

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

Writing Proofs in the DRAT Format

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

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \vee y$

$x \vee \overline{y}$

x

$\overline{x}$

$\perp$

Writing Proofs in the DRAT Format

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

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \vee y$

$x \vee \overline{y}$

x

$\overline{x}$

$\perp$

DPLL Proof in DRAT

```
6 7 0
6 -7 0
6 0
-6 0
0
```

Writing Proofs in the DRAT Format

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

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \vee y$

$x \vee \overline{y}$

x

$\overline{x}$

$\perp$

DPLL Proof in DRAT

```
6 7 0
6 -7 0
6 0
-6 0
0
```

CDCL Proof in RUP

$u \vee x$

$\overline{x}$

$\perp$

Writing Proofs in the DRAT Format

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

In DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \lor y$

$x \lor \overline{y}$

x

$\overline{x}$

$\bot$

DPLL Proof in DRAT

```
6 7 0
6 -7 0
6 0
-6 0
0
```

CDCL Proof in RUP

$u \lor x$

$\overline{x}$

$\bot$

CDCL Proof in DRAT

```
4 6 0
-6 0
0
```

More Ingredients in Proof Logging for SAT

Fact

RUP proofs are shorthand for so-called Resolution proofs.

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

But RUP and Resolution aren't enough for preprocessing, inprocessing, and some other kinds of reasoning.

Extension Variables, Part 1

Suppose we want new, fresh variable a encoding

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

Extended Resolution: allow to introduce clauses

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

Should be fine, so long as a doesn't appear anywhere previously.

Extension Variables, Part 1

Suppose we want new, fresh variable a encoding

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

Extended Resolution: allow to introduce clauses

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

Should be fine, so long as a doesn't appear anywhere previously.

Fact

Extended Resolution (RUP + definition of new variables) is essentially equivalent to the DRAT proof logging system.

Deleting Clauses

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

Very easy to deal with from the point of view of proof logging.

So ignored in this tutorial for simplicity and clarity.

Why Aren’t We Done?

Practical limitations of SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently.
- Clausal proofs can’t easily reflect what other algorithms do.

Why Aren't We Done?

Practical limitations of SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently.
- Clausal proofs can't easily reflect what other algorithms do.

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

- Can justify graph reasoning without knowing what a graph is.
- Can justify constraint programming inference without knowing what an integer variable is.
- This even helps justify advanced SAT techniques (cardinality reasoning, Gaussian elimination, symmetry breaking) so far beyond reach for efficient DRAT proof logging.

Pseudo-Boolean Constraints

0-1 integer linear inequalities or pseudo-Boolean constraints:

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

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

Pseudo-Boolean Constraints

0-1 integer linear inequalities or pseudo-Boolean constraints:

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

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

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

Pseudo-Boolean Constraints

0-1 integer linear inequalities or pseudo-Boolean constraints:

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

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

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

Write (partial) assignment ρ as

- set of variable assignments $\rho = \{x \mapsto 1, y \mapsto 0, z \mapsto 1, \ldots\}$, or
- set of true literals $\rho = \{x, \overline{y}, z, \ldots\}$

Some Types of Pseudo-Boolean Constraints

1. Clauses

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

2. Cardinality constraints

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

3. General pseudo-Boolean constraints

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

RUP Revisited

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

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 literals.

Propagation, Conflict, and Slack

Slack measures how far assignment ρ is from falsifying $\sum_i a_i\ell_i \geq A$
Assuming normalized form:

$$slack\left(\textstyle\sum_i a_i\ell_i \geq A; \rho\right) = \sum_{i\,:\,\rho(\ell_i)\neq 0} a_i \; - A$$

Propagation, Conflict, and Slack

Slack measures how far assignment ρ is from falsifying $\sum_i a_i\ell_i \geq A$
Assuming normalized form:

$$slack\left(\textstyle\sum_i a_i\ell_i \geq A; \rho\right) = \sum_{i\,:\,\rho(\ell_i)\neq 0} a_i \; - A$$

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

ρ	$slack(C;\rho)$	comment

Propagation, Conflict, and Slack

Slack measures how far assignment ρ is from falsifying $\sum_i a_i\ell_i \geq A$
Assuming normalized form:

$$slack\left(\textstyle\sum_i a_i\ell_i \geq A; \rho\right) = \sum_{i\,:\,\rho(\ell_i)\neq 0} a_i \; - A$$

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

ρ	$slack(C;\rho)$	comment
$\{\}$	8	

Propagation, Conflict, and Slack

Slack measures how far assignment ρ is from falsifying $\sum_i a_i\ell_i \geq A$
Assuming normalized form:

$$slack\left(\textstyle\sum_i a_i\ell_i \geq A; \rho\right) = \sum_{i\,:\,\rho(\ell_i)\neq 0} a_i \; - A$$

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

ρ	$slack(C;\rho)$	comment
$\{\}$	8	
$\{\overline{x}_5\}$	3	propagates $\overline{x}_4$ (coefficient > slack)

Propagation, Conflict, and Slack

Slack measures how far assignment ρ is from falsifying $\sum_i a_i\ell_i \geq A$
Assuming normalized form:

$$slack\left(\textstyle\sum_i a_i\ell_i \geq A; \rho\right) = \sum_{i\,:\,\rho(\ell_i)\neq 0} a_i \; - A$$

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

ρ	$slack(C;\rho)$	comment
$\{\}$	8	
$\{\overline{x}_5\}$	3	propagates $\overline{x}_4$ (coefficient > slack)
$\{\overline{x}_5, \overline{x}_4\}$	3	propagation doesn't change slack

Propagation, Conflict, and Slack

Slack measures how far assignment ρ is from falsifying $\sum_i a_i \ell_i \geq A$
Assuming normalized form:

$$slack\left(\textstyle\sum_i a_i \ell_i \geq A; \rho\right) = \sum_{i\,:\,\rho(\ell_i) \neq 0} a_i \; - A$$

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

ρ	$slack(C;\rho)$	comment
$\{\}$	8	
$\{\overline{x}_5\}$	3	propagates $\overline{x}_4$ (coefficient > slack)
$\{\overline{x}_5, \overline{x}_4\}$	3	propagation doesn't change slack
$\{\overline{x}_5, \overline{x}_4, \overline{x}_3, x_2\}$	-2	conflict (slack < 0)

Propagation, Conflict, and Slack

Slack measures how far assignment ρ is from falsifying $\sum_i a_i\ell_i \geq A$
Assuming normalized form:

$$slack\left(\textstyle\sum_i a_i\ell_i \geq A; \rho\right) = \sum_{i\,:\,\rho(\ell_i)\neq 0} a_i \; - A$$

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

ρ	$slack(C;\rho)$	comment
$\{\}$	8	
$\{\overline{x}_5\}$	3	propagates $\overline{x}_4$ (coefficient > slack)
$\{\overline{x}_5, \overline{x}_4\}$	3	propagation doesn't change slack
$\{\overline{x}_5, \overline{x}_4, \overline{x}_3, x_2\}$	-2	conflict (slack < 0)

Note: constraint can be conflicting though not all variables assigned

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Model axioms From the input

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Model axioms From the input

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Model axioms From the input

Literal axioms

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

Addition

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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 $c \in \mathbb{N}^+$

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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 $c \in \mathbb{N}^+$
$$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \left\lceil \frac{A}{c} \right\rceil}$$

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

$$\text{Mul by 2}\ \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5$$

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

$$\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Mul by 2}\ \dfrac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \qquad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \qquad \dfrac{\overline{z} \geq 0}{2\overline{z} \geq 0}\ \text{Mul by 2}}{\text{Div by 3}\ \dfrac{3w + 6x + 6y \qquad \geq 7}{w + 2x + 2y \geq 2\frac{1}{3}}}$$

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Such a calculation can be written in a proof line assuming handles

$$\begin{array}{rcl} C_1 & \doteq & 2x + y + w \geq 2 \\ C_2 & \doteq & 2x + 4y + 2z + w \geq 5 \\ Ax(\overline{z}) & \doteq & \overline{z} \geq 0 \end{array}$$

Cutting Planes Toy Example

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

Such a calculation can be written in a proof line assuming handles

$$\begin{aligned} C_1 &\doteq 2x + y + w \geq 2 \\ C_2 &\doteq 2x + 4y + 2z + w \geq 5 \\ Ax(\overline{z}) &\doteq \overline{z} \geq 0 \end{aligned}$$

using postfix notation something like

C_1 `2 Mul` C_2 `Add` $Ax(\overline{z})$ `2 Mul Add 3 Div`

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 $0 \geq 1$.

(*) Not actually implemented this way — details later…

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 are “unsatisfiable, 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 are “unsatisfiable, 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 in the pseudo-Boolean model.
- To maximise, negate objective.
- Log a solution α, get a solution-improving constraint $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \alpha(\ell_i)$.

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.

Full details: Stephan Gocht’s PhD thesis [Goc22].

Progress So Far

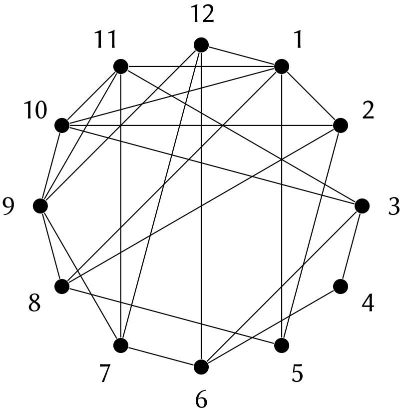
We've seen proof logging, and how it works for SAT.

We've learned about

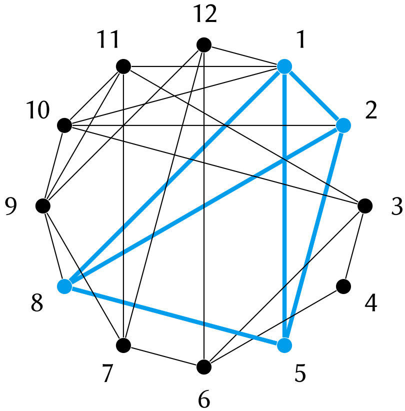
- pseudo-Boolean constraints (0-1 linear inequalities),
- cutting planes reasoning, and
- VeriPB.

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].

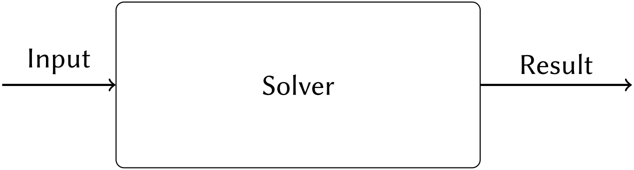
Often used inside other solvers.

- An off-by-one result can cause much larger errors.

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.

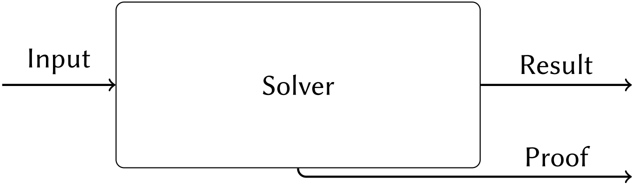
A Slightly Different Workflow


Input

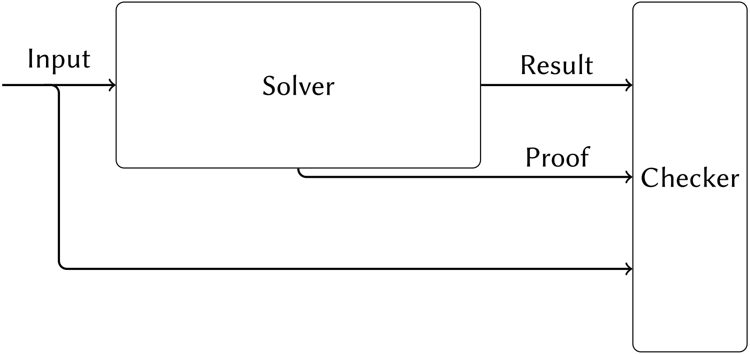
Solver

Result

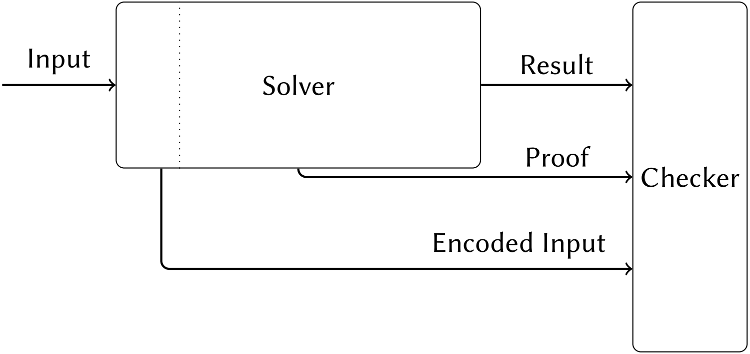
A Slightly Different Workflow



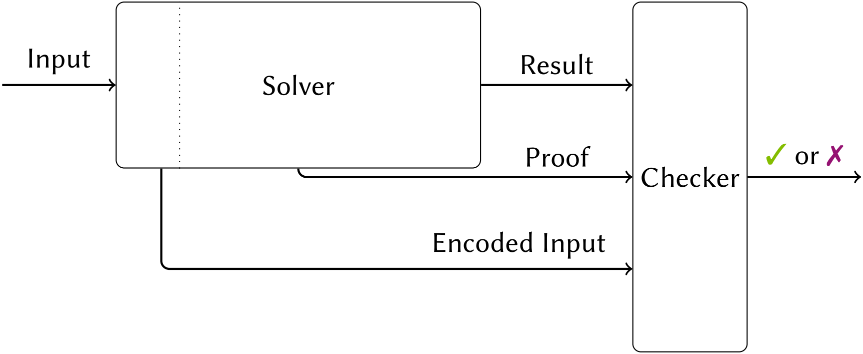
A Slightly Different Workflow



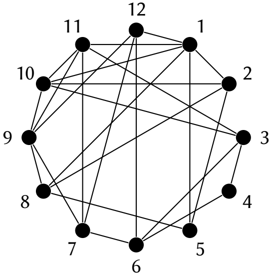
A Slightly Different Workflow



A Slightly Different Workflow



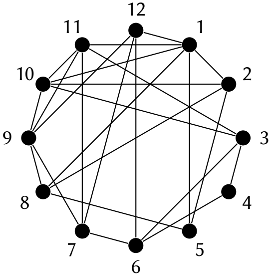
A Pseudo-Boolean Encoding for Clique (in OPB Format)



```
* #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
```

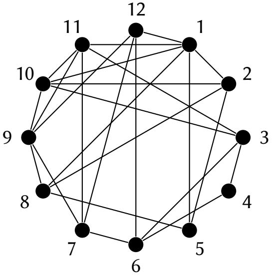
First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



First Attempt at a Proof

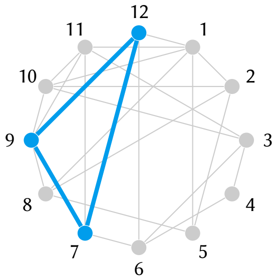
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Start with a header.
Load the 41 problem axioms.

First Attempt at a Proof

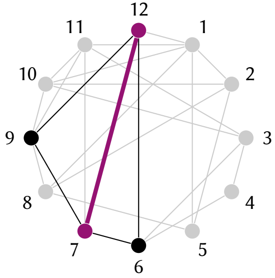
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Branch on 12, 7, 9.
Find a new incumbent.
Now looking for a ≥ 4 vertex clique.

First Attempt at a Proof

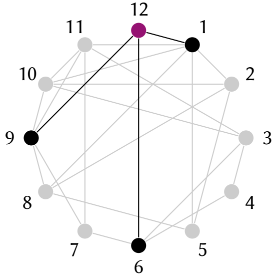
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 12, 7.
Only 6 and 9 feasible.
No ≥ 4 vertex clique possible.
Effectively this deletes the 7–12 edge.

First Attempt at a Proof

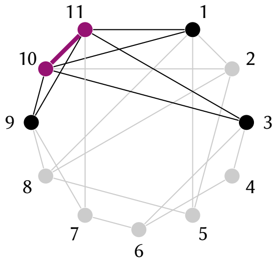
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 12.
Only 1, 6 and 9 feasible.
No ≥ 4 vertex clique possible.
Effectively this deletes vertex 12.

First Attempt at a Proof

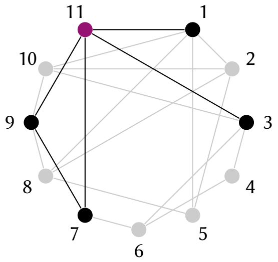
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Branch on 11 then 10.
Only 1, 3 and 9 feasible.
No ≥ 4 vertex clique possible.
Backtrack, deleting the edge.

First Attempt at a Proof

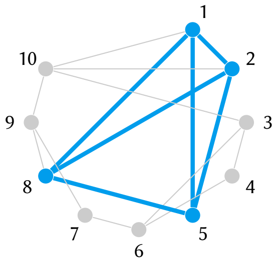
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 11.
Clearly no ≥ 4 clique.
Delete the vertex.

First Attempt at a Proof

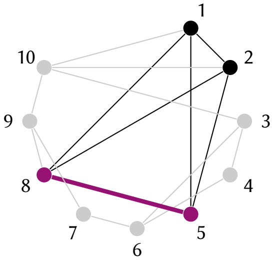
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Branch on 8, 5, 1, 2.
Find a new incumbent.
Now looking for a ≥ 5 vertex clique.

First Attempt at a Proof

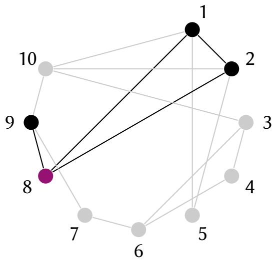
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 8, 5.
Only 4 vertices, can't have a ≥ 5 clique.
Delete the edge.

First Attempt at a Proof

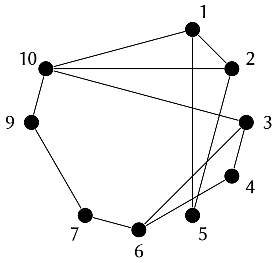
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Backtrack from 8.
Still not enough vertices.
Delete the vertex.

First Attempt at a Proof

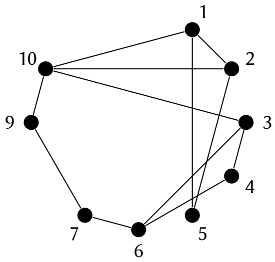
```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



Now obvious to solver that claim of ≥ 5 clique is contradictory (we'll see why).

First Attempt at a Proof

```
pseudo-Boolean proof version 1.2
f 41
o x7 x9 x12
u 1 ~x12 1 ~x7 >= 1 ;
u 1 ~x12 >= 1 ;
u 1 ~x11 1 ~x10 >= 1 ;
u 1 ~x11 >= 1 ;
o x1 x2 x5 x8
u 1 ~x8 1 ~x5 >= 1 ;
u 1 ~x8 >= 1 ;
u >= 1 ;
c -1
```



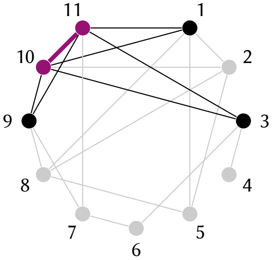
Assert previous line has derived contradiction, ending proof.

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…)

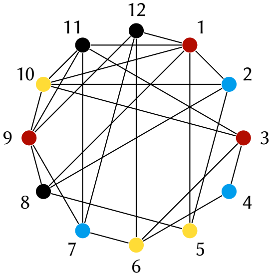
```
$ veripb clique.opb clique-attempt-one.veripb
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
```



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: o 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 x1
line 004: u 1 ~x12 1 ~x7 >= 1 ;
  ConstraintId 043: 1 ~x7 1 ~x12 >= 1
line 005: u 1 ~x12 >= 1 ;
  ConstraintId 044: 1 ~x12 >= 1
line 006: u 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.
```

Bound Functions



Given a *k*-colouring of a subgraph, that subgraph cannot have a clique of more than *k* vertices.

- Each colour class describes an at-most-one constraint.

This does *not* follow by reverse unit propagation.

Recovering At-Most-One Constraints

Practically infeasible to list every colour class we *might* use in the pseudo-Boolean input.

But we can use cutting planes to recover colour classes lazily!

Recovering At-Most-One Constraints

Practically infeasible to list every colour class we *might* use in the pseudo-Boolean input.

But we can use cutting planes to recover colour classes lazily!

$$\begin{aligned}
&(\overline{x}_1 + \overline{x}_6 \geq 1) && \\
+\,&(\overline{x}_1 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
+\,&(\overline{x}_6 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3 \\
&/\,2 && = \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
& && \text{i.e. } x_1 + x_6 + x_9 \leq 1
\end{aligned}$$

Recovering At-Most-One Constraints

Practically infeasible to list every colour class we *might* use in the pseudo-Boolean input.

But we can use cutting planes to recover colour classes lazily!

$$\begin{aligned}
&(\overline{x}_1 + \overline{x}_6 \geq 1) && \\
+\,&(\overline{x}_1 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
+\,&(\overline{x}_6 + \overline{x}_9 \geq 1) && = 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3 \\
&/\,2 && = \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2 \\
& && \text{i.e. } x_1 + x_6 + x_9 \leq 1
\end{aligned}$$

This generalises for arbitrarily large colour classes.

- 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 out.

What This Looks Like

```
pseudo-Boolean proof version 1.2
f 41
o x12 x7 x9
u 1 ~x12 1 ~x7 >= 1 ;
* bound, colour classes [ x1 x6 x9 ]
p $7_{1\nsim 6}$ $19_{1\nsim 9}$ + $24_{6\nsim 9}$ + 2 d
p $42_{\mathrm{obj}}$ -1 +
u 1 ~x12 >= 1 ;
* bound, colour classes [ x1 x3 x9 ]
p $1_{1\nsim 3}$ $19_{1\nsim 9}$ + $21_{3\nsim 9}$ + 2 d
p $42_{\mathrm{obj}}$ -1 +
u 1 ~x11 1 ~x10 >= 1 ;
* bound, colour classes [ x1 x3 x7 ] [ x9 ]
p $1_{1\nsim 3}$ $10_{1\nsim 7}$ + $12_{3\nsim 7}$ + 2 d
p $42_{\mathrm{obj}}$ -1 +
u 1 ~x11 >= 1 ;
o x8 x5 x2 x1
u 1 ~x8 1 ~x5 >= 1 ;
* bound, colour classes [ x1 x9 ] [ x2 ]
p $53_{\mathrm{obj}}$ $19_{1\nsim 9}$ +
u 1 ~x8 >= 1 ;
* bound, colour classes [ x1 x3 x7 ] [ x2 x4 x9 ] [ x5 x6 x10 ]
p $1_{1\nsim 3}$ $10_{1\nsim 7}$ + $12_{3\nsim 7}$ + 2 d
p $53_{\mathrm{obj}}$ -1 +
p $4_{2\nsim 4}$ $20_{2\nsim 9}$ + $22_{4\nsim 9}$ + 2 d
p $53_{\mathrm{obj}}$ -3 + -1 +
p $9_{5\nsim 6}$ $26_{5\nsim 10}$ + $27_{6\nsim 10}$ + 2 d
p $53_{\mathrm{obj}}$ -5 + -3 + -1 +
u >= 1 ;
c -1
```

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: o 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: u 1 ~x12 1 ~x7 >= 1 ;
  ConstraintId 043: 1 ~x7 1 ~x12 >= 1
line 005: * bound, colour classes [ x1 x6 x9 ]
line 006: p 7 19 + 24 + 2 d
  ConstraintId 044: 1 ~x1 1 ~x6 1 ~x9 >= 2
line 007: p 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: p 53 57 + 59 + 61 +
  ConstraintId 062: 1 x8 1 x11 1 x12 >= 2
line 029: u >= 1 ;
  ConstraintId 063: >= 1
line 030: c -1
=== end trace ===

Verification succeeded.
```

Different Clique Algorithms

Different search orders?

- ✓ Irrelevant for proof logging.

Using local search to initialise?

- ✓ Just log the incumbent.

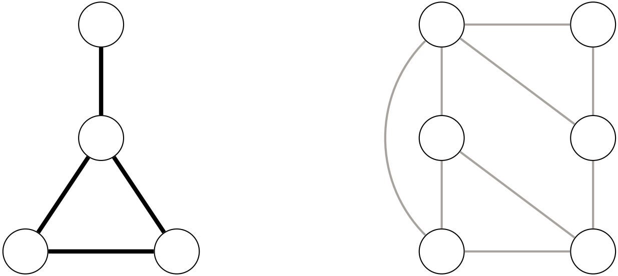
Different bound functions?

- ■ Is cutting planes strong enough to justify every useful bound function ever invented?
- ■ So far, seems like it…

Weighted cliques?

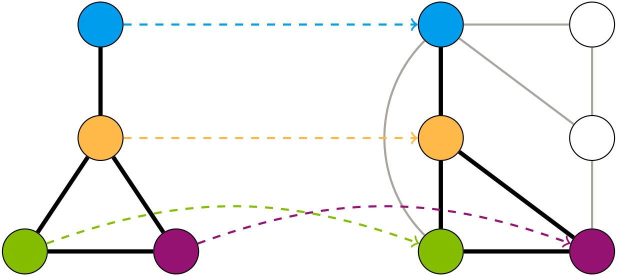
- ✓ Multiply a colour class by its largest weight.
- ✓ Also works for vertices "split between colour classes".

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 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:

$$\sum_{t \in \mathrm{V}(T)} x_{p,t} = 1 \qquad\qquad p \in \mathrm{V}(P)$$

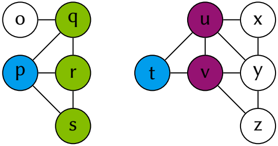
Each target vertex may be used at most once:

$$\sum_{p \in \mathrm{V}(P)} -x_{p,t} \geq -1 \qquad\qquad t \in \mathrm{V}(T)$$

Adjacency constraints, if p is mapped to t, then p's neighbours must be mapped to t's neighbours:

$$\overline{x}_{p,t} + \sum_{u \in \mathrm{N}(t)} x_{q,u} \geq 1 \qquad p \in \mathrm{V}(P),\ q \in \mathrm{N}(p),\ t \in \mathrm{V}(T)$$

Degree Reasoning in Cutting Planes

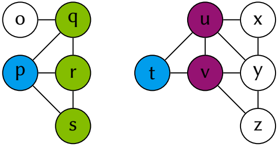


A pattern vertex p of degree $\deg(p)$ can never be mapped to a target vertex t of degree $\deg(p) - 1$ or lower in any subgraph isomorphism.

Observe $\mathrm{N}(p) = \{q, r, s\}$ and $\mathrm{N}(t) = \{u, v\}$.

We wish to derive $\overline{x}_{p,t} \geq 1$.

Degree Reasoning in Cutting Planes



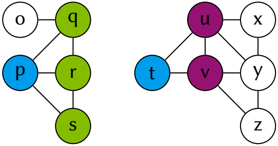
We have the three adjacency constraints,

$$\overline{x}_{p,t} + x_{q,u} + x_{q,v} \geq 1$$
$$\overline{x}_{p,t} + x_{r,u} + x_{r,v} \geq 1$$
$$\overline{x}_{p,t} + x_{s,u} + x_{s,v} \geq 1$$

Their sum is

$$3\overline{x}_{p,t} + x_{q,u} + x_{q,v} + x_{r,u} + x_{r,v} + x_{s,u} + x_{s,v} \geq 3$$

Degree Reasoning in Cutting Planes



Continuing with the sum

$$3\overline{x}_{p,t} + x_{q,u} + x_{q,v} + x_{r,u} + x_{r,v} + x_{s,u} + x_{s,v} \geq 3$$

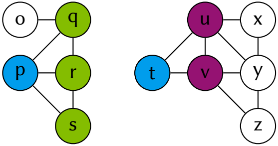
Due to injectivity,

$$-x_{o,u} + -x_{p,u} + -x_{q,u} + -x_{r,u} + -x_{s,u} \geq -1$$
$$-x_{o,v} + -x_{p,v} + -x_{q,v} + -x_{r,v} + -x_{s,v} \geq -1$$

Add all these together, getting

$$3\overline{x}_{p,t} + -x_{o,u} + -x_{o,v} + -x_{p,u} + -x_{p,v} \geq 1$$

Degree Reasoning in Cutting Planes



We're more or less there. We have:

$$3\overline{x}_{p,t} + -x_{o,u} + -x_{o,v} + -x_{p,u} + -x_{p,v} \geq 1$$

Add the literal axioms $x_{o,u} \geq 0$, $x_{o,v} \geq 0$, $x_{p,u} \geq 0$ and $x_{p,v} \geq 0$ to get

$$3\overline{x}_{p,t} \geq 1$$

Divide by 3 to get the desired

$$\overline{x}_{p,t} \geq 1$$

Degree Reasoning in VeriPB

p $18_{p\sim t:q}$ $19_{p\sim t:r}$ + $20_{p\sim t:s}$ + * 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:

p $18_{p\sim t:q}$ $19_{p\sim t:r}$ + $20_{p\sim t:s}$ + * sum adjacency constraints
$12_{inj(u)}$ + $13_{inj(v)}$ + * sum injectivity constraints
`j -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.

- The length of the proof steps are no worse than the time complexity of the reasoning algorithms.
- Most proof steps require only trivial additional computations.

Limitations

Why trust the encoding?

- Here we can formally verify the correctness of the encoding!
  Work in progress…

Limitations

Why trust the encoding?

- Here we can formally verify the correctness of the encoding!
  Work in progress…

Proof logging can introduce large slowdowns

- Writing to disk is much slower than bit-parallel algorithms.

Limitations

Why trust the encoding?

- Here we can formally verify the correctness of the encoding!
  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?

- Here we can formally verify the correctness of the encoding! 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

`https://github.com/ciaranm/glasgow-subgraph-solver`

Released under MIT Licence.

What About Constraint Programming?

Non-Boolean variables?

Constraints?

- Encoding constraints as Pseudo-Boolean constraints?
- Justifying inference?

Reformulation?

Compiling CP Variables

Given $A \in \{-3 \dots 9\}$, 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

Given $A \in \{-3 \ldots 9\}$, the direct encoding is:

$$\begin{aligned} a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3} & \\ + a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} &= 1 \end{aligned}$$

This doesn't work for large domains.

Compiling CP Variables

Given $A \in \{-3 \ldots 9\}$, the direct encoding is:

$$\begin{aligned} a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3} \\ + a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} = 1 \end{aligned}$$

This doesn't work for large domains.

We could use a binary encoding:

$$\begin{aligned} -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} &\geq -3 \text{ and} \\ 16a_{\text{neg}} + -1a_{\text{b0}} + -2a_{\text{b1}} + -4a_{\text{b2}} + -8a_{\text{b3}} &\geq -9 \end{aligned}$$

This doesn't propagate much, but that isn't a problem for proof logging.

Compiling CP Variables

We can mix binary and an order encoding. Where needed, define:

$$
\begin{aligned}
a_{\geq 4} &\Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 4 \\
a_{\geq 5} &\Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5 \\
a_{=4} &\Leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}
\end{aligned}
$$

When creating $a_{=i}$, 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 Constraints

Given $2A + 3B + 4C \geq 42$, where $A, B, C \in \{-3 \ldots 9\}$,

$$\begin{aligned} & -32a_{\mathrm{neg}} + 2a_{\mathrm{b0}} + 4a_{\mathrm{b1}} + 8a_{\mathrm{b2}} + 16a_{\mathrm{b3}} \\ + & -48b_{\mathrm{neg}} + 3b_{\mathrm{b0}} + 6b_{\mathrm{b1}} + 12b_{\mathrm{b2}} + 24b_{\mathrm{b3}} \\ + & -64c_{\mathrm{neg}} + 4c_{\mathrm{b0}} + 8c_{\mathrm{b1}} + 16c_{\mathrm{b2}} + 32c_{\mathrm{b3}} \geq 42 \end{aligned}$$

Compiling Constraints

Constraints can be specified *extensionally* as list of feasible tuples, called a *table*. We have to pick one of the tuples from the table, and give it to the associated variables.

Given a table constraint $(A, B, C) \in [(1, 2, 3), (1, 3, 4), (2, 2, 5)]$, define

$$\begin{aligned} 3\bar{t}_0 + a_{=1} + b_{=2} + c_{=3} &\geq 3 \quad \text{i.e.} \quad && t_0 \Rightarrow (a_{=1} \wedge b_{=2} \wedge c_{=3}) \\ 3\bar{t}_1 + a_{=1} + b_{=4} + c_{=4} &\geq 3 \quad \text{i.e.} \quad && t_1 \Rightarrow (a_{=1} \wedge b_{=4} \wedge c_{=4}) \\ 3\bar{t}_2 + a_{=2} + b_{=2} + c_{=5} &\geq 3 \quad \text{i.e.} \quad && t_2 \Rightarrow (a_{=2} \wedge b_{=2} \wedge c_{=5}) \end{aligned}$$

using a tuple selector variable

$$t_0 + t_1 + t_2 = 1$$

Encoding Constraint Definitions

We already know how to do it for any constraint that has 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. We only justify backtracking.

Justifying Inference

If it follows from unit propagation, 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.

- 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

$$\begin{array}{rcccccccc}
V & \in & \{ & 1 & & & 4 & 5 & \} \\
W & \in & \{ & 1 & 2 & 3 & & & \} \\
X & \in & \{ & & 2 & 3 & & & \} \\
Y & \in & \{ & 1 & & 3 & & & \} \\
Z & \in & \{ & 1 & & 3 & & & \}
\end{array}$$

Justifying All-Different Failures

$$\begin{array}{rl} V \in & \{\, 1 \quad\quad\quad 4 \quad 5 \,\} \\ W \in & \{\, 1 \quad 2 \quad 3 \quad\quad\quad \} \\ X \in & \{\, \quad\; 2 \quad 3 \quad\quad\quad \} \\ Y \in & \{\, 1 \quad\quad 3 \quad\quad\quad \} \\ Z \in & \{\, 1 \quad\quad 3 \quad\quad\quad \} \end{array}$$

$$w_{=1} + w_{=2} + w_{=3} \geq 1$$

Justifying All-Different Failures

$$\begin{array}{ll}
V \in \{1 \quad\quad 4 \quad 5\} & \\
W \in \{1 \quad 2 \quad 3 \quad\quad\} & w_{=1} + w_{=2} + w_{=3} \geq 1 \\
X \in \{\quad 2 \quad 3 \quad\quad\} & x_{=2} + x_{=3} \geq 1 \\
Y \in \{1 \quad\quad 3 \quad\quad\} & y_{=1} + y_{=3} \geq 1 \\
Z \in \{1 \quad\quad 3 \quad\quad\} & z_{=1} + z_{=3} \geq 1
\end{array}$$

Justifying All-Different Failures

$$\begin{array}{rlllllllll}
V \in \{ & 1 & & & 4 & 5 \} & & & & \\
W \in \{ & 1 & 2 & 3 & & \} & w_{=1} + & w_{=2} + & w_{=3} & & & \geq 1 \\
X \in \{ & & 2 & 3 & & \} & & x_{=2} + & x_{=3} & & & \geq 1 \\
Y \in \{ & 1 & & 3 & & \} & y_{=1} & + & y_{=3} & & & \geq 1 \\
Z \in \{ & 1 & & 3 & & \} & z_{=1} & + & z_{=3} & & & \geq 1
\end{array}$$

$$\begin{array}{l}
\rightarrow \quad -v_{=1} + -w_{=1} + \qquad -y_{=1} + -z_{=1} \geq -1 \\
\quad \rightarrow \quad -w_{=2} + -x_{=2} \geq -1 \\
\qquad \rightarrow \quad -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1
\end{array}$$

Justifying All-Different Failures

$$\begin{array}{llcr}
V \in \{\, 1 \qquad\quad 4 \quad 5 \,\} & & & \\
W \in \{\, 1 \quad 2 \quad 3 \qquad\quad \} & w_{=1} + w_{=2} + w_{=3} & & \geq 1 \\
X \in \{\, \quad\; 2 \quad 3 \qquad\quad \} & x_{=2} + x_{=3} & & \geq 1 \\
Y \in \{\, 1 \qquad\; 3 \qquad\quad \} & y_{=1} + y_{=3} & & \geq 1 \\
Z \in \{\, 1 \qquad\; 3 \qquad\quad \} & z_{=1} + z_{=3} & & \geq 1 \\
 & & & \\
\rightarrow & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} & & \geq -1 \\
\quad \rightarrow & -w_{=2} + -x_{=2} & & \geq -1 \\
\qquad \rightarrow & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} & & \geq -1 \\
 & & & \\
 & -v_{=1} & & \geq 1
\end{array}$$

Justifying All-Different Failures

$$\begin{array}{rl}
V \in \{\, 1 \qquad\quad 4 \quad 5 \,\} & \\
W \in \{\, 1 \quad 2 \quad 3 \qquad\quad \} & w_{=1} + w_{=2} + w_{=3} \geq 1 \\
X \in \{\, \quad\; 2 \quad 3 \qquad\quad \} & x_{=2} + x_{=3} \geq 1 \\
Y \in \{\, 1 \qquad\; 3 \qquad\quad \} & y_{=1} + y_{=3} \geq 1 \\
Z \in \{\, 1 \qquad\; 3 \qquad\quad \} & z_{=1} + z_{=3} \geq 1
\end{array}$$

$$\begin{array}{rl}
\longrightarrow & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} \geq -1 \\
\longrightarrow & -w_{=2} + -x_{=2} \geq -1 \\
\longrightarrow & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1
\end{array}$$

$$\begin{array}{l}
-v_{=1} \geq 1 \\
v_{=1} \geq 0
\end{array}$$

Justifying All-Different Failures

$$\begin{array}{ll}
V \in \{\, 1 \qquad\quad 4 \quad 5 \,\} & \\
W \in \{\, 1 \quad 2 \quad 3 \qquad\quad \,\} & w_{=1} + w_{=2} + w_{=3} \geq 1 \\
X \in \{\, \quad\; 2 \quad 3 \qquad\quad \,\} & x_{=2} + x_{=3} \geq 1 \\
Y \in \{\, 1 \qquad\; 3 \qquad\quad \,\} & y_{=1} + y_{=3} \geq 1 \\
Z \in \{\, 1 \qquad\; 3 \qquad\quad \,\} & z_{=1} + z_{=3} \geq 1
\end{array}$$

$$\begin{array}{ll}
\rightarrow & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} \geq -1 \\
\quad\rightarrow & -w_{=2} + -x_{=2} \geq -1 \\
\qquad\rightarrow & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1
\end{array}$$

$$\begin{array}{l}
-v_{=1} \geq 1 \\
v_{=1} \geq 0
\end{array}$$

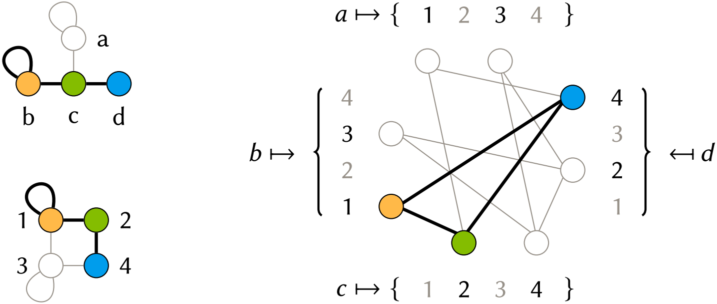
$$0 \geq 1$$

Reformulation

Auto-tabulation is possible.

- Heavy use of extension variables.

Can re-encode maximum common subgraph as a clique problem, without changing the pseudo-Boolean model.



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?

Future research…

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).
- Table.
- Minimum / maximum of an array.
- Element.
- Absolute value.

Details in [GMN22].

What’s Left?

- The truth about extension variables (redundance rule)
- Some applications of this rule (parity reasoning & PB-to-CNF translations)
- Extensions of the redundance rule to optimization
- Symmetry Breaking

The Truth About Extension Variables

Recall: we want new, fresh variable *a* encoding

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

Introduce clauses

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

Or, in pseudo-Boolean language, constraints

$$a + \overline{x} + \overline{y} \geq 1 \qquad 2\overline{a} + x + y \geq 2$$

Resolution and cutting planes proof system inherently cannot certify such derivations: they are not implied!

Redundance-Based Strengthening

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Adding redundant constraints should be OK

Redundance-Based Strengthening

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Adding redundant constraints should be OK

Redundance-based strengthening [BT19, GN21] (extending RAT)

C is redundant with respect to F iff there is a substitution ω (mapping variables to truth values or literals), called a witness, for which

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

Redundance-Based Strengthening

Fact

α satisfies $\phi\restriction_\omega$ iff $\alpha \circ \omega$ satisfies ϕ

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Adding redundant constraints should be OK

Redundance-based strengthening [BT19, GN21] (extending RAT)

C is redundant with respect to F iff there is a substitution ω (mapping variables to truth values or literals), called a witness, for which

$$F \wedge \neg C \models (F \wedge C)\restriction_\omega$$

Redundance-Based Strengthening

Fact

α satisfies $\phi\restriction_{\omega}$ iff $\alpha \circ \omega$ satisfies ϕ

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Adding redundant constraints should be OK

Redundance-based strengthening [BT19, GN21] (extending RAT)

C is redundant with respect to F iff there is a substitution ω (mapping variables to truth values or literals), called a witness, for which

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

Proof sketch for interesting direction: If α satisfies F but falsifies C, then $\alpha \circ \omega$ satisfies $F \wedge C$

Redundance-Based Strengthening

Fact

α satisfies $\phi\restriction_\omega$ iff $\alpha \circ \omega$ satisfies ϕ

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Adding redundant constraints should be OK

Redundance-based strengthening [BT19, GN21] (extending RAT)

C is redundant with respect to F iff there is a substitution ω (mapping variables to truth values or literals), called a witness, for which

$$F \wedge \neg C \models (F \wedge C)\restriction_\omega$$

Proof sketch for interesting direction: If α satisfies F but falsifies C, then $\alpha \circ \omega$ satisfies $F \wedge C$

Witness ω should be specified, and implication be efficiently verifiable (which is the case, e.g., if all constraints in $(F \wedge C)\restriction_\omega$ are RUP)

Deriving $a \Leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$a + \overline{x} + \overline{y} \geq 1 \qquad 2\overline{a} + x + y \geq 2$$

using condition $F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$

Deriving $a \Leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$a + \overline{x} + \overline{y} \geq 1 \qquad 2\overline{a} + x + y \geq 2$$

using condition $F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$

1. $F \wedge \neg(2\overline{a} + x + y \geq 2) \models (F \wedge (2\overline{a} + x + y \geq 2))\restriction_{\omega}$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new constraint satisfied

Deriving $a \Leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

$$a + \overline{x} + \overline{y} \geq 1 \qquad 2\overline{a} + x + y \geq 2$$

using condition $F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$

1. $F \wedge \neg(2\overline{a} + x + y \geq 2) \models (F \wedge (2\overline{a} + x + y \geq 2))\restriction_{\omega}$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new constraint satisfied
2. $F \wedge (2\overline{a} + x + y \geq 2) \wedge \neg(a + \overline{x} + \overline{y} \geq 1) \models$
   $(F \wedge (2\overline{a} + x + y \geq 2) \wedge (a + \overline{x} + \overline{y} \geq 1))\restriction_{\omega}$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new constraint satisfied
   $\neg(a + \overline{x} + \overline{y} \geq 1)$ forces $x \mapsto 1$ and $y \mapsto 1$, hence $2\overline{a} + x + y \geq 2$ remains satisfied after forcing a to be true

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$s_{4,2}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$s_{4,2}$

How to know translation is correct?

VeriPB can certify pseudo-Boolean-to-CNF rewriting

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

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

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]

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

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

Use redundance rule with fresh variables a, b to 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

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

Use redundance rule with fresh variables a, b to 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

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

Use redundance rule with fresh variables a, b to 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

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

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

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

Use redundance rule with fresh variables a, b to 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]

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models (F \wedge C)\restriction_{\omega} \wedge f\restriction_{\omega} \leq f$$

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \land \neg C \models (F \land C)\restriction_{\omega} \ \land \ f\restriction_{\omega} \leq f$$

Can be more aggressive if witness ω strictly improves solution.

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models (F \wedge C)\restriction_{\omega} \; \wedge \; f\restriction_{\omega} \leq f$$

Can be more aggressive if witness ω strictly improves solution.

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models F\restriction_{\omega} \; \wedge \; f\restriction_{\omega} < f$$

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \land \neg C \models (F \land C)\upharpoonright_{\omega} \land f\upharpoonright_{\omega} \leq f$$

Can be more aggressive if witness ω strictly improves solution.

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \land \neg C \models F\upharpoonright_{\omega} \land f\upharpoonright_{\omega} < f$$

- Applying ω should strictly decrease f.
- If so, don't need to show that $C\upharpoonright_{\omega}$ holds!

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

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

Why is this sound?

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \land \neg C \models F\restriction_\omega \ \land \ f\restriction_\omega < f$$

Why is this sound?

1. Suppose α satisfies F but falsifies C (i.e. satisfies $\neg C$).

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

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

Why is this sound?

1. Suppose α satisfies F but falsifies C (i.e. satisfies $\neg C$).
2. Then $\alpha \circ \omega$ satisfies F and $f(\alpha \circ \omega) < f(\alpha)$.

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

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

Why is this sound?

1. Suppose α satisfies F but falsifies C (i.e. satisfies $\neg C$).
2. Then $\alpha \circ \omega$ satisfies F and $f(\alpha \circ \omega) < f(\alpha)$.
3. If $\alpha \circ \omega$ satisfies C, we're done.

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models F\restriction_\omega \; \wedge \; f\restriction_\omega < f$$

Why is this sound?

1. Suppose α satisfies F but falsifies C (i.e. satisfies $\neg C$).
2. Then $\alpha \circ \omega$ satisfies F and $f(\alpha \circ \omega) < f(\alpha)$.
3. If $\alpha \circ \omega$ satisfies C, we're done.
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies F and $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$.

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models F\upharpoonright_{\omega} \wedge f\upharpoonright_{\omega} < f$$

Why is this sound?

1. Suppose α satisfies F but falsifies C (i.e. satisfies $\neg C$).
2. Then $\alpha \circ \omega$ satisfies F and $f(\alpha \circ \omega) < f(\alpha)$.
3. If $\alpha \circ \omega$ satisfies C, we're done.
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies F and $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$.
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done.

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

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

Why is this sound?

1. Suppose α satisfies F but falsifies C (i.e. satisfies $\neg C$).
2. Then $\alpha \circ \omega$ satisfies F and $f(\alpha \circ \omega) < f(\alpha)$.
3. If $\alpha \circ \omega$ satisfies C, we're done.
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies F and $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$.
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done.
6. Otherwise $\big((\alpha \circ \omega) \circ \omega\big) \circ \omega$ satisfies F and $f\big(((\alpha \circ \omega) \circ \omega) \circ \omega\big) < f\big((\alpha \circ \omega) \circ \omega\big)$.

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \wedge \neg C \models F\restriction_\omega \ \wedge \ f\restriction_\omega < f$$

Why is this sound?

1. Suppose α satisfies F but falsifies C (i.e. satisfies $\neg C$).
2. Then $\alpha \circ \omega$ satisfies F and $f(\alpha \circ \omega) < f(\alpha)$.
3. If $\alpha \circ \omega$ satisfies C, we're done.
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies F and $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$.
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done.
6. Otherwise $\big((\alpha \circ \omega) \circ \omega\big) \circ \omega$ satisfies F and $f\big(((\alpha \circ \omega) \circ \omega) \circ \omega\big) < f\big((\alpha \circ \omega) \circ \omega\big)$.
7. ...

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint C to formula F if exists witness substitution ω s.t.

$$F \land \neg C \models F\restriction_\omega \ \land\ f\restriction_\omega < f$$

Why is this sound?

1. Suppose α satisfies F but falsifies C (i.e. satisfies $\neg C$).
2. Then $\alpha \circ \omega$ satisfies F and $f(\alpha \circ \omega) < f(\alpha)$.
3. If $\alpha \circ \omega$ satisfies C, we're done.
4. Otherwise $(\alpha \circ \omega) \circ \omega$ satisfies F and $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$.
5. If $(\alpha \circ \omega) \circ \omega$ satisfies C, we're done.
6. Otherwise $\big((\alpha \circ \omega) \circ \omega\big) \circ \omega$ satisfies F and $f\big(((\alpha \circ \omega) \circ \omega) \circ \omega\big) < f\big((\alpha \circ \omega) \circ \omega\big)$.
7. …
8. Can't go on forever, so finally reach α' satisfying $F \land C$.

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

If $C_1, C_2, \ldots, C_{m-1}$ have been derived from F (maybe using dominance), then can derive C_m if exists witness substitution ω s.t.

$$F \land \bigwedge_{i=1}^{m-1} C_i \land \neg C_m \models F\!\upharpoonright_\omega \land f\!\upharpoonright_\omega < f$$

Only consider F — no need to show that any $C_i\!\upharpoonright_\omega$ implied!

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

If $C_1, C_2, \ldots, C_{m-1}$ have been derived from F (maybe using dominance), then can derive C_m if exists witness substitution ω s.t.

$$F \wedge \bigwedge_{i=1}^{m-1} C_i \wedge \neg C_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Only consider F — no need to show that any $C_i\restriction_\omega$ implied!

Now why is *this* sound?

- Same inductive proof as before, but nested.

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

If $C_1, C_2, \ldots, C_{m-1}$ have been derived from F (maybe using dominance), then can derive C_m if exists witness substitution ω s.t.

$$F \wedge \bigwedge_{i=1}^{m-1} C_i \wedge \neg C_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Only consider F — no need to show that any $C_i\restriction_\omega$ implied!

Now why is *this* sound?

- Same inductive proof as before, but nested.
- Or pick solution α minimizing f and argue by contradiction.

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

If $C_1, C_2, \ldots, C_{m-1}$ have been derived from F (maybe using dominance), then can derive C_m if exists witness substitution ω s.t.

$$F \wedge \bigwedge_{i=1}^{m-1} C_i \wedge \neg C_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Only consider F — no need to show that any $C_i\restriction_\omega$ implied!

Now why is *this* sound?

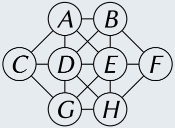
- Same inductive proof as before, but nested.
- Or pick solution α minimizing f and argue by contradiction.

Further extensions:

- Define dominance rule w.r.t. order independent of objective.
- Switch between different orders in same proof.
- See [BGMN22a] for details.

Symmetry Elimination (CP)

The Crystal Maze Puzzle



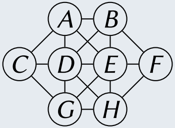
Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

Symmetry Elimination (CP)

Human modellers might add:

- $A < G$ (mirror vertically)
- $A < B$ (mirror horizontally)
- $A \leq 4$ (value symmetry)

The Crystal Maze Puzzle



Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

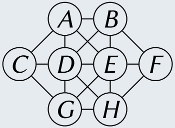
Symmetry Elimination (CP)

Human modellers might add:

- $A < G$ (mirror vertically)
- $A < B$ (mirror horizontally)
- $A \leq 4$ (value symmetry)

Are these valid simultaneously?

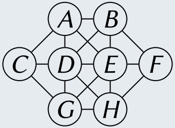
The Crystal Maze Puzzle



Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

Symmetry Elimination (CP)

The Crystal Maze Puzzle



Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

Human modellers might add:

- $A < G$ (mirror vertically)
- $A < B$ (mirror horizontally)
- $A \leq 4$ (value symmetry)

Are these valid simultaneously?

We can introduce these constraints inside the proof, rather than as part of the pseudo-Boolean model!

- Can use permutation of variable-values as the witness ω.
- The constraints give us the order.
- No group theory required!

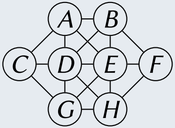
Symmetry Elimination (CP)

Human modellers might add:

- $A < G$ (mirror vertically)
- $A < B$ (mirror horizontally)
- $A \leq 4$ (value symmetry)

Are these valid simultaneously?

The Crystal Maze Puzzle



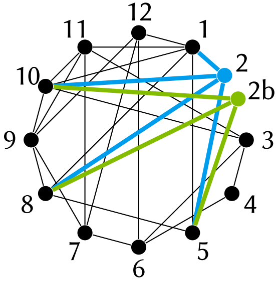
Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

We can introduce these constraints inside the proof, rather than as part of the pseudo-Boolean model!

- Can use permutation of variable-values as the witness ω.
- The constraints give us the order.
- No group theory required!

Research challenge: a CP toolchain supporting this.

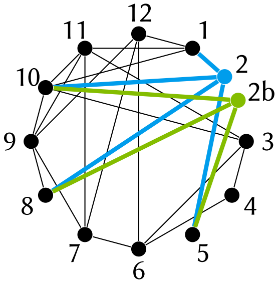
Lazy Global Domination For Maximum Clique [MP16]



Can ignore vertex 2b.

- Every neighbour of 2b is also a neighbour of 2.
- Not a symmetry, but a dominance.

Lazy Global Domination For Maximum Clique [MP16]



Can ignore vertex 2b.

- Every neighbour of 2b is also a neighbour of 2.
- Not a symmetry, but a dominance.

Dominance rule can justify this.

- Even when detected dynamically during search.

Strategy for SAT Symmetry Breaking

1. Pretend to solve optimisation problem minimizing
   $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$
   (search lexicographically smallest assignment satisfying formula)

Strategy for SAT Symmetry Breaking

1. Pretend to solve optimisation problem minimizing
   $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$
   (search lexicographically smallest assignment satisfying formula)
2. Derive pseudo-Boolean lex-leader constraint

$$C_\sigma \quad \doteq \quad f \leq f\restriction_\sigma \quad \doteq \quad \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

Strategy for SAT Symmetry Breaking

1. Pretend to solve optimisation problem minimizing $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$ (search lexicographically smallest assignment satisfying formula)
2. Derive pseudo-Boolean lex-leader constraint

$$C_\sigma \quad \doteq \quad f \leq f\restriction_\sigma \quad \doteq \quad \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

3. Derive CNF encoding of lex-leader constraints from PB constraint (in same spirit as [GMNO22])

$$\begin{array}{ll} y_0 & \overline{y}_j \vee \overline{\sigma(x_j)} \vee x_j \\ \overline{y}_{j-1} \vee \overline{x}_j \vee \sigma(x_j) & y_j \vee \overline{y}_{j-1} \vee \overline{x}_j \\ \overline{y}_j \vee y_{j-1} & y_j \vee \overline{y}_{j-1} \vee \sigma(x_j) \end{array}$$

Symmetry Breaking: Example

Example: Pigeonhole principle formula

- Variables p_{ij} $(1 \leq i \leq 4, 1 \leq j \leq 3)$ true iff pigeon i in hole j
- Focus on pigeon symmetries — notation:
  - $\sigma_{(12)}$ swaps pigeons 1 and 2

Symmetry Breaking: Example

Example: Pigeonhole principle formula

- Variables p_{ij} $(1 \leq i \leq 4, 1 \leq j \leq 3)$ true iff pigeon i in hole j
- Focus on pigeon symmetries — notation:
  - $\sigma_{(12)}$ swaps pigeons 1 and 2
    Formally: $\sigma_{(12)}(p_{1j}) = p_{2j}$ and $\sigma_{(12)}(p_{2j}) = p_{1j}$ for all j
  - $\sigma_{(1234)}$ shifts all pigeons

Symmetry Breaking: Example

Example: Pigeonhole principle formula

- Variables p_{ij} $(1 \leq i \leq 4, 1 \leq j \leq 3)$ true iff pigeon i in hole j
- Focus on pigeon symmetries — notation:
  - $\sigma_{(12)}$ swaps pigeons 1 and 2
    Formally: $\sigma_{(12)}(p_{1j}) = p_{2j}$ and $\sigma_{(12)}(p_{2j}) = p_{1j}$ for all j
  - $\sigma_{(1234)}$ shifts all pigeons

Order: “Pigeon 1 preferred in the smallest hole; next pigeon 2, ...”

$$f \doteq 2^{11} \cdot p_{13} + 2^{10} \cdot p_{12} + 2^{9} \cdot p_{11} + 2^{8} \cdot p_{23} + \cdots + 1 \cdot p_{41}$$

Breaking a Single Simple Symmetry (Example)

- F is a formula expressing PHP constraints with $F\restriction_{\sigma_{(12)}} = F$
- Want to add constraint C_{12} breaking $\sigma_{(12)}$ — should be satisfied by α iff α "at least as good" as $\sigma_{(12)}(\alpha)$

Breaking a Single Simple Symmetry (Example)

- F is a formula expressing PHP constraints with $F\restriction_{\sigma_{(12)}} = F$
- Want to add constraint C_{12} breaking $\sigma_{(12)}$ — should be satisfied by α iff α "at least as good" as $\sigma_{(12)}(\alpha)$

$$C_{12} \doteq f \leq f\restriction_{\sigma_{(12)}}$$

Breaking a Single Simple Symmetry (Example)

- F is a formula expressing PHP constraints with $F\restriction_{\sigma_{(12)}} = F$
- Want to add constraint C_{12} breaking $\sigma_{(12)}$ — should be satisfied by α iff α "at least as good" as $\sigma_{(12)}(\alpha)$

$$\begin{aligned} C_{12} &\doteq f \leq f\restriction_{\sigma_{(12)}} \\ &\doteq \textstyle\sum_{i=1}^{n} 2^{n-i} \cdot \left(\sigma_{(12)}(x_i) - x_i\right) \geq 0 \end{aligned}$$

Breaking a Single Simple Symmetry (Example)

- F is a formula expressing PHP constraints with $F\restriction_{\sigma_{(12)}} = F$
- Want to add constraint C_{12} breaking $\sigma_{(12)}$ — should be satisfied by α iff α "at least as good" as $\sigma_{(12)}(\alpha)$

$$\begin{aligned} C_{12} &\doteq f \leq f\restriction_{\sigma_{(12)}} \\ &\doteq \textstyle\sum_{i=1}^{n} 2^{n-i} \cdot \left(\sigma_{(12)}(x_i) - x_i\right) \geq 0 \\ &\doteq \left(2^{11}-2^{8}\right)(p_{23}-p_{13}) + \left(2^{10}-2^{7}\right)(p_{22}-p_{12}) + \left(2^{9}-2^{6}\right)(p_{21}-p_{11}) \geq 0 \end{aligned}$$

"Pigeon 1 in smaller hole than pigeon 2"

Breaking a Single Simple Symmetry (Example)

- F is a formula expressing PHP constraints with $F\restriction_{\sigma_{(12)}} = F$
- Want to add constraint C_{12} breaking $\sigma_{(12)}$ — should be satisfied by α iff α "at least as good" as $\sigma_{(12)}(\alpha)$

$$\begin{aligned} C_{12} &\doteq f \leq f\restriction_{\sigma_{(12)}} \\ &\doteq \textstyle\sum_{i=1}^{n} 2^{n-i} \cdot \left(\sigma_{(12)}(x_i) - x_i\right) \geq 0 \\ &\doteq \left(2^{11}-2^{8}\right)(p_{23}-p_{13}) + \left(2^{10}-2^{7}\right)(p_{22}-p_{12}) + \left(2^{9}-2^{6}\right)(p_{21}-p_{11}) \geq 0 \end{aligned}$$

"Pigeon 1 in smaller hole than pigeon 2"

- Can use redundance rule (the symmetry is the witness):

$$F \land \neg C_{12} \models F\restriction_{\sigma_{(12)}} \land C_{12}\restriction_{\sigma_{(12)}} \land f\restriction_{\sigma_{(12)}} \leq f$$

$$F \land \neg\left(f \leq f\restriction_{\sigma_{(12)}}\right) \models F\restriction_{\sigma_{(12)}} \land \left(f \leq f\restriction_{\sigma_{(12)}}\right)\restriction_{\sigma_{(12)}} \land f\restriction_{\sigma_{(12)}} \leq f$$

Breaking a Single Simple Symmetry (Example)

- F is a formula expressing PHP constraints with $F\restriction_{\sigma_{(12)}} = F$
- Want to add constraint C_{12} breaking $\sigma_{(12)}$ — should be satisfied by α iff α “at least as good” as $\sigma_{(12)}(\alpha)$

$$\begin{aligned} C_{12} &\doteq f \leq f\restriction_{\sigma_{(12)}} \\ &\doteq \textstyle\sum_{i=1}^{n} 2^{n-i} \cdot \left(\sigma_{(12)}(x_i) - x_i\right) \geq 0 \\ &\doteq \left(2^{11}-2^{8}\right)(p_{23}-p_{13}) + \left(2^{10}-2^{7}\right)(p_{22}-p_{12}) + \left(2^{9}-2^{6}\right)(p_{21}-p_{11}) \geq 0 \end{aligned}$$

“Pigeon 1 in smaller hole than pigeon 2”

- Can use redundance rule (the symmetry is the witness):

$$F \wedge \neg C_{12} \models F\restriction_{\sigma_{(12)}} \wedge C_{12}\restriction_{\sigma_{(12)}} \wedge f\restriction_{\sigma_{(12)}} \leq f$$

$$F \wedge f > f\restriction_{\sigma_{(12)}} \models F\restriction_{\sigma_{(12)}} \wedge f\restriction_{\sigma_{(12)}} \leq f \wedge f\restriction_{\sigma_{(12)}} \leq f$$

Breaking a Single Simple Symmetry (Example)

- F is a formula expressing PHP constraints with $F\restriction_{\sigma_{(12)}} = F$
- Want to add constraint C_{12} breaking $\sigma_{(12)}$ — should be satisfied by α iff α "at least as good" as $\sigma_{(12)}(\alpha)$

$$\begin{aligned} C_{12} &\doteq f \leq f\restriction_{\sigma_{(12)}} \\ &\doteq \textstyle\sum_{i=1}^{n} 2^{n-i} \cdot \left(\sigma_{(12)}(x_i) - x_i\right) \geq 0 \\ &\doteq \left(2^{11}-2^{8}\right)(p_{23}-p_{13}) + \left(2^{10}-2^{7}\right)(p_{22}-p_{12}) + \left(2^{9}-2^{6}\right)(p_{21}-p_{11}) \geq 0 \end{aligned}$$

"Pigeon 1 in smaller hole than pigeon 2"

- Can use redundance rule (the symmetry is the witness):

$$F \wedge \neg C_{12} \models F\restriction_{\sigma_{(12)}} \wedge C_{12}\restriction_{\sigma_{(12)}} \wedge f\restriction_{\sigma_{(12)}} \leq f$$

$$F \wedge f > f\restriction_{\sigma_{(12)}} \models F\restriction_{\sigma_{(12)}} \wedge f\restriction_{\sigma_{(12)}} \leq f \wedge f\restriction_{\sigma_{(12)}} \leq f$$

Similar to DRAT symmetry breaking [HHW15]

Breaking More/Other symmetries

Problem

This idea does not generalize.

- Breaking two symmetries

- Breaking complex symmetries

Breaking More/Other symmetries

Problem

This idea does not generalize.

- Breaking two symmetries

  $$F \land C_{12} \land \neg C_{23} \not\models F\restriction_{\sigma_{(23)}} \land C_{12}\restriction_{\sigma_{(23)}} \land C_{23}\restriction_{\sigma_{(23)}} \land f\restriction_{\sigma_{(23)}} \leq f$$

  Intuitively: applying $\sigma_{(23)}$ potentially falsifies C_{12}

- Breaking complex symmetries

Breaking More/Other symmetries

Problem

This idea does not generalize.

- Breaking two symmetries

$$F \wedge C_{12} \wedge \neg C_{23} \not\models F\restriction_{\sigma_{(23)}} \wedge C_{12}\restriction_{\sigma_{(23)}} \wedge C_{23}\restriction_{\sigma_{(23)}} \wedge f\restriction_{\sigma_{(23)}} \leq f$$

Intuitively: applying $\sigma_{(23)}$ potentially falsifies C_{12}
We might have to apply $\sigma_{(12)}$ again

- Breaking complex symmetries

Breaking More/Other symmetries

Problem

This idea does not generalize.

- Breaking two symmetries

$$F \wedge C_{12} \wedge \neg C_{23} \not\models F\restriction_{\sigma_{(23)}} \wedge C_{12}\restriction_{\sigma_{(23)}} \wedge C_{23}\restriction_{\sigma_{(23)}} \wedge f\restriction_{\sigma_{(23)}} \leq f$$

Intuitively: applying $\sigma_{(23)}$ potentially falsifies C_{12}
We might have to apply $\sigma_{(12)}$ again

- Breaking complex symmetries

$$F \wedge \neg C_{1234} \models F\restriction_{\sigma_{(1234)}} \wedge C_{1234}\restriction_{\sigma_{(1234)}} \wedge f\restriction_{\sigma_{(1234)}} \leq f$$

Intuitively, C_{1234} holds if shifting all the pigeons results in a worse assignment.

Breaking More/Other symmetries

Problem

This idea does not generalize.

- Breaking two symmetries

$$F \wedge C_{12} \wedge \neg C_{23} \not\models F\restriction_{\sigma_{(23)}} \wedge C_{12}\restriction_{\sigma_{(23)}} \wedge C_{23}\restriction_{\sigma_{(23)}} \wedge f\restriction_{\sigma_{(23)}} \leq f$$

Intuitively: applying $\sigma_{(23)}$ potentially falsifies C_{12}
We might have to apply $\sigma_{(12)}$ again

- Breaking complex symmetries

$$F \wedge \neg C_{1234} \models F\restriction_{\sigma_{(1234)}} \wedge C_{1234}\restriction_{\sigma_{(1234)}} \wedge f\restriction_{\sigma_{(1234)}} \leq f$$

Intuitively, C_{1234} holds if shifting all the pigeons results in a worse assignment.
Can "restore" its truth by applying $\sigma_{(1234)}$ once, twice, or thrice.

Breaking Symmetries With the Dominance Rule (1/2)

Definition

Given a symmetry σ, the (pseudo-Boolean) breaking constraint of σ is

$$C_\sigma \doteq f \leq f\restriction_\sigma$$

Breaking Symmetries With the Dominance Rule (1/2)

Definition

Given a symmetry σ, the (pseudo-Boolean) breaking constraint of σ is

$$C_\sigma \doteq f \leq f\restriction_\sigma$$

Theorem

C_σ can be derived from F using dominance with witness σ

$$F \wedge \neg C_\sigma \models F\restriction_\sigma \wedge f\restriction_\sigma < f$$

Breaking Symmetries With the Dominance Rule (2/2)

Breaking symmetries with the dominance rule

- Surprisingly simple

Breaking Symmetries With the Dominance Rule (2/2)

Breaking symmetries with the dominance rule

- Surprisingly simple
- Generalizes well

Breaking Symmetries With the Dominance Rule (2/2)

Breaking symmetries with the dominance rule

- Surprisingly simple
- Generalizes well
    - Works for arbitrary symmetries

Breaking Symmetries With the Dominance Rule (2/2)

Breaking symmetries with the dominance rule

- Surprisingly simple
- Generalizes well
    - Works for arbitrary symmetries
    - Works for multiple symmetries (ignore previously derived constraints)

$$F \wedge C_{12} \wedge \neg C_{23} \models F\restriction_{\sigma_{(23)}} \wedge f\restriction_{\sigma_{(23)}} < f$$

Breaking Symmetries With the Dominance Rule (2/2)

Breaking symmetries with the dominance rule

- Surprisingly simple
- Generalizes well
  - Works for arbitrary symmetries
  - Works for multiple symmetries (ignore previously derived constraints)

$$F \wedge C_{12} \wedge \neg C_{23} \models F\restriction_{\sigma_{(23)}} \wedge f\restriction_{\sigma_{(23)}} < f$$

Why does it work?

- Witness need not satisfy all derived constraints
- Sufficient to just produce “better” assignment

Making Your Solver Output Proofs

The VeriPB proof verifier lives at

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

And it's documented!

See [GMM⁺20, EGMN20, BGMN22b, GN22, GMN22] for worked examples, and even more in Stephan Gocht's PhD thesis [Goc22].

We're happy to collaborate with you! And we're hiring!

Challenges and Work In Progress

Verification:

- Formally verified encoding and proof checking.
- Performance.

Challenges and Work In Progress

Verification:

- Formally verified encoding and proof checking.
- Performance.

Proof-related:

- “Lemmas”, or substitution proofs?
- Approximate counting, uniform sampling, etc? Pareto fronts?
- Proof trimming or minimisation?

Challenges and Work In Progress

Verification:

- Formally verified encoding and proof checking.
- Performance.

Proof-related:

- "Lemmas", or substitution proofs?
- Approximate counting, uniform sampling, etc? Pareto fronts?
- Proof trimming or minimisation?

Things to proof log:

- Every single dedicated solving algorithm ever.
- The 400 remaining global constraints not implemented yet
- CP symmetries, dynamic symmetry handling, . . .
- MaxSAT, MIP, SMT, . . .

Challenges and Work In Progress

Verification:

- Formally verified encoding and proof checking.
- Performance.

Proof-related:

- “Lemmas”, or substitution proofs?
- Approximate counting, uniform sampling, etc? Pareto fronts?
- Proof trimming or minimisation?

Things to proof log:

- Every single dedicated solving algorithm ever.
- The 400 remaining global constraints not implemented yet
- CP symmetries, dynamic symmetry handling, …
- MaxSAT, MIP, SMT, …

The end.

Challenges and Work In Progress

Verification:

- Formally verified encoding and proof checking.
- Performance.

Proof-related:

- “Lemmas”, or substitution proofs?
- Approximate counting, uniform sampling, etc? Pareto fronts?
- Proof trimming or minimisation?

Things to proof log:

- Every single dedicated solving algorithm ever.
- The 400 remaining global constraints not implemented yet
- CP symmetries, dynamic symmetry handling, …
- MaxSAT, MIP, SMT, …

The end. Or rather, the beginning!

References I

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

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

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

[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.

[Bat68] Kenneth E. Batcher. Sorting networks and their applications. In *Proceedings of the Spring Joint Computer Conference of the American Federation of Information Processing Societies (AFIPS '68)*, volume 32, pages 307–314, April 1968.

[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.

[BGMN22a] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI '22)*, pages 3698–3707, February 2022.

[BGMN22b] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. Technical Report 2203.12275, arXiv.org, March 2022.

References II

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

[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.

[BN21] Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Biere et al. [BHvMW21], chapter 7, pages 233–350.

[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.

[BT19] Samuel R. Buss and Neil Thapen. DRAT proofs, propagation redundancy, and extended resolution. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT '19)*, volume 11628 of *Lecture Notes in Computer Science*, pages 71–89. Springer, July 2019.

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

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

References III

[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/`.

[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.

[Dub20] Catherine Dubois. Formally verified constraints solvers: a guided tour. CICM. Invited talk, 2020.

[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.

References IV

[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.

[GMN22] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, volume 235 of *LIPIcs*, pages 25:1–25:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 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.

[GN22] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. Technical Report 2209.12185, arXiv.org, September 2022.

References V

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

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

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

[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.

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

[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.

References VI

[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.

[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.

[MP16] Ciaran McCreesh and Patrick Prosser. Finding maximum k-cliques faster using lazy global domination. In *Proceedings of the 9th Annual Symposium on Combinatorial Search (SOCS '16)*, pages 72–80, July 2016.

[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*.

[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.

References VII

[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.

[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.

[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`.

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

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

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

https://github.com/ciaranm/glasgow-constraint-solver

https://github.com/ciaranm/glasgow-subgraph-solver

https://bitbucket.org/krr/breakid

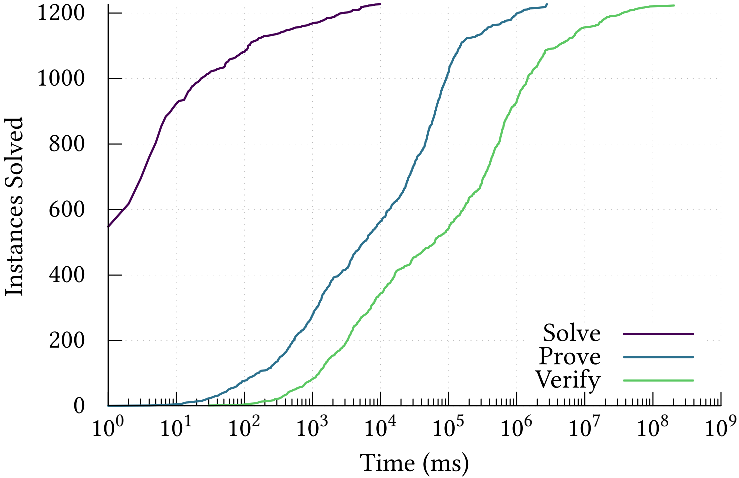
Clique Results

- 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.

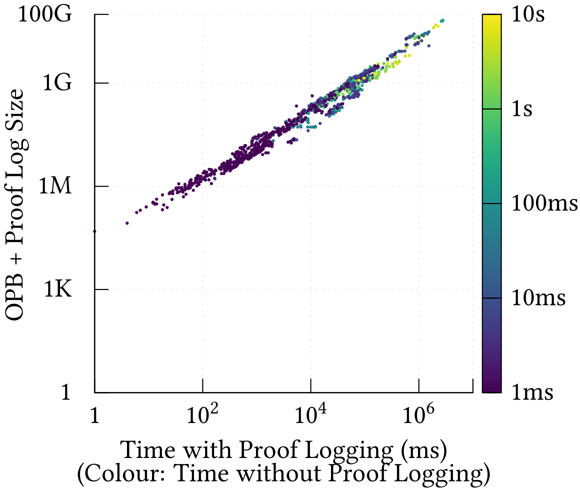
Subgrap Isomorphism Results

- 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.

Subgrap Isomorphism Results



Subgrap Isomorphism Results



How Expensive is Proof Logging?

- Laurent D. Michel, Pierre Schaus, Pascal Van Hentenryck: MiniCP: a lightweight solver for constraint programming. Math. Program. Comput. 13(1) (2021).
- 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 (although we don’t have circuit yet).
- Probably close to the worst case for proof logging performance.
- Also: Crystal Maze and World’s Hardest Sudoku.

How Expensive is Proof Logging?

- Our solver: faster than the fastest of MiniCP, OscaR, and Choco.
- Proof logging slowdown: between 8.4 to 61.1.
  - 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 10 to 100.
  - Probably possible to reduce this substantially if we are prepared to put more care into writing proofs.

PB-to-CNF Translation: Experiments

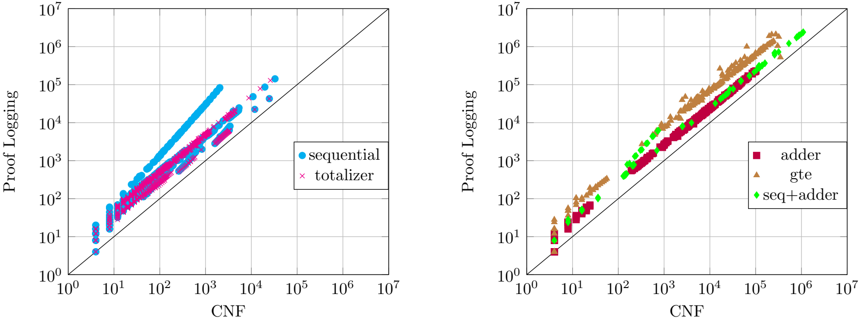
- Certified translations for the following CNF encodings:[2]
  - Sequential counter [Sin05]
  - Totalizer [BB03]
  - Generalized totalizer [JMM15]
  - Adder network [ES06]
- Proof verified by proof checker VeriPB
- Benchmarks from PB 2016 Evaluation:[3]
  - `SMALLINT` decision benchmarks without purely clausal formulas
  - 3 subclasses of benchmarks:
    - 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)

---

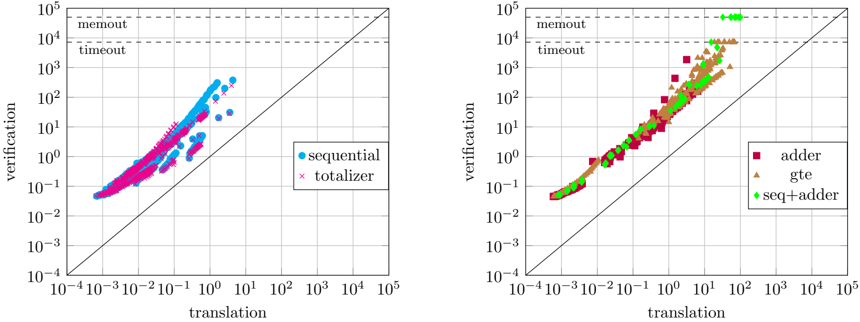
[2] `https://github.com/forge-lab/VeritasPBLib`

[3] `http://www.cril.univ-artois.fr/PB16/`

PB-to-CNF: CNF Size vs Proof Size in KiB

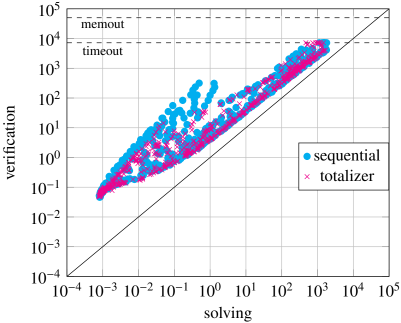


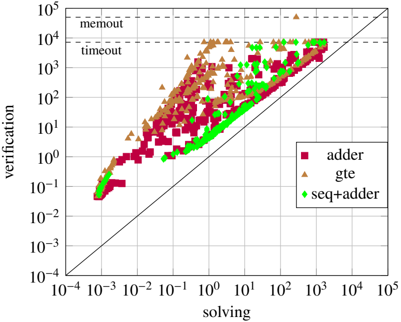
PB-to-CNF: Translation vs Verification Time in Seconds



- Translation just generates clauses and proof
- Verification slower, as reasoning has to be performed

PB-to-CNF: Solving Time vs Verification Time in Seconds





- Solved with fork of Kissat[4] syntactically modified to output pseudo-Boolean proofs
- Room for improvement, but clearly shows approach is viable

---

[4] `https://gitlab.com/MIAOresearch/tools-and-utilities/kissat_fork`

PB-to-CNF: Future Work

Improving performance:

- Cutting Planes derivations instead of reverse unit propagations [VDB22]
- Backwards checking/trimming for verification (as in DRAT-trim [HHW13a])

PB-to-CNF: Future Work

Improving performance:

- Cutting Planes derivations instead of reverse unit propagations [VDB22]
- Backwards checking/trimming for verification (as in DRAT-trim [HHW13a])

Extend proof logging further:

- Sorting networks like odd-even mergesort, bitonic sorter [Bat68]
- MaxSAT solving

Parity Reasoning: Experiments

Implemented parity reasoning and PB proof logging engine[5]

Also DRAT proof logging as described in [PR16]

Experiments with MiniSat[6]

Set-up:[7]

- Intel Core i5-1145G7 @2.60GHz × 4
- Memory limit 8GiB
- Disk write speed roughly 200 MiB/s
- Read speed of 2 GiB/s

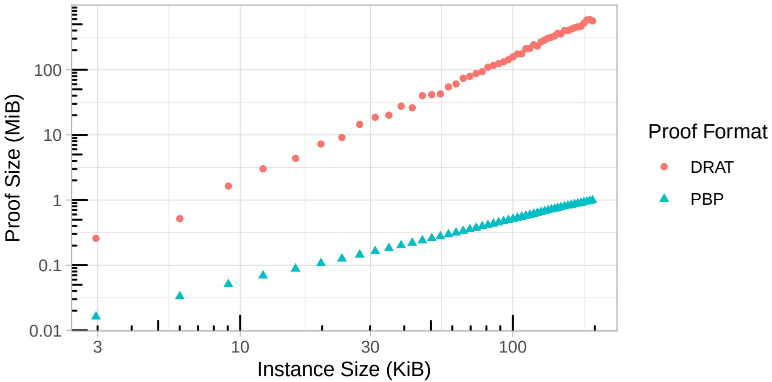
---

[5] `https://gitlab.com/MIAOresearch/tools-and-utlities/xorengine`

[6] `http://minisat.se/`

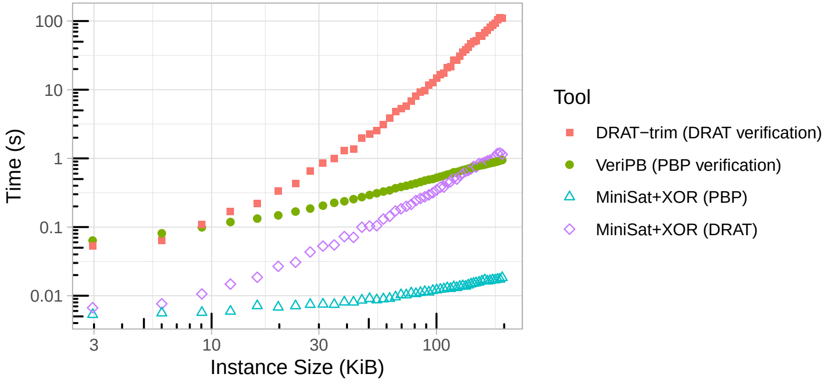
[7] Tools, benchmarks, data and evaluation scripts available at `https://doi.org/10.5281/zenodo.7083485`

Parity Reasoning: Proof Size



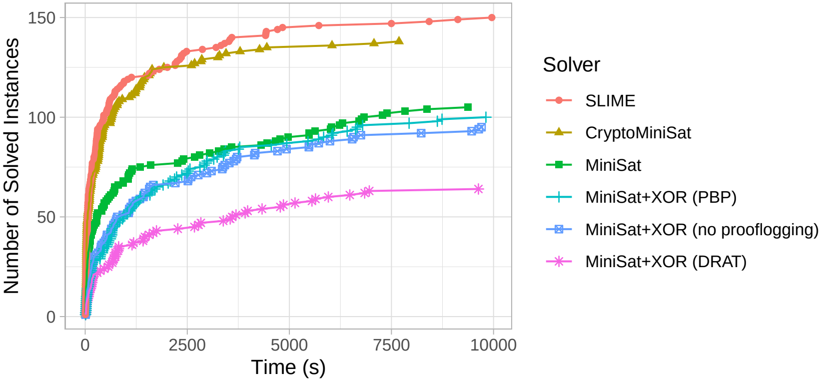
Proof sizes for Tseitin formulas using DRAT and PB proof logging

Parity Reasoning: Solving and Verification Time



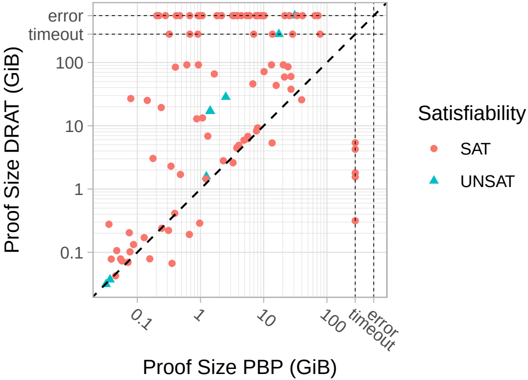
Solving and verification time for Tseitin formulas

Parity Reasoning: Crypto Track of SAT 2021 Competition



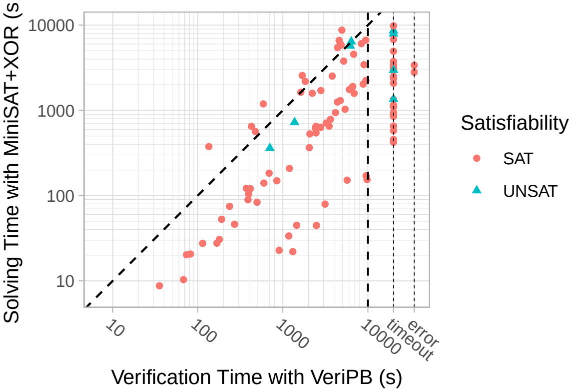
Cumulative plot for the crypto track of the SAT Competition 2021

Parity Reasoning: Crypto Track Proof Size



DRAT and PB proof sizes for crypto track of SAT Competition 2021

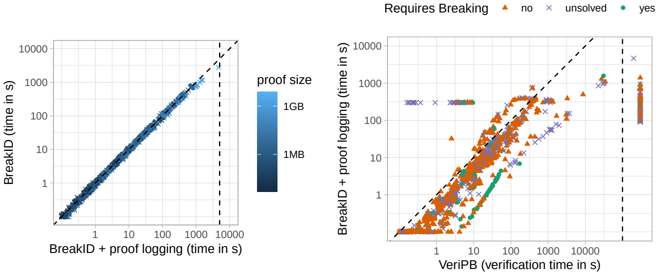
Parity Reasoning: Crypto Track Verification Time



Time required for solving and verifying crypto instances

Experimental Evaluation of SAT Symmetry Breaking

- Evaluated on SAT competition benchmarks
- BreakID [DBBD16, Bre] used to find and break symmetries



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

Strategy for SAT Symmetry Breaking

1. Pretend to solve optimisation problem minimizing $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$
(search lexicographically smallest assignment satisfying formula)

2. Derive pseudo-Boolean lex-leader constraint

$$C_\sigma \quad \doteq \quad f \leq f\restriction_\sigma \quad \doteq \quad \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

3. Derive CNF encoding of lex-leader constraints from PB constraint (in same spirit as [GMNO22])

$$\begin{array}{ll} y_0 & \overline{y}_j \vee \overline{\sigma(x_j)} \vee x_j \\ \overline{y}_{j-1} \vee \overline{x}_j \vee \sigma(x_j) & y_j \vee \overline{y}_{j-1} \vee \overline{x}_j \\ \overline{y}_j \vee y_{j-1} & y_j \vee \overline{y}_{j-1} \vee \sigma(x_j) \end{array}$$

Symmetry Breaking in CNF

- In SAT symmetry breakers, symmetry is broken in CNF

Symmetry Breaking in CNF

- In SAT symmetry breakers, symmetry is broken in CNF
- Still need to show how to derive CNF encoding

Symmetry Breaking in CNF

- In SAT symmetry breakers, symmetry is broken in CNF
- Still need to show how to derive CNF encoding
- We use the encoding of BreakID [DBBD16]:

$$y_0$$

$$\overline{y}_{j-1} \vee \overline{x}_j \vee \sigma(x_j)$$

$$\overline{y}_j \vee y_{j-1}$$

$$\overline{y}_j \vee \overline{\sigma(x_j)} \vee x_j$$

$$y_j \vee \overline{y}_{j-1} \vee \overline{x}_j$$

$$y_j \vee \overline{y}_{j-1} \vee \sigma(x_j)$$

Symmetry Breaking in CNF

- In SAT symmetry breakers, symmetry is broken in CNF
- Still need to show how to derive CNF encoding
- We use the encoding of BreakID [DBBD16]:

y_0

$\overline{y}_{j-1} \vee \overline{x}_j \vee \sigma(x_j)$

$\overline{y}_j \vee y_{j-1}$

$\overline{y}_j \vee \overline{\sigma(x_j)} \vee x_j$

$y_j \vee \overline{y}_{j-1} \vee \overline{x}_j$

$y_j \vee \overline{y}_{j-1} \vee \sigma(x_j)$

Define y_j to be true if x_k equals $\sigma(x_k)$ for all $k \leq j$

$$y_k \Leftrightarrow y_{k-1} \wedge (x_k \Leftrightarrow \sigma(x_k))$$

(derivable with redundance rule)

Symmetry Breaking in CNF

- In SAT symmetry breakers, symmetry is broken in CNF
- Still need to show how to derive CNF encoding
- We use the encoding of BreakID [DBBD16]:

Define y_j to be true if x_k equals $\sigma(x_k)$ for all $k \leq j$

$$y_k \Leftrightarrow y_{k-1} \wedge (x_k \Leftrightarrow \sigma(x_k))$$

$$y_0$$

$$\overline{y}_{j-1} \vee \overline{x}_j \vee \sigma(x_j)$$

$$\overline{y}_j \vee y_{j-1}$$

$$\overline{y}_j \vee \overline{\sigma(x_j)} \vee x_j$$

$$y_j \vee \overline{y}_{j-1} \vee \overline{x}_j$$

$$y_j \vee \overline{y}_{j-1} \vee \sigma(x_j)$$

(derivable with redundance rule) If y_k is true, x_k is at most $\sigma(x_k)$ (derivable from the PB breaking constraint)

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

Pseudo-Boolean breaking constraint

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

Derivable by redundance with witness $\omega : y_0 \mapsto 1$

$$F \wedge D \wedge \{\overline{y}_0\} \models (F \wedge D)\restriction_{\omega} \wedge \{y_0\}\restriction_{\omega}$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

Derivable by redundance with witness $\omega : y_0 \mapsto 1$

$$F \wedge D \wedge \{\overline{y}_0\} \models (F \wedge D)\restriction_\omega \wedge \{y_0\}\restriction_\omega$$

$$F \wedge \{\overline{y}_0\} \models (F \wedge D)\restriction_\omega \wedge \{1\}$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

Derivable by RUP

$$F \wedge D \wedge \neg(\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1))$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$y_0$$

$$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$$

Derivable by RUP

$$F \wedge D \wedge \neg(\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1))$$

$$= F \wedge D \wedge \{y_0 \wedge x_1 \wedge \overline{\sigma(x_1)}\}$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

Derivable by RUP

$$F \wedge D \wedge \neg(\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1))$$
$$= F \wedge D \wedge \{y_0 \wedge x_1 \wedge \overline{\sigma(x_1)}\}$$

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$y_0$$

$$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$$

Derivable by RUP

$$F \wedge D \wedge \neg(\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1))$$
$$= F \wedge D \wedge \{y_0 \wedge x_1 \wedge \overline{\sigma(x_1)}\}$$

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$2^{n-1} \cdot (-1) + \sum_{i=2}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$y_0$$

$$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$$

Derivable by RUP

$$F \wedge D \wedge \neg(\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1))$$

$$= F \wedge D \wedge \{y_0 \wedge x_1 \wedge \overline{\sigma(x_1)}\}$$

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$2^{n-1} \cdot (-1) + \sum_{i=2}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

with

$$\sum_{i=2}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \leq 2^{n-1} - 1$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

Derivable by redundance with witness $\omega : y_1 \mapsto 0$

$$F \wedge D \wedge \neg(\overline{y}_1 \vee y_0)$$
$$\models (F \wedge D)\restriction_{\omega} \wedge \{\overline{y}_1 \vee y_0\}\restriction_{\omega}$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

Derivable by redundance with witness $\omega : y_1 \mapsto 0$

$$F \wedge D \wedge \neg(\overline{y}_1 \vee y_0) \models (F \wedge D)\restriction_\omega \wedge \{\overline{y}_1 \vee y_0\}\restriction_\omega$$

$$F \wedge D \wedge \neg(\overline{y}_1 \vee y_0) \models (F \wedge D)\restriction_\omega \wedge \{1 \vee y_0\}$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$

Derivable by redundance with witness
$\omega : y_1 \mapsto 0$
(same argument)

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$

$y_1 \vee \overline{y}_0 \vee \overline{x}_1$

Derivable by redundance with witness $\omega : y_1 \mapsto 1$

$$F \wedge D \wedge \neg(y_1 \vee \overline{y}_0 \vee \overline{x}_1)$$
$$\models (F \wedge D)\restriction_\omega \wedge \{y_1 \vee \overline{y}_0 \vee \overline{x}_1\}\restriction_\omega$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$

$y_1 \vee \overline{y}_0 \vee \overline{x}_1$

Derivable by redundance with witness $\omega : y_1 \mapsto 1$

$$F \wedge D \wedge \neg(y_1 \vee \overline{y}_0 \vee \overline{x}_1)$$
$$\models (F \wedge D)\restriction_\omega \wedge \{y_1 \vee \overline{y}_0 \vee \overline{x}_1\}\restriction_\omega$$
$$F \wedge D \wedge \{\overline{y}_1 \wedge y_0 \wedge x_1)$$
$$\models \cdots \wedge D\restriction_\omega \wedge \ldots$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$

$y_1 \vee \overline{y}_0 \vee \overline{x}_1$

Derivable by redundance with witness $\omega : y_1 \mapsto 1$

$$F \wedge D \wedge \neg(y_1 \vee \overline{y}_0 \vee \overline{x}_1) \models (F \wedge D)\restriction_\omega \wedge \{y_1 \vee \overline{y}_0 \vee \overline{x}_1\}\restriction_\omega$$

$$F \wedge D \wedge \{\overline{y}_1 \wedge y_0 \wedge x_1) \models \cdots \wedge D\restriction_\omega \wedge \ldots$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$

$y_1 \vee \overline{y}_0 \vee \overline{x}_1$

Derivable by redundance with witness $\omega : y_1 \mapsto 1$

$$F \wedge D \wedge \neg(y_1 \vee \overline{y}_0 \vee \overline{x}_1)$$

$$\models (F \wedge D)\restriction_\omega \wedge \{y_1 \vee \overline{y}_0 \vee \overline{x}_1\}\restriction_\omega$$

$$F \wedge D \wedge \{\overline{y}_1 \wedge y_0 \wedge x_1)$$

$$\models \cdots \wedge D\restriction_\omega \wedge \ldots$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$y_0$$

$$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$$

$$\overline{y}_1 \vee y_0$$

$$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$$

$$y_1 \vee \overline{y}_0 \vee \overline{x}_1$$

$$y_1 \vee \overline{y}_0 \vee \sigma(x_1)$$

Derivable by redundance with witness
$\omega : y_1 \mapsto 1$
(same argument)

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$

$y_1 \vee \overline{y}_0 \vee \overline{x}_1$

$y_1 \vee \overline{y}_0 \vee \sigma(x_1)$

$\overline{y}_1 \vee \overline{x}_2 \vee \sigma(x_2)$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$

$y_1 \vee \overline{y}_0 \vee \overline{x}_1$

$y_1 \vee \overline{y}_0 \vee \sigma(x_1)$

$\overline{y}_1 \vee \overline{x}_2 \vee \sigma(x_2)$

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$+ 2^{n-1} \cdot \left(\overline{y}_1 + \overline{\sigma(x_1)} + x_1 \geq 1\right)$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

y_0

$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$

$\overline{y}_1 \vee y_0$

$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$

$y_1 \vee \overline{y}_0 \vee \overline{x}_1$

$y_1 \vee \overline{y}_0 \vee \sigma(x_1)$

$\overline{y}_1 \vee \overline{x}_2 \vee \sigma(x_2)$

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$+ 2^{n-1} \cdot \left(\overline{y}_1 + \overline{\sigma(x_1)} + x_1 \geq 1\right)$$

$$2^{n-1} \cdot \overline{y}_1 + \sum_{i=2}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

Detailed Derivation of CNF Breaking Constraints

Derived constraints (D):

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$y_0$$

$$\overline{y}_0 \vee \overline{x}_1 \vee \sigma(x_1)$$

$$\overline{y}_1 \vee y_0$$

$$\overline{y}_1 \vee \overline{\sigma(x_1)} \vee x_1$$

$$y_1 \vee \overline{y}_0 \vee \overline{x}_1$$

$$y_1 \vee \overline{y}_0 \vee \sigma(x_1)$$

$$\overline{y}_1 \vee \overline{x}_2 \vee \sigma(x_2)$$

$$\sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

$$+ 2^{n-1} \cdot \left(\overline{y}_1 + \overline{\sigma(x_1)} + x_1 \geq 1\right)$$

$$2^{n-1} \cdot \overline{y}_1 + \sum_{i=2}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

The clause to derive is RUP with respect to this constraint