Combinatorial Solving and Optimisation

- Revolution last couple of decades in **combinatorial solvers** for
  - Boolean satisfiability (SAT) solving [BHvMW21]¹
  - Constraint programming (CP) [RvBW06]
  - Mixed integer linear programming (MIP) [AW13, BR07]

- Solve NP-complete problems (or worse) very successfully in practice!

- **Except solvers are sometimes wrong…** (Even best commercial ones) [BLB10, CKSW13, AGJ+18, GSD19, GS19, BMN22, BBN+23]

- Even get feasibility of solutions wrong (though this should be straightforward!)

- And how to check the absence of solutions?

- Or that a solution is optimal? (Even off-by-one mistakes can snowball into large errors if solver used as subroutine)

¹See end of slides for all references with bibliographic details
What Can Be Done About Solver Bugs?

- **Software testing**
  
  Hard to get good test coverage for sophisticated solvers
  
  Inherently can only detect presence of bugs, not absence
What Can Be Done About Solver Bugs?

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

- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to this level of complexity
What Can Be Done About Solver Bugs?

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

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

- **Proof logging**
  Make solver certifying [ABM^11, MMNS11] by outputting
  1. not only answer but also
  2. simple, machine-verifiable proof that answer is correct
Proof Logging with Certifying Solvers: Workflow

1. Run combinatorial solving algorithm on problem input
Ensembling Correctness with the Help of Proof Logging

Proof Logging with Certifying Solvers: Workflow

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
Proof Logging with Certifying Solvers: Workflow

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
Proof Logging with Certifying Solvers: Workflow

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

Proof format for certifying solver should be:

- 
  
  Minimal overhead for sophisticated reasoning
  
  Dead simple: checking correctness of proofs should be trivial
  
  Clear conflict expressivity vs. simplicity!
  
  Asking for both perhaps a little too good to be true?
Introduction

Proof Logging for SAT

Pseudo-Boolean Proof Logging

Advanced SAT Techniques and Optimisation

Subgraph Algorithms

Ensuring Correctness with the Help of Proof Logging

Proof Logging Desiderata

Proof format for certifying solver should be

- **very powerful**: minimal overhead for sophisticated reasoning
Proof Logging Desiderata

Proof format for certifying solver should be

- **very powerful**: minimal overhead for sophisticated reasoning
- **dead simple**: checking correctness of proofs should be trivial
Ensuring Correctness with the Help of Proof Logging

Proof Logging Desiderata

Proof format for certifying solver should be

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

Clear conflict expressivity vs. simplicity!
Proof Logging Desiderata

Proof format for certifying solver should be

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

Clear conflict expressivity vs. simplicity!

Asking for both perhaps a little bit too good to be true?
Take-Away Message from This Tutorial

Proof logging for combinatorial optimisation is possible with single, unified method!
Take-Away Message from This Tutorial

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

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

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

Explain how to use **VERIPB** to do proof logging for

- SAT solving (including advanced techniques)
- SAT-based optimisation (MaxSAT)
- Subgraph algorithms
- Constraint programming
- Symmetry and dominance reasoning

in a unified way
The SAT Problem

- **Variable** \( x \): takes value **true** \((= 1)\) or **false** \((= 0)\)
- **Literal** \( \ell \): 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

Given a CNF formula \( F \), is it satisfiable?

For instance, what about:

\[
(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land \\
(x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})
\]
Proofs for SAT

For satisfiable instances: just specify satisfying assignment

For unsatisfiability: a sequence of clauses (CNF constraints)

- Each clause follows “obviously” from everything we know so far
- Final clause is empty, meaning contradiction (written \(\bot\))
- Means original formula must be inconsistent
What Is Obvious? Unit Propagation

**Unit Propagation**

Clause $C$ **unit propagates** $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$
What Is Obvious? Unit Propagation

**Unit Propagation**

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

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

$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$
What Is Obvious? Unit Propagation

Unit Propagation

Clause **C unit propagates** \( \ell \) under partial assignment \( \rho \) if \( \rho \) falsifies all literals in \( C \) except \( \ell \)

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

\[
(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor z) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})
\]
What Is Obvious? Unit Propagation

**Unit Propagation**

Clause $C$ **unit propagates** $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

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

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

- $p \lor \bar{u}$ propagates $u \mapsto 0$
What Is Obvious? Unit Propagation

**Unit Propagation**

Clause $C$ **unit propagates** $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

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

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

- $p \lor \bar{u}$ propagates $u \mapsto 0$
- $q \lor r$ propagates $r \mapsto 1$
What Is Obvious? Unit Propagation

**Unit Propagation**

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$.

**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 $w \mapsto 1$
What Is Obvious? Unit Propagation

**Unit Propagation**

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

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

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

- $p \lor \bar{u}$ propagates $u \mapsto 0$
- $q \lor r$ propagates $r \mapsto 1$
- Then $\bar{r} \lor w$ propagates $w \mapsto 1$
- No further unit propagations
What Is Obvious? Unit Propagation

**Unit Propagation**

Clause $C$ unit propagates $\ell$ under partial assignment $\rho$ if $\rho$ falsifies all literals in $C$ except $\ell$

**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

\[(p \lor \overline{u}) \land (q \lor r) \land (\overline{F} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{p} \lor \overline{u})\]
Davis-Putman-Logemann-Loveland (DPLL)

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

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

\[(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})\]
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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

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

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

\[(p \lor \neg u) \land (q \lor r) \land (\neg s \lor w) \land (u \lor x \lor y) \land (x \lor \neg y \lor z) \land (\neg x \lor z) \land (\neg y \lor \neg z) \land (\neg x \lor \neg z) \land (\neg p \lor \neg u)\]

1 \[x \lor y\]

\[x\]

\[y\]

\[0\] \[0\] \[1\]
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

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

1. \(x \lor y\)
2. \(x \lor \overline{y}\)
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

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

\[\begin{align*}
1 & x \lor y \\
2 & x \lor \overline{y} \\
3 & x
\end{align*}\]
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

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

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

1. $x \lor y$
2. $x \lor \overline{y}$
3. $x$
4. $\overline{x}$
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

1 \( x \lor y \)
2 \( x \lor \overline{y} \)
3 \( x \)
4 \( \overline{x} \)
5 \( \perp \)
Reverse Unit Propagation (RUP)

To make this a proof, need backtrack clauses to be easily verifiable
Reverse Unit Propagation (RUP)

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

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

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

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

If so, F clearly implies C, and this condition is easy to verify efficiently
Reverse Unit Propagation (RUP)

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

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

$C$ is a reverse unit propagation (RUP) clause with respect to $F$ if
- assigning $C$ to false
- then unit propagating on $F$ until saturation
- leads to contradiction

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

Fact

Backtrack clauses from DPLL solver generate a RUP proof
What About Conflict-Driven Clause Learning (CDCL)?

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

\[(p \lor \overline{u}) \land (q \lor r) \land (\overline{f} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})\]
What About Conflict-Driven Clause Learning (CDCL)?

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

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

**Decision**
Free choice to assign value to variable

**Notation** \( p \overset{d}{=} 0 \)
**What About Conflict-Driven Clause Learning (CDCL)?**

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

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

\[d_p = 0\]

**Decision**

Free choice to assign value to variable

**Notation** \( p \equiv 0 \)
What About Conflict-Driven Clause Learning (CDCL)?

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

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

**Decision**
Free choice to assign value to variable

**Notation** \(p^d = 0\)

**Unit propagation**
Forced choice to avoid falsifying clause
Given \(p = 0\), clause \(p \lor \overline{u}\) forces \(u = 0\)

**Notation** \(u^{p \lor \overline{u}} = 0\) \((p \lor \overline{u}\) is reason clause)
What About Conflict-Driven Clause Learning (CDCL)?

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

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

**Decision**

Free choice to assign value to variable

**Notation** \(p \overset{d}{=} 0\)

**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:

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

**Decision**

Free choice to assign value to variable

**Notation** \( p \overset{d}{=} 0 \)

**Unit propagation**

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict
What About Conflict-Driven Clause Learning (CDCL)?

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

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

**Decision**

Free choice to assign value to variable

Notation \(p = 0\)

**Unit propagation**

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict
What About Conflict-Driven Clause Learning (CDCL)?

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

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

### Decision
Free choice to assign value to variable

**Notation** \( p \models d \)

### Unit propagation
Forced choice to avoid falsifying clause

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

**Notation** \( u \models p \lor \overline{u} \) (\( p \lor \overline{u} \) is reason clause)

Always propagate if possible, otherwise decide
Add to assignment **trail**
Continue until satisfying assignment or **conflict**
What About Conflict-Driven Clause Learning (CDCL)?

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

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

\[
\begin{align*}
\begin{array}{c}
\text{Decision} \\
\text{Free choice to assign value to variable} \\
\text{Notation } p^d = 0
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\text{Unit propagation} \\
\text{Forced choice to avoid falsifying clause} \\
\text{Given } p = 0, \text{ clause } p \lor \overline{u} \text{ forces } u = 0 \\
\text{Notation } u^{p \lor \overline{u}} = 0 (p \lor \overline{u} \text{ is reason clause})
\end{array}
\end{align*}
\]

Always propagate if possible, otherwise decide
Add to assignment trail
Continue until satisfying assignment or conflict
What About Conflict-Driven Clause Learning (CDCL)?

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

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

\[
\begin{align*}
\quad & d = 0 \\
\quad & p = 0 \\
\quad & u = 0 \\
\quad & q = 0 \\
\quad & r = 1 \\
\quad & w = 1 \\
\quad & x = 0 \\
\quad & y = 1
\end{align*}
\]

**Decision**

Free choice to assign value to variable

Notation \( p \overset{d}{=} 0 \)

**Unit propagation**

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict
What About Conflict-Driven Clause Learning (CDCL)?

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

\[(p \lor \bar{u}) \land (q \lor r) \land (\bar{r} \lor w) \land (u \lor x \lor y) \land (x \lor \bar{y} \lor z) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z}) \land (\bar{x} \lor \bar{z}) \land (\bar{p} \lor \bar{u})\]

<table>
<thead>
<tr>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free choice to assign value to variable</td>
</tr>
<tr>
<td>Notation ( p \Downarrow = 0 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unit propagation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forced choice to avoid falsifying clause</td>
</tr>
<tr>
<td>Given ( p = 0 ), clause ( p \lor \bar{u} ) forces ( u = 0 )</td>
</tr>
<tr>
<td>Notation ( u \Uparrow = 0 ) (( p \lor \bar{u} ) is reason clause)</td>
</tr>
</tbody>
</table>

Always propagate if possible, otherwise decide
Add to assignment trail
Continue until satisfying assignment or conflict
What About Conflict-Driven Clause Learning (CDCL)?

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

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

### Decision
Free choice to assign value to variable

**Notation** \( p \overset{d}{=} 0 \)

### Unit propagation
Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide
Add to assignment **trail**
Continue until satisfying assignment or **conflict**
What About Conflict-Driven Clause Learning (CDCL)?

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

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

### Decision
Free choice to assign value to variable

Notation \( p^d = 0 \)

### Unit propagation
Forced choice to avoid falsifying clause

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

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

### Always propagate if possible, otherwise decide
Add to assignment trail
Continue until satisfying assignment or conflict
Conflict Analysis

Time to analyse this conflict and learn from it!

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

\[
\begin{array}{c}
\begin{array}{c}
\text{\(d\)} \\
\text{\(p = 0\)}
\end{array} \\
\begin{array}{c}
\text{\(u \lor \overline{u}\)} \\
\text{\(u = 0\)}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{\(d\)} \\
\text{\(q = 0\)}
\end{array} \\
\begin{array}{c}
\text{\(r = 1\)} \\
\text{\(w = 1\)}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{\(d\)} \\
\text{\(x = 0\)}
\end{array} \\
\begin{array}{c}
\text{\(u \lor x \lor y\)} \\
\text{\(y = 1\)}
\end{array} \\
\begin{array}{c}
\text{\(z = 1\)} \\
\text{\(\overline{y} \lor \overline{z}\)}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\text{\(d\)} \\
\text{\(p \lor \overline{u}\)} \\
\text{\(q \lor r\)} \\
\text{\(r \lor x\)} \\
\text{\(x \lor y\)} \\
\text{\(y \lor z\)} \\
\text{\(z \lor \overline{y}\)} \\
\text{\(\overline{y} \lor \overline{z}\)} \\
\text{\(\overline{z} \lor \overline{y}\)} \\
\text{\(\overline{p} \lor \overline{u}\)}
\end{array}
\]

\[\text{decision level 1}\]
\[\text{decision level 2}\]
\[\text{decision level 3}\]
Conflict Analysis

Time to analyse this conflict and learn from it!

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

\[
\begin{align*}
\text{decision level 1} & \quad \text{decision level 2} & \quad \text{decision level 3} \\
\frac{\text{d} = 0}{\frac{p = 0}{\frac{u \lor \overline{u}}{\text{d} = 0}}} & \quad \frac{q = 0}{\frac{r = 1}{\frac{\overline{r} \lor w}{w = 1}}} & \quad \frac{x = 0}{\frac{u \lor x \lor y}{y = 1} \quad \frac{x \lor \overline{y} \lor z}{z = 1} \quad \frac{\overline{y} \lor \overline{z}}{\bot}}
\end{align*}
\]

Could backtrack by erasing conflict level & flipping last decision.
Conflict Analysis

Time to analyse this conflict and learn from it!

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

\[
\begin{align*}
    &p \at{d}= 0 \\
    &u \at{d}= 0 \\
    &q \at{d}= 0 \\
    &r \at{d}= 1 \\
    &w \at{d}= 1 \\
    &x \at{d}= 0 \\
    &y \at{d}= 1 \\
    &z \at{d}= 1 \\
    &\overline{y} \lor \overline{z} \at{d}= 1
\end{align*}
\]

Could backtrack by erasing conflict level & flipping last decision

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

Time to analyse this conflict and learn from it!

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

Could backtrack by erasing conflict level & flipping last decision

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

Case analysis over \(z\) for last two clauses:

- \(x \lor \overline{y} \lor z\) wants \(z = 1\)
- \(\overline{y} \lor \overline{z}\) wants \(z = 0\)
- **Resolve** clauses by merging them & removing \(z\) — must satisfy \(x \lor \overline{y}\)
Conflict Analysis

Time to analyse this conflict and learn from it!

\[(p \lor \bar{u}) \land (q \lor r) \land (\bar{r} \lor w) \land (u \lor x \lor y) \land (x \lor \bar{y} \lor z) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z}) \land (\bar{x} \lor \bar{z}) \land (\bar{p} \lor \bar{u})\]

Could backtrack by erasing **conflict level** & flipping last decision

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

Case analysis over \(z\) for last two clauses:

- \(x \lor \bar{y} \lor z\) wants \(z = 1\)
- \(\bar{y} \lor \bar{z}\) wants \(z = 0\)
- **Resolve** clauses by merging them & removing \(z\) — must satisfy \(x \lor \bar{y}\)

Repeat until **UIP clause** with only 1 variable at conflict level after last decision — **learn** and **backjump**
Complete Example of CDCL Execution

**Backjump**: undo max #decisions while learned clause propagates

\((p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})\)
Complete Example of CDCL Execution

**Backjump:** undo max #decisions while learned clause propagates

\[(p \lor \bar{u}) \land (q \lor r) \land (\bar{f} \lor w) \land (u \lor x \lor y) \land (x \lor \bar{y} \lor z) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z}) \land (\bar{p} \lor \bar{u})\]

\[
\begin{array}{c}
\begin{array}{c}
\text{d} \\
p = 0
\end{array} \\
\begin{array}{c}
\text{d} \\
u = 0
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{d} \\
p = 0
\end{array} \\
\begin{array}{c}
\text{d} \\
u = 0
\end{array}
\end{array}
\]

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

\[
\begin{array}{c}
\begin{array}{c}
\text{d} \\
x = 0
\end{array} \\
\begin{array}{c}
\text{d} \\
x = 0
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{d} \\
x = 0
\end{array} \\
\begin{array}{c}
\text{d} \\
x = 0
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{d} \\
x = 0
\end{array} \\
\begin{array}{c}
\text{d} \\
x = 0
\end{array}
\end{array}
\]
Complete Example of CDCL Execution

**Backjump:** undo max #decisions while learned clause propagates

\[(p \lor \neg u) \land (q \lor r) \land (\neg f \lor w) \land (u \lor x \lor y) \land (x \lor \neg y \lor z) \land (\neg x \lor z) \land (\neg y \lor \neg z) \land (\neg p \lor \neg u)\]

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

\[(p \lor \bar{u}) \land (q \lor r) \land (\bar{r} \lor w) \land (u \lor x \lor y) \land (x \lor \bar{y} \lor z) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z}) \land (\bar{x} \lor \bar{z}) \land (\bar{p} \lor \bar{u})\]
Complete Example of CDCL Execution

**Backjump**: undo max #decisions while learned clause propagates

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

\[d = 0\]
\[p \lor \overline{u} = 0\]
\[u = 0\]
\[q = 0\]
\[r = 1\]
\[\overline{r} \lor w = 1\]
\[w = 1\]
\[x = 0\]
\[u \lor x = 1\]
\[x \lor \overline{y} = 1\]
\[z \lor \overline{y} \lor z = 1\]
\[\overline{y} \lor z = 1\]
\[\perp\]
Complete Example of CDCL Execution

**Backjump:** undo max #decisions while learned clause propagates

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

\[
\begin{align*}
d & = 0 \\
p & = 0 \\
u & = 0 \\
d & = 0 \\
r & = 1 \\
q & = 0 \\
w & = 1 \\
d & = 0 \\
x & = 1 \\
v & = 1 \\
y & = 1 \\
z & = 1 \\
\overline{y} & = 1 \\
\overline{z} & = 1 \\
\overline{x} & = 1 \\
\overline{p} & = 0 \\
\overline{u} & = 0
\end{align*}
\]
Complete Example of CDCL Execution

**Backjump:** undo max #decisions while learned clause propagates

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

```
\begin{align*}
  & (p \lor \overline{u}) \land (q \lor r) \land (\overline{f} \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}) \\
\end{align*}
```
Complete Example of CDCL Execution

**Backjump:** undo max #decisions while learned clause propagates

\[(p \lor \bar{u}) \land (q \lor r) \land (\bar{f} \lor w) \land (u \lor x \lor y) \land (x \lor \bar{y} \lor z) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z}) \land (\bar{x} \lor \bar{z}) \land (\bar{p} \lor \bar{u})\]

\[d \quad p = 0\]
\[u = 0\]
\[q = 0\]
\[r = 1\]
\[\bar{w} = 1\]
\[x = 0\]
\[y = 1\]
\[\bar{y} \lor \bar{z} = 1\]
\[z = 1\]
\[x \lor \bar{y} = 0\]
\[u \lor x\]
\[\bar{x}\]
\[x = 1\]
\[u = 1\]
Complete Example of CDCL Execution

**Backjump:** undo max #decisions while learned clause propagates

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

\[
\begin{array}{c|c}
 d & p = 0 \\
 u & 0 \\
\end{array}
\]

\[
\begin{array}{c|c}
 d & q = 0 \\
 r & 1 \\
 w & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
 d & u = 0 \\
 x & 0 \\
 y & 1 \\
 z & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
 x & 1 \\
 u & 1 \\
 p & 1 \\
\end{array}
\]

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

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

Then continue as before...
Complete Example of CDCL Execution

**Backjump**: undo max #decisions while learned clause propagates

\[(p \lor \overline{u}) \land (q \lor r) \land (\overline{f} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor u)\]
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

\[(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})\]
Complete Example of CDCL Execution

**Backjump:** undo max #decisions while learned clause propagates

\[(p \lor \lnot u) \land (q \lor r) \land (\lnot f \lor w) \land (u \lor x \lor y) \land (x \lor \lnot y \lor z) \land (\lnot x \land z) \land (\lnot y \lor \lnot z) \land (\lnot x \lor \lnot z) \land (\lnot p \lor \lnot u)\]

**Combinatorial Solving with Provably Correct Results**

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
Complete Example of CDCL Execution

**Backjump**: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \land (q \vee r) \land (\overline{f} \vee w) \land (u \vee x \vee y) \land (x \vee \overline{y} \vee z) \land (\overline{x} \vee z) \land (\overline{y} \vee \overline{z}) \land (\overline{x} \vee \overline{z}) \land (\overline{p} \vee \overline{u})$$
CDCL Reasoning and the Resolution Proof System

To describe CDCL reasoning, need formal proof system for unsatisfiable formulas
CDCL Reasoning and the Resolution Proof System

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

Resolution proof system [Bla37, Rob65]

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

\[ \frac{C \lor x \quad D \lor \overline{x}}{C \lor D} \]

- Done when contradiction \( \perp \) in form of empty clause derived
CDCL Reasoning and the Resolution Proof System

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

Resolution proof system [Bla37, Rob65]
- Start with clauses of formula (axioms)
- Derive new clauses by resolution rule

\[
\begin{array}{c}
C \lor x \\
\hline
\end{array} \quad \begin{array}{c}
D \lor \overline{x} \\
\hline
C \lor D
\end{array}
\]

- Done when contradiction \( \bot \) in form of empty clause derived

When run on unsatisfiable formula, CDCL generates resolution proof*
CDCL Reasoning and the Resolution Proof System

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

Resolution proof system [Bla37, Rob65]

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

\[
C \lor x \quad D \lor \overline{x} \quad \frac{C \lor x}{C \lor D} \quad \frac{D \lor \overline{x}}{}
\]

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

When run on unsatisfiable formula, CDCL generates resolution proof*

(*) Ignores pre- and inprocessing, but we will get there…
Resolution Proofs from CDCL Executions

Obtain resolution proof...
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution…
Resolution Proofs from CDCL Executions

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

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

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses
RUP Proofs and CDCL

But it turns out we can be lazier…

Fact
All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

1. \(u \lor x\)
2. \(\overline{x}\)
3. \(\bot\)
RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

1  \(u \lor x\)

2  \(\bar{x}\)

3  \(\perp\)
RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

1. \(u \lor x\)
2. \(\overline{x}\)
3. \(\bot\)
RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

\[
(p \lor \bar{u}) \land (q \lor r) \land (\bar{r} \lor w) \land (u \lor x \lor y) \land (x \lor \bar{y} \lor z) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z}) \land (\bar{x} \lor \bar{z}) \land (\bar{p} \lor \bar{u})
\]

is sequence of reverse unit propagation (RUP) clauses

1. \( u \lor x \)
2. \( \bar{x} \)
3. \( \perp \)
RUP Proofs and CDCL

But it turns out we can be lazier…

**Fact**

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\bar{x}$
3. $\perp$
RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \lor x$
2. $\overline{x}$
3. $\perp$
RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

\[(p \lor \neg u) \land (q \lor r) \land (\neg \bar{r} \lor w) \land (u \lor x \lor y) \land (x \lor \neg \bar{y} \lor \bar{z}) \land (\neg \bar{x} \lor z) \land (\bar{y} \lor \bar{z}) \land (\neg \bar{x} \lor \bar{z}) \land (\neg \bar{p} \lor \bar{u})\]

is sequence of reverse unit propagation (RUP) clauses

1. \( u \lor x \)
2. \( \neg \bar{x} \)
3. \( \perp \)
RUP Proofs and CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisifiability for

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

is sequence of reverse unit propagation (RUP) clauses

1. $$u \lor x$$
2. $$\bar{x}$$
3. $$\bot$$
RUP Proofs and CDCL

But it turns out we can be lazier…

Fact
All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

1. \(u \lor x\)
2. \(\overline{x}\)
3. \(\bot\)
RUP Proofs and CDCL

But it turns out we can be lazier…

Fact
All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

\[ (p \lor \bar{u}) \land (q \lor r) \land (\bar{r} \lor w) \land (u \lor x \lor y) \land (x \lor \bar{y} \lor z) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z}) \land (\bar{x} \lor \bar{z}) \land (\bar{p} \lor \bar{u}) \]

is sequence of reverse unit propagation (RUP) clauses

1. \( u \lor x \)
2. \( \bar{x} \)
3. \( \bot \)
More Ingredients in Proof Logging for SAT

Fact

RUP proofs can be viewed as shorthand for resolution proofs

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

But RUP and resolution are not enough for preprocessing, inprocessing, and some other kinds of reasoning
Extension Variables, Part 1

Suppose we want a variable $a$ encoding

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

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

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

for fresh variable $a$ (this is fine since $a$ doesn’t appear anywhere previously)
Extension Variables, Part 1

Suppose we want a variable $a$ encoding

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

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

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

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

Fact

Extended resolution (RUP + definition of new variables) is essentially equivalent to the DRAT proof logging system most commonly used for SAT solving.
Why Aren’t We Done?

Practical limitations of current SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently (even for SAT solving)
- Clausal proofs can’t easily reflect what algorithms for other problems do
Why Aren’t We Done?

Practical limitations of current SAT proof logging technology:

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

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

- Enables proof logging for advanced SAT techniques so far beyond reach for efficient DRAT proof logging:
  - Cardinality reasoning
  - Gaussian elimination
  - Symmetry breaking

- Supports use of SAT solvers for optimisation problems (MaxSAT)
- Can justify graph reasoning without knowing what a graph is
- Can justify constraint programming inference without knowing what an integer variable is
Pseudo-Boolean Constraints

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

\[ \sum_{i} a_i \ell_i \geq A \]

- \( a_i, A \in \mathbb{Z} \)
- literals \( \ell_i: x_i \) or \( \bar{x}_i \) (where \( x_i + \bar{x}_i = 1 \))
Pseudo-Boolean Constraints

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

\[ \sum_{i} a_i \ell_i \geq A \]

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

Sometimes convenient to use normalized form [Bar95] with all \( a_i, A \) positive (without loss of generality)
Some Types of Pseudo-Boolean Constraints

1. Clauses

\[ x_1 \lor \overline{x}_2 \lor x_3 \iff x_1 + \overline{x}_2 + x_3 \geq 1 \]

2. Cardinality constraints

\[ x_1 + x_2 + x_3 + x_4 \geq 2 \]

3. General pseudo-Boolean constraints

\[ x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input/model axioms**

From the input
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input/model axioms**

**Literal axioms**

From the input

\[ \ell_i \geq 0 \]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms

Literal axioms

Addition

\[
\ell_i \geq 0
\]

\[
\sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B \\
\sum_i (a_i + b_i) \ell_i \geq A + B
\]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input/model axioms**

**Literal axioms**

**Addition**

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

From the input

\[
\ell_i \geq 0
\]

\[
\sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B
\]

\[
\sum_i (a_i + b_i) \ell_i \geq A + B
\]

\[
\sum_i a_i \ell_i \geq A
\]

\[
\sum_i c a_i \ell_i \geq cA
\]
Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

**Input/model axioms**

From the input

\[ \ell_i \geq 0 \]

**Literal axioms**

\[ \sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B \]

\[ \sum_i (a_i + b_i) \ell_i \geq A + B \]

**Addition**

**Multiplication** for any \( c \in \mathbb{N}^+ \)

\[ \sum_i a_i \ell_i \geq A \]

\[ \sum_i c a_i \ell_i \geq cA \]

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

\[ \sum_i a_i \ell_i \geq A \]

\[ \sum_i \left[ \frac{a_i}{c} \right] \ell_i \geq \left[ \frac{A}{c} \right] \]
Cutting Planes Toy Example

\[ w + 2x + y \geq 2 \]
Cutting Planes Toy Example

Multiply by 2

\[
\begin{align*}
\frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}
\end{align*}
\]
Cutting Planes Toy Example

\[
\frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{2w + 4x + 2y \geq 4}
\]
Cutting Planes Toy Example

Multiply by 2

\[
2w + 4x + 2y \geq 4
\]

Add

\[
w + 2x + y \geq 2 \\
3w + 6x + 2y + 2z \geq 9
\]
Cutting Planes Toy Example

Multiply by 2

\[ \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \]

Add

\[ \frac{3w + 6x + 6y + 2z \geq 9}{w + 2x + 4y + 2z \geq 5} \]
**Cutting Planes Toy Example**

Multiply by 2

\[
\frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4}
\]

Add

\[
\frac{2w + 4x + 2y \geq 4}{3w + 6x + 6y + 2z \geq 9}
\]

Multiply by 2

\[
\frac{w + 2x + 4y + 2z \geq 5}{2z \geq 0}
\]

Multiply by 2

\[
\frac{z \geq 0}{2z \geq 0}
\]
Cutting Planes Toy Example

Multiply by 2: \[ w + 2x + y \geq 2 \implies 2w + 4x + 2y \geq 4 \]

Add: \[ w + 2x + 4y + 2z \geq 5 \]

Multiply by 2: \[ \overline{z} \geq 0 \implies 2\overline{z} \geq 0 \]

Add: \[ 3w + 6x + 6y + 2z \geq 9 \]

Add: \[ 3w + 6x + 6y + 2z + 2\overline{z} \geq 9 \]
Cutting Planes Toy Example

Multiply by 2

Add

w + 2x + y ≥ 2

2w + 4x + 2y ≥ 4

w + 2x + 4y + 2z ≥ 5

w + 2x + 4y + 2z ≥ 5

3w + 6x + 6y + 2 ≥ 9

3w + 6x + 6y + 2 ≥ 9

Multiply by 2

Add

$\bar{z} ≥ 0$

$2\bar{z} ≥ 0$

Multiply by 2

Add

$\bar{z} ≥ 0$

$2\bar{z} ≥ 0$

$\bar{z} ≥ 0$

$2\bar{z} ≥ 0$

$\bar{z} ≥ 0$

$2\bar{z} ≥ 0$
Cutting Planes Toy Example

Multiply by 2

Add

Multiply by 2

Add

$w + 2x + y \geq 2$

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

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

$\bar{z} \geq 0$

$3w + 6x + 6y \geq 9$

$\geq 7$

$2\bar{z} \geq 0$
Cutting Planes Toy Example

Multiply by 2

\[ w + 2x + y \geq 2 \]

\[ 2w + 4x + 2y \geq 4 \]

\[ w + 2x + 4y + 2z \geq 5 \]

\[ \bar{z} \geq 0 \]

Multiply by 2

Add

\[ \frac{3w + 6x + 6y + 2z}{2} \geq 9 \]

Add

\[ \frac{3w + 6x + 6y}{3} \geq 7 \]

Divide by 3

\[ w + 2x + 2y \geq 2 \frac{1}{3} \]
Cutting Planes Toy Example

Multiply by 2: \( w + 2x + y \geq 2 \)

Add: \( 2w + 4x + 2y \geq 4 \)

Multiply by 2: \( w + 2x + 4y + 2z \geq 5 \)

Add: \( 3w + 6x + 6y + 2z \geq 9 \)

Divide by 3: \( 3w + 6x + 6y \geq 7 \)

Add: \( w + 2x + 2y \geq 3 \)
Cutting Planes Toy Example

Multiply by 2

\[
\begin{align*}
2w + 4x + 2y & \geq 4 \\
2w + 4x + 2y & \geq 4 \\
3w + 6x + 6y + 2z & \geq 9 \\
3w + 6x + 6y & \geq 7 \\
w + 2x + 2y & \geq 3
\end{align*}
\]

Add

\[
\begin{align*}
w + 2x + y & \geq 2 \\
w + 2x + 4y + 2z & \geq 5 \\
\bar{z} & \geq 0 \\
2\bar{z} & \geq 0 \\
\bar{z} & \geq 0
\end{align*}
\]

Multiply by 2

Add

Divide by 3

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

Constraint 1 \(\equiv\) \(2x + y + w \geq 2\)

Constraint 2 \(\equiv\) \(2x + 4y + 2z + w \geq 5\)

\(\sim z \equiv \bar{z} \geq 0\)
Cutting Planes Toy Example

Multiply by 2

Add

Multiply by 2

Add

Divide by 3

\[ \begin{align*}
\frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} & \quad \frac{w + 2x + 4y + 2z \geq 5}{\overline{z} \geq 0} \\
\frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y \geq 7} & \quad \frac{w + 2x + 2y \geq 3}{2\overline{z} \geq 0}
\end{align*} \]

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

Constraint 1  \( \equiv 2x + y + w \geq 2 \)
Constraint 2  \( \equiv 2x + 4y + 2z + w \geq 5 \)
\( \overline{z} \equiv \overline{\overline{z}} \geq 0 \)

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

\[ \text{pol 1 2 } \ast \ 2 \ + \ \overline{z} \ 2 \ \ast \ + \ 3 \ d \]
Resolution and Cutting Planes

To simulate resolution step such as

\[
\begin{align*}
\overline{y} \lor z & \quad x \lor \overline{y} \lor z \\
\hline
x \lor \overline{y}
\end{align*}
\]

we can perform the cutting planes steps

\[
\begin{align*}
\overline{y} + \overline{z} & \geq 1 & x + \overline{y} + z & \geq 1 \\
\hline
\text{Add} & \quad x + 2\overline{y} & \geq 1 \\
\text{Divide by 2} & \quad x + \overline{y} & \geq 1
\end{align*}
\]
Resolution and Cutting Planes

To simulate resolution step such as

\[
\frac{\bar{y} \lor \bar{z} \quad x \lor \bar{y} \lor z}{x \lor \bar{y}}
\]

we can perform the cutting planes steps

\[
\frac{\bar{y} + \bar{z} \geq 1 \quad x + \bar{y} + z \geq 1}{\text{Add}}
\]

\[
\frac{x + 2\bar{y} \geq 1}{\text{Divide by 2}}
\]

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

Constraint 7 \(\bar{y} + \bar{z} \geq 1\)

Constraint 5 \(x + \bar{y} + z \geq 1\)

we can write this in the proof log as

\[\text{pol 7 5 + 2 d}\]
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses

\[
\begin{align*}
p^d &= 0 \\
u^u &= 0 \\
q^d &= 0 \\
r^v &= 1 \\
w^w &= 1 \\
x^d &= 0 \\
y^v &= 1 \\
z^z &= 1 \\
\overline{y}^\overline{v} &= 1
\end{align*}
\]
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses

\[
\begin{align*}
\forall x : (p_d = 0) \\
\forall u : (u \lor \neg u = 0) \\
\forall q : (q_d = 0) \\
\forall r : (r \lor \neg r = 1) \\
\forall w : (w \lor \neg w = 1) \\
\forall x : (x_d = 0) \\
\forall y : (u \lor x \lor y) \\
\forall z : (x \lor \neg y \lor z) \\
\forall \bar{y} : (\bar{y} \lor \bar{z}) \\
\forall \bar{x} : (u \lor x \lor \bar{x}) \\
\forall \bar{x} : (\bar{x} \lor \bar{z}) \\
\forall x : (\bar{x} \lor \bar{z}) \\
\forall u : (\bar{u} \lor \bar{\bar{x}}) \\
\forall \bar{u} : (\bar{\bar{x}} \lor \bar{u}) \\
\forall \neg p \lor \bar{u} \\
\forall p \lor \bar{u} \\
\forall \neg x : (\bar{x} \lor \neg \bar{x}) \\
\forall x : (\bar{x} \lor \bar{x}) \\
\forall \neg u : (\bar{u} \lor \neg \bar{u}) \\
\forall u : (\bar{u} \lor \bar{u}) \\
\forall \neg p \lor x : (\neg p \lor \bar{x}) \\
\forall p \lor \bar{u} : (p \lor \bar{u}) \\
\forall \neg x : (\neg x \lor \neg \bar{x}) \\
\forall x : (\bar{x} \lor \bar{x}) \\
\forall \neg u : (\neg u \lor \neg \bar{u}) \\
\forall u : (\neg \bar{u} \lor \neg u) \\
\end{align*}
\]
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses

\[
\begin{align*}
(p \lor u)^1 & \land (q \lor r)^2 \land (\overline{r} \lor w)^3 \land (u \lor x \lor y)^4 \land \\
(x \lor \overline{y} \lor z)^5 & \land (\overline{x} \lor z)^6 \land (\overline{y} \lor \overline{z})^7 \land (\overline{x} \lor \overline{z})^8 \land (\overline{p} \lor \overline{u})^9
\end{align*}
\]
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses

\[(p \lor \bar{u}) \land (q \lor r) \land (\bar{r} \lor w) \land (u \lor x \lor y) \land (x \lor \bar{y} \lor z) \land (\bar{x} \lor z) \land (\bar{y} \lor \bar{z}) \land (\bar{x} \lor \bar{z}) \land (p \lor \bar{u})\]

\[
\begin{align*}
\text{pol } 7 & \ 5 + 2 \ d \ 4 + 2 \ d & \implies \text{Constraint } 10 & \doteq u + x \geq 1 \\
\text{pol } 8 & \ 6 + 2 \ d & \implies \text{Constraint } 11 & \doteq \bar{x} \geq 1 \\
\text{pol } 9 & \ 1 + 2 \ d \ 10 + 2 \ d \ 11 + 2 \ d & \implies \text{Constraint } 12 & \doteq 0 \geq 1
\end{align*}
\]
RUP Revisited

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

Constraint $C$ propagates variable $x$ if setting $x$ to “wrong value” would make $C$ unsatisfiable
RUP Revisited

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

Constraint $C$ propagates variable $x$ if setting $x$ to “wrong value” would make $C$ unsatisfiable.

Risk for confusion:
- Constraint programming people might call this (reverse) integer bounds consistency.
  - Does the same thing if we’re working with clauses.
  - More interesting for general pseudo-Boolean constraints.
- SAT people beware: constraints can propagate multiple times and multiple variables.
Pseudo-Boolean Proof Logging for Example CDCL Execution with RUP

\[(p \lor u) ^1 \land (q \lor r) ^2 \land (\overline{r} \lor w) ^3 \land (u \lor x \lor y) ^4 \land (x \lor \overline{y} \lor z) ^5 \land (\overline{x} \lor z) ^6 \land (\overline{y} \lor \overline{z}) ^7 \land (\overline{x} \lor \overline{z}) ^8 \land (\overline{p} \lor \overline{u}) ^9\]
Pseudo-Boolean Proof Logging for Example CDCL Execution with RUP

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

\begin{align*}
\text{rup} & \ 1 \ u \ 1 \ x \ >= \ 1 \ ; \\
\text{rup} & \ 1 \ \sim x \ >= \ 1 \ ; \\
\text{rup} & \ >= \ 1 \ ; \\
\Rightarrow & \ \text{Constraint 10} \ \triangleq u + x \geq 1 \\
\Rightarrow & \ \text{Constraint 11} \ \triangleq \overline{x} \geq 1 \\
\Rightarrow & \ \text{Constraint 12} \ \triangleq 0 \geq 1
\end{align*}
Extension Variables, Part 2

Suppose we want new, fresh variable $a$ encoding

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

This time, introduce constraints

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

Again, needs support from the proof system
Proof Logs for “Extended Cutting Planes”

For satisfiable instances: just specify a satisfying assignment.

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

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

For satisfiable instances: just specify a satisfying assignment.

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

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

(*) Not actually implemented this way — details to come later…
Deleting Constraints

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

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

So ignored in this tutorial for simplicity and clarity
Enumeration and Optimisation Problems

Enumeration:

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

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

For optimisation:
- Define an objective $f = \sum_i w_i \ell_i$, $w_i \in \mathbb{Z}$, to minimise subject to the contraints in the formula
- To maximise, negate objective
- Log a solution $\alpha$; get an objective-improving constraint $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \alpha(\ell_i)$
- Semantics for proof of optimality: “infeasible to find better solution than best so far”
Pseudo-Boolean Proof Logging — How and Why?

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

■ just do proof logging
Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging

Otherwise

- do trusted or verified translation to 0–1 ILP
- provide proof logging for 0–1 ILP formulation
Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0–1 integer linear program (ILP)
  - just do proof logging

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

Goldilocks compromise between expressivity and simplicity:

1. 0–1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments (even for SAT)
3. Efficient reification of constraints
Pseudo-Boolean Proof Logging — How and Why?

If problem is (special case of) 0–1 integer linear program (ILP)
- just do proof logging

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

**Goldilocks compromise** between expressivity and simplicity:

1. 0–1 ILP **expressive formalism** for combinatorial problems (including objective)
2. **Powerful reasoning** capturing many combinatorial arguments (even for SAT)
3. Efficient **reification** of constraints — example:
   \[ r \Rightarrow x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 \geq 7 \]
   \[ r \Leftarrow x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 \geq 7 \]
Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging

Otherwise

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

**Goldilocks compromise** between expressivity and simplicity:

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

\[
\begin{align*}
    r & \Rightarrow x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7 \\
    r & \Leftarrow x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7
\end{align*}
\]

\[
\begin{align*}
    7r & + x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7 \\
    9r & + \overline{x}_1 + 2x_2 + 3\overline{x}_3 + 4x_4 + 5\overline{x}_5 \geq 9
\end{align*}
\]
The VeriPB Format and Tool

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

Released under MIT Licence

Various features to help development:
- Extended variable name syntax allowing human-readable names
- Proof tracing
- "Trust me" assertions for incremental proof logging

Documentation:
- Specific details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23]
- Lots of concrete example files at https://gitlab.com/MIAOresearch/software/VeriPB
Parity (XOR) Reasoning

Given clauses

\[ x \lor y \lor z \]
\[ x \lor \overline{y} \lor \overline{z} \]
\[ \overline{x} \lor y \lor \overline{z} \]
\[ \overline{x} \lor \overline{y} \lor z \]

and

\[ y \lor z \lor w \]
\[ y \lor \overline{z} \lor \overline{w} \]
\[ y \lor z \lor \overline{w} \]
\[ \overline{y} \lor \overline{z} \lor w \]

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

\[ y \lor z \lor w \]
\[ y \lor \overline{z} \lor \overline{w} \]
\[ y \lor z \lor \overline{w} \]
\[ \overline{y} \lor \overline{z} \lor w \]

want to derive

\[ x \lor \overline{w} \]
\[ \overline{x} \lor w \]
Parity (XOR) Reasoning

Given clauses

\[ x \lor y \lor z \]
\[ x \lor \neg y \lor \neg z \]
\[ \neg x \lor y \lor \neg z \]
\[ \neg x \lor \neg y \lor z \]

and

\[ y \lor z \lor w \]
\[ y \lor \neg z \lor \neg w \]
\[ \neg y \lor z \lor \neg w \]
\[ \neg y \lor \neg z \lor w \]

want to derive

\[ x \lor \neg w \]
\[ \neg x \lor w \]

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

\[ y \lor z \lor w \]
\[ y \lor \overline{z} \lor \overline{w} \]
\[ y \lor z \lor \overline{w} \]
\[ \overline{y} \lor \overline{z} \lor w \]

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

\[ y \lor z \lor w \]
\[ y \lor \overline{z} \lor \overline{w} \]
\[ y \lor z \lor \overline{w} \]
\[ \overline{y} \lor \overline{z} \lor w \]

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

\[ x \lor y \lor z \]
\[ x \lor \bar{y} \lor \bar{z} \]
\[ \bar{x} \lor y \lor \bar{z} \]
\[ \bar{x} \lor \bar{y} \lor z \]

and

\[ y \lor z \lor w \]
\[ y \lor \bar{z} \lor \bar{w} \]
\[ y \lor z \lor \bar{w} \]
\[ \bar{y} \lor \bar{z} \lor w \]

want to derive

\[ x \lor \bar{w} \]
\[ \bar{x} \lor w \]

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 \[\text{[Urq87]}\]
But used in CryptoMiniSat \[\text{[Cry]}\]

DRAT proof logging like \[\text{[PR16]}\] 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

\[ y \lor z \lor w \]
\[ y \lor \overline{z} \lor \overline{w} \]
\[ \overline{y} \lor \overline{z} \lor w \]

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

\[ y \lor z \lor w \]
\[ y \lor \overline{z} \lor \overline{w} \]
\[ \overline{y} \lor z \lor w \]
\[ \overline{y} \lor \overline{z} \lor w \]

want to derive

\[ x \lor \overline{w} \]
\[ \overline{x} \lor w \]

Introduce extension variables \(a, b\) and derive

\[ x + y + z + 2a = 3 \]
\[ y + z + w + 2b = 3 \]

(“=” syntactic sugar for “≥” plus “≤”)
Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

\[ x \lor y \lor z \]
\[ x \lor \overline{y} \lor \overline{z} \]
\[ \overline{x} \lor y \lor \overline{z} \]
\[ \overline{x} \lor \overline{y} \lor z \]

and

\[ y \lor z \lor w \]
\[ y \lor \overline{z} \lor \overline{w} \]
\[ y \lor z \lor \overline{w} \]
\[ \overline{y} \lor \overline{z} \lor w \]

want to derive

\[ x \lor \overline{w} \]
\[ \overline{x} \lor w \]

Introduce extension variables \( a, b \) and derive

\[ x + y + z + 2a = 3 \]
\[ y + z + w + 2b = 3 \]

(“=” syntactic sugar for “\( \geq \)” plus “\( \leq \)"

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

\[ y \lor z \lor w \]
\[ y \lor \overline{z} \lor \overline{w} \]
\[ y \lor z \lor \overline{w} \]
\[ \overline{y} \lor \overline{z} \lor w \]

want to derive

\[ x \lor \overline{w} \]
\[ \overline{x} \lor w \]

Introduce extension variables \( a, b \) and derive

\[ x + y + z + 2a = 3 \]
\[ y + z + w + 2b = 3 \]

(“=” syntactic sugar for “\( \geq \)” plus “\( \leq \)"

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

\[ \begin{align*}
  x \lor y \lor z \\
  x \lor \overline{y} \lor \overline{z} \\
  \overline{x} \lor y \lor \overline{z} \\
  \overline{x} \lor \overline{y} \lor z \\
  \end{align*} \]

and

\[ \begin{align*}
  y \lor z \lor w \\
  y \lor \overline{z} \lor \overline{w} \\
  y \lor \overline{z} \lor w \\
  \overline{y} \lor \overline{z} \lor w \\
  \end{align*} \]

want to derive

\[ \begin{align*}
  x \lor \overline{w} \\
  \overline{x} \lor w \\
  \end{align*} \]

Introduce extension variables \( a, b \) and derive

\[ \begin{align*}
  x + y + z + 2a &= 3 \\
  y + z + w + 2b &= 3 \\
  \end{align*} \]

(“=” syntactic sugar for “\( \geq \)” plus “\( \leq \)”)

Add to get

\[ x + w + 2y + 2z + 2a + 2b = 6 \]

From this can extract

\[ \begin{align*}
  x + \overline{w} &\geq 1 \\
  \overline{x} + w &\geq 1 \\
  \end{align*} \]

\textsc{VeriPB} can certify \textbf{XOR reasoning} \cite{GN21}
CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

- *MiniSat*+ [ES06]
- *Open-WBO* [MML14]
- *NaPS* [SN15]
CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

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

E.g., encode pseudo-Boolean constraint

\[ x_1 + x_2 + x_3 + x_4 \geq 2 \]

to clauses with extension variables

\[ s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k \]
CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

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

E.g., encode pseudo-Boolean constraint

\[ x_1 + x_2 + x_3 + x_4 \geq 2 \]

to clauses with extension variables

\[ s_{i,k} \leftrightarrow \sum_{j=1}^{i} x_j \geq k \]
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

\[ s_{i,k} \leftrightarrow \sum_{j=1}^{i} x_j \geq k \]

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

\[ s_{i,k} \iff \sum_{j=1}^{i} x_j \geq k \]

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

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

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

\[ s_{i,k} \iff \sum_{j=1}^{i} x_j \geq k \]

\[ k \cdot s_{i,k} + \sum_{j=1}^{i} x_j \geq k \]

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

How to know translation is correct?

\[ \overline{s}_{1,1} \lor x_1 \]

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

\[ \overline{s}_{2,2} \lor s_{1,1} \]

\[ \overline{s}_{2,2} \lor x_2 \]

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

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

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

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

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

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

\[ s_{4,2} \]

VeriPB can certify **pseudo-Boolean-to-CNF rewriting** [GMNO22, VDB22]
Certified Maximum Satisfiability (MaxSAT) Solving

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

\[
\min 2x_1 + x_2 \\
\text{s.t. } x_1 \lor \bar{z} \\
z \lor x_2
\]

MaxSAT solver

Result: optimum 1

Many MaxSAT solvers internally make use of SAT solver.
Certified Maximum Satisfiability (MaxSAT) Solving

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

\[
\begin{align*}
\min & \quad 2x_1 + x_2 \\
\text{s.t.} & \quad x_1 \lor \overline{z} \\
& \quad z \lor x_2
\end{align*}
\]

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

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

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

\[
\begin{align*}
\text{min} & \quad 2x_1 + x_2 \\
\text{s.t.} & \quad x_1 \lor \bar{z} \\
& \quad z \lor x_2
\end{align*}
\]

MaxSAT solver

Result: optimum 1

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

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

**Does not work**
Certified Maximum Satisfiability (MaxSAT) Solving

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

\[
\text{min } 2x_1 + x_2 \\
\text{s.t. } x_1 \lor \overline{z} \\
z \lor x_2
\]

MaxSAT solver

Result: optimum 1

Many MaxSAT solvers internally make use of SAT solver. Idea:
- Find optimal solution (checking that it is a solution is easy)
- Add clauses claiming a better solution exists
  - Requires proof logging — can be done with VERIPB
- Use one extra SAT call to get proof of optimality (with standard SAT proof logging)
  - Causes serious overhead

Does not work
Certified Maximum Satisfiability (MaxSAT) Solving

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

\[
\begin{align*}
\text{min} & \quad 2x_1 + x_2 \\
\text{s.t.} & \quad x_1 \lor \overline{z} \\
& \quad z \lor x_2
\end{align*}
\]

MaxSAT solver

Result: optimum 1

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

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

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

Three main categories:

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

Three main categories:

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

VERIPB-based proof logging available [VDB22, Van23]
MaxSAT Solvers

Three main categories:

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

  VERIPB-based proof logging available [VDB22, Van23]

- Core-guided search
  1. Call SAT solver to find solution under most optimistic assumptions
  2. If impossible, rewrite objective given output of SAT solver
  3. Repeat (first solution is optimal)
MaxSAT Solvers

Three main categories:

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

  VERIPB-based proof logging available [VDB22, Van23]

- **Core-guided search**
  1. Call SAT solver to find solution under most optimistic assumptions
  2. If impossible, rewrite objective given output of SAT solver
  3. Repeat (first solution is optimal)

  VERIPB-based proof logging available [BBN+23]
MaxSAT Solvers

Three main categories:

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

  VERIPB-based proof logging available [VDB22, Van23]

- **Core-guided search**
  1. Call SAT solver to find solution under most optimistic assumptions
  2. If impossible, rewrite objective given output of SAT solver
  3. Repeat (first solution is optimal)

  VERIPB-based proof logging available [BBN+23]

- **Implicit Hitting Set**
  1. Call SAT solver to find solution under most optimistic assumptions
  2. Use hitting set solver (MIP solver) to recompute what most possible optimistic assumptions are
  3. Repeat (first solution is optimal)
MaxSAT Solvers

Three main categories:

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

  VERIPB-based proof logging available [VDB22, Van23]

- **Core-guided search**
  1. Call SAT solver to find solution under most optimistic assumptions
  2. If impossible, rewrite objective given output of SAT solver
  3. Repeat (first solution is optimal)

  VERIPB-based proof logging available [BBN+23]

- **Implicit Hitting Set**
  1. Call SAT solver to find solution under most optimistic assumptions
  2. Use hitting set solver (MIP solver) to recompute what most possible optimistic assumptions are
  3. Repeat (first solution is optimal)

  No proof logging available yet
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

VERIPB proof:

<table>
<thead>
<tr>
<th>Derived</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \overline{x}_1 \lor x_2 )</td>
<td>( x_1 \lor \overline{x}_2 \lor r_1 )</td>
</tr>
<tr>
<td>( x_1 \lor \overline{x}_2 )</td>
<td>( x_1 \lor x_2 \lor r_2 )</td>
</tr>
<tr>
<td>( \overline{x}_2 \lor x_3 )</td>
<td>( x_2 \lor x_4 \lor r_3 )</td>
</tr>
<tr>
<td>( \overline{x}_3 \lor x_4 )</td>
<td>( x_1 \lor \overline{x}_2 \lor r_1 )</td>
</tr>
</tbody>
</table>

Run SAT solver to find model

- SAT
- UNSAT

Encode model improving constraints

Last found solution is optimal
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

VERiPB proof:

- derived justification

\[
\begin{align*}
\overline{x}_1 \lor x_2 \\
x_1 \lor \overline{x}_2 \\
\overline{x}_2 \lor x_3 \\
\overline{x}_3 \lor x_4 \\
\end{align*}
\]
MaxSAT example (LSU search)

Objective: $\min \sum_i r_i$

VERIPB proof:

<table>
<thead>
<tr>
<th>derived</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_2 + r_2 \geq 1$</td>
<td>Reverse Unit Propagation</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\overline{x}_1 \lor x_2 \\
x_1 \lor \overline{x}_2 \\
\overline{x}_2 \lor x_3 \\
\overline{x}_3 \lor x_4 \\
\overline{x}_1 \lor \overline{x}_2 \lor r_1 \\
x_1 \lor x_2 \lor r_2 \\
x_2 \lor x_4 \lor r_3 \\
x_2 \lor r_2
\end{align*}
\]
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

\( \text{\textsc{VerIPB}} \) proof:

<table>
<thead>
<tr>
<th>derived</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 + r_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\overline{x}_1 \lor x_2 & \quad \overline{x}_1 \lor \overline{x}_2 \lor r_1 \\
 x_1 \lor \overline{x}_2 & \quad x_1 \lor x_2 \lor r_1 \\
 \overline{x}_2 \lor x_3 & \quad x_2 \lor x_4 \lor r_3 \\
 \overline{x}_3 \lor x_4 & \quad x_2 \lor r_2
\end{align*}
\]

Run SAT solver to find model

Encode model improving constraints

Last found solution is optimal

Combinatorial Solving with Provably Correct Results

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

\text{\textsc{VeriPB} proof:}

<table>
<thead>
<tr>
<th>derived</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 + r_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( {\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3 } )</td>
<td>Incumbent solution</td>
</tr>
</tbody>
</table>

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

Run SAT solver to find model

Encode model improving constraints

Last found solution is optimal

Combinatorial Solving with Provably Correct Results
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

\text{VERIPB proof:}

<table>
<thead>
<tr>
<th>derived</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 + r_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( {\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3 } )</td>
<td>Incumbent solution</td>
</tr>
<tr>
<td>( \sum_i r_i \leq 1 )</td>
<td>Objective Improvement Rule</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\overline{x}_1 & \lor x_2 \\
x_1 & \lor \overline{x}_2 \\
\overline{x}_2 & \lor x_3 \\
\overline{x}_3 & \lor x_4
\end{align*} \]

\[ \begin{align*}
\overline{x}_1 & \lor \overline{x}_2 \lor r_1 \\
x_1 & \lor x_2 \lor r_2 \\
x_2 & \lor x_4 \lor r_3
\end{align*} \]

Run SAT solver to find model

SAT \rightarrow Encode model improving constraints

UNSAT \rightarrow Last found solution is optimal
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

VERIPB proof:

| \( x_2 + r_2 \geq 1 \)  | Reverse Unit Propagation |
| \( \{\overline{x}_1, \ldots, \overline{x}_4, r_1, r_2, r_3 \} \) | Incumbent solution |
| \( \sum_i r_i \leq 1 \)  | Objective Improvement Rule |
| PB\((p_1 \iff (\sum_i r_i \geq 1))\) | Fresh variable (RBS) |
| PB\((p_2 \iff (\sum_i r_i \geq 2))\) |

\[
\begin{align*}
\overline{x}_1 & \lor x_2 \\
x_1 & \lor \overline{x}_2 \\
\overline{x}_2 & \lor x_3 \\
\overline{x}_3 & \lor x_4
\end{align*}
\]

\[
\begin{align*}
\overline{x}_1 & \lor x_2 \lor r_1 \\
x_1 & \lor x_2 \lor r_2 \\
x_2 & \lor x_4 \lor r_3 \\
x_2 & \lor r_2
\end{align*}
\]

Run SAT solver to find model

SAT \quad \text{UNSAT}

Encode model improving constraints

Last found solution is optimal

Combinatorial Solving with Provably Correct Results

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

**\text{VERIPB proof:}**

<table>
<thead>
<tr>
<th>derived</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 + r_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( {\overline{x}_1, \ldots, \overline{x}_4, r_1, r_2, r_3} )</td>
<td>Incumbent solution</td>
</tr>
<tr>
<td>( \sum_i r_i \leq 1 )</td>
<td>Objective Improvement Rule</td>
</tr>
<tr>
<td>( j \cdot \overline{p}_j + \sum_i r_i \geq j )</td>
<td>Fresh variable (RBS)</td>
</tr>
<tr>
<td>( (4 - j) \cdot \overline{p}_j + \sum_i \overline{r}_i \geq 4 - j )</td>
<td></td>
</tr>
</tbody>
</table>

Run SAT solver to find model

- \( x_1 \lor \overline{x}_2 \lor r_1 \)
- \( x_1 \lor \overline{x}_2 \lor r_2 \)
- \( x_2 \lor \overline{x}_3 \lor r_3 \)
- \( x_2 \lor \overline{x}_4 \lor r_3 \)

Encode model improving constraints

Last found solution is optimal
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

\text{VERIPB proof:}

\begin{align*}
\text{derived} & & \text{justification} \\
\bar{x}_2 + r_2 & \geq 1 & \text{Reverse Unit Propagation} \\
\{\bar{x}_1, \ldots, \bar{x}_4, r_1, r_2, r_3\} & & \text{Incumbent solution} \\
\sum_i r_i & \leq 1 & \text{Objective Improvement Rule} \\
 j \cdot p_j + \sum_i r_i & \geq j & \text{Fresh variable (RBS)} \\
(4 - j) \cdot p_j + \sum_i \bar{r}_i & \geq 4 - j & \text{Explicit CP derivation}
\end{align*}

\[ \begin{align*}
\bar{x}_1 & \lor x_2 \\
x_1 & \lor \bar{x}_2 \\
\bar{x}_2 & \lor x_3 \\
\bar{x}_3 & \lor x_4 \\
x_1 & \lor \bar{x}_2 \lor r_1 \\
x_1 & \lor x_2 \lor r_2 \\
x_2 & \lor x_4 \lor r_3 \\
x_2 & \lor r_2
\end{align*} \]

Run SAT solver to find model

SAT \rightarrow \text{Encode model improving constraints} \rightarrow \text{Last found solution is optimal}

Combinatorial Solving with Provably Correct Results

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

\text{VERIPB proof:}

\begin{align*}
\text{derived} & & \text{justification} \\
x_2 + r_2 & \geq 1 & \text{Reverse Unit Propagation} \\
\{\overline{x}_1, \ldots, \overline{x}_4, r_1, r_2, r_3\} & & \text{Incumbent solution} \\
\sum_i r_i & \leq 1 & \text{Objective Improvement Rule} \\
j \cdot \overline{p}_j + \sum_i r_i & \geq j & \text{Fresh variable (RBS)} \\
(4 - j) \cdot p_j + \sum_i \overline{r}_i & \geq 4 - j & \text{Explicit CP derivation} \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j)) & & \\
\end{align*}

SAT

UNSAT

Run SAT solver to find model

Encode model improving constraints

Last found solution is optimal
MaxSAT example (LSU search)

Objective: min $\sum_i r_i$

VERIPB proof:

\[
\begin{align*}
x_2 + r_2 &\geq 1 \\
\{\overline{x}_1, \ldots, \overline{x}_4, r_1, r_2, r_3\} &\\
\sum_i r_i &\leq 1 \\
j \cdot \overline{p}_j + \sum_i r_i &\geq j \\
(4 - j) \cdot p_j + \sum_i \overline{r}_i &\geq 4 - j \\
CNF(p_j \iff (\sum_i r_i \geq j)) &\\
\overline{p}_2 &\geq 1
\end{align*}
\]

\[
\begin{align*}
\overline{x}_1 \lor x_2 &\\
x_1 \lor \overline{x}_2 &\\
\overline{x}_2 \lor x_3 &\\
\overline{x}_3 \lor x_4 &\\
CNF(p_j \iff (\sum_i r_i \geq j)) &
\end{align*}
\]

Run SAT solver to find model

SAT

UNSAT

Encode model improving constraints

Last found solution is optimal

Combinatorial Solving with Provably Correct Results

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

\text{VERIPB proof:}

\begin{align*}
\text{derived} & \quad \text{justification} \\
 x_2 + r_2 & \geq 1 & \text{Reverse Unit Propagation} \\
 \{\overline{x}_1, \ldots, \overline{x}_4, r_1, r_2, r_3\} & \quad \text{Incumbent solution} \\
 \sum_i r_i & \leq 1 & \text{Objective Improvement Rule} \\
 j \cdot \overline{p}_j + \sum_i r_i & \geq j & \text{Fresh variable (RBS)} \\
 (4 - j) \cdot \overline{p}_j + \sum_i \overline{r}_i & \geq 4 - j & \text{Explicit CP derivation} \\
 \text{CNF}(p_j \iff (\sum_i r_i \geq j)) & \quad \overline{p}_2 \\
 \overline{p}_2 & \geq 1 & \text{Explicit CP derivation}
\end{align*}

Run SAT solver to find model

\begin{align*}
\overline{x}_1 \lor x_2 & \quad \overline{x}_1 \lor \overline{x}_2 \lor r_1 \\
x_1 \lor \overline{x}_2 & \quad x_1 \lor x_2 \lor r_2 \\
\overline{x}_2 \lor x_3 & \quad x_2 \lor x_4 \lor r_3 \\
\overline{x}_3 \lor x_4 & \quad x_2 \lor r_2 \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j))
\end{align*}

SAT \quad \text{UNSAT}

Encode model improving constraints

Last found solution is optimal
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

**VERIPB** proof:

<table>
<thead>
<tr>
<th>Derived</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 + r_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( { \overline{x}_1, \ldots, \overline{x}_4, r_1, r_2, r_3 } )</td>
<td>Incumbent solution</td>
</tr>
<tr>
<td>( \sum_i r_i \leq 1 )</td>
<td>Objective Improvement Rule</td>
</tr>
<tr>
<td>( j \cdot \overline{p}_j + \sum_i r_i \geq j )</td>
<td>Fresh variable (RBS)</td>
</tr>
<tr>
<td>( (4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j )</td>
<td>Explicit CP derivation</td>
</tr>
<tr>
<td>CNF(( p_j \Leftrightarrow (\sum_i r_i \geq j) ))</td>
<td>Explicit CP derivation</td>
</tr>
<tr>
<td>( \overline{p}_2 \geq 1 )</td>
<td></td>
</tr>
</tbody>
</table>

SAT solver:

1. Run SAT solver to find model
2. Encode model improving constraints
3. Last found solution is optimal

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

CNF(\( p_j \Leftrightarrow (\sum_i r_i \geq j) \))
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

\text{VERIPB proof:}

<table>
<thead>
<tr>
<th>derived</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 + r_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( {\neg x_1, \ldots, \neg x_4, \neg r_1, r_2, r_3} )</td>
<td>Incumbent solution</td>
</tr>
<tr>
<td>( \sum_i r_i \leq 1 )</td>
<td>Objective Improvement Rule</td>
</tr>
<tr>
<td>( j \cdot \neg p_j + \sum_i r_i \geq j )</td>
<td>Fresh variable (RBS)</td>
</tr>
<tr>
<td>( (4 - j) \cdot p_j + \sum_i \neg r_i \geq 4 - j )</td>
<td>Explicit CP derivation</td>
</tr>
<tr>
<td>( \text{CNF}(p_j \iff (\sum_i r_i \geq j)) )</td>
<td>Explicit CP derivation</td>
</tr>
<tr>
<td>( \neg p_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( x_4 \geq 1 )</td>
<td></td>
</tr>
</tbody>
</table>

\( x_1 \lor \bar{x}_2 \lor r_1 \)
\( x_1 \lor \bar{x}_2 \lor r_2 \)
\( x_2 \lor \bar{x}_3 \lor r_3 \)
\( x_2 \lor \bar{x}_4 \lor r_4 \)
\( \text{CNF}(p_j \iff (\sum_i r_i \geq j)) \)
\( \bar{p}_2 \)

Run SAT solver to find model

SAT \rightarrow Encode model improving constraints

UNSAT \rightarrow Last found solution is optimal

Combinatorial Solving with Provably Correct Results

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
MaxSAT example (LSU search)

Objective: \( \text{min } \sum_i r_i \)

\[ \begin{align*}
\text{derived} & \quad \text{justification} \\
\quad x_2 + r_2 \geq 1 & \quad \text{Reverse Unit Propagation} \\
\quad \{\bar{x}_1, \ldots, \bar{x}_4, \bar{r}_1, r_2, r_3\} & \quad \text{Incumbent solution} \\
\quad \sum_i r_i \leq 1 & \quad \text{Objective Improvement Rule} \\
\quad j \cdot \bar{p}_j + \sum_i r_i \geq j & \quad \text{Fresh variable (RBS)} \\
\quad (4-j) \cdot p_j + \sum_i \bar{r}_i \geq 4-j & \\
\quad \text{CNF}(p_j \iff (\sum_i r_i \geq j)) & \\
\quad \bar{p}_2 \geq 1 & \\
\quad x_4 \geq 1 & \\
\quad \{\bar{x}_1, \bar{x}_2, \bar{x}_3, x_4, \bar{r}_1, r_2, \bar{r}_3\} & \quad \text{Reverse Unit Propagation} \\
\quad \text{Incumbent solution} & \\
\end{align*} \]

\[ \begin{align*}
\bar{x}_1 \lor x_2 & \quad \bar{x}_1 \lor \bar{x}_2 \lor r_1 \\
x_1 \lor \bar{x}_2 & \quad x_1 \lor x_2 \lor r_2 \\
\bar{x}_2 \lor x_3 & \quad x_2 \lor x_4 \lor r_3 \\
\bar{x}_3 \lor x_4 & \quad x_2 \lor r_2 \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j)) & \\
\bar{p}_2 & \quad x_4 \\
\end{align*} \]

Run SAT solver to find model

CNF simplification:

- \( \bar{x}_1, \bar{x}_2, \bar{x}_3, x_4 \)
- \( \bar{r}_1, r_2, \bar{r}_3 \)

Encode model improving constraints

Last found solution is optimal

Combinatorial Solving with Provably Correct Results

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

\( \text{VERIPB proof:} \)

\[
\begin{align*}
x_2 + r_2 & \geq 1 \\
\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\} & \text{Reverse Unit Propagation} \\
\sum_i r_i & \leq 1 \quad \text{Incumbent solution} \\
j \cdot \overline{p}_j + \sum_i r_i & \geq j \quad \text{Objective Improvement Rule} \\
(4 - j) \cdot p_j + \sum_i \overline{r}_i & \geq 4 - j \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j)) & \text{Fresh variable (RBS)} \\
\overline{p}_2 & \geq 1 \quad \text{Explicit CP derivation} \\
x_4 & \geq 1 \quad \text{Explicit CP derivation} \\
\{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{r}_1, r_2, \overline{r}_3\} & \text{Reverse Unit Propagation} \\
\sum_i r_i & \leq 0 \quad \text{Incumbent solution} \\
\text{Objective Improvement Rule}
\end{align*}
\]

\( \overline{x}_1 \lor x_2 \lor \overline{x}_2 \lor r_1 \)

\( x_1 \lor \overline{x}_2 \lor \overline{x}_2 \lor r_2 \)

\( \overline{x}_2 \lor x_3 \lor \overline{x}_3 \lor \overline{x}_4 \lor r_3 \)

\( \overline{x}_3 \lor \overline{r}_2 \lor \overline{r}_2 \lor x_4 \)

\( \text{CNF}(p_j \iff (\sum_i r_i \geq j)) \)

\( \overline{x}_1 \lor \overline{x}_2 \lor r_1 \)

\( x_1 \lor \overline{x}_2 \lor \overline{x}_2 \lor r_2 \)

\( \overline{x}_2 \lor x_3 \lor \overline{x}_3 \lor x_4 \lor r_3 \)

\( \overline{x}_3 \lor \overline{x}_4 \lor \overline{r}_2 \lor r_3 \)

\( \text{CNF}(p_j \iff (\sum_i r_i \geq j)) \)

\( \overline{p}_2 \)

\( x_4 \)

Run SAT solver to find model

SAT

UNSAT

Encode model improving constraints

Last found solution is optimal

Combinatorial Solving with Provably Correct Results

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
MaxSAT example (LSU search)

Objective: \(\text{min } \sum_i r_i\)

\text{VERIPB proof:}

\begin{align*}
\text{derived} & \quad \text{justification} \\
x_2 + r_2 & \geq 1 & \text{Reverse Unit Propagation} \\
\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\} & \quad \text{Incumbent solution} \\
\sum_i r_i & \leq 1 & \text{Objective Improvement Rule} \\
j \cdot \overline{p}_j + \sum_i r_i & \geq j & \text{Fresh variable (RBS)} \\
(4 - j) \cdot \overline{p}_j + \sum_i \overline{r}_i & \geq 4 - j & \text{Explicit CP derivation} \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j)) & & \\
\overline{p}_2 & \geq 1 & \text{Explicit CP derivation} \\
x_4 & \geq 1 & \text{Reverse Unit Propagation} \\
\{\overline{x}_1, \overline{x}_2, x_3, x_4, \overline{r}_1, r_2, \overline{r}_3\} & \quad \text{Incumbent solution} \\
\sum_i r_i & \leq 0 & \text{Objective Improvement Rule} \\
\overline{p}_1 & \geq 1 & \text{Explicit CP derivation}
\end{align*}

\[\begin{align*}
\overline{x}_1 \lor x_2 & \quad \overline{x}_1 \lor \overline{x}_2 \lor r_1 \\
x_1 \lor \overline{x}_2 & \quad x_1 \lor x_2 \lor r_2 \\
\overline{x}_2 \lor x_3 & \quad x_2 \lor x_4 \lor r_3 \\
\overline{x}_3 \lor x_4 & \quad x_2 \lor r_2 \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j)) & & \\
\overline{p}_2 & \quad x_4
\end{align*}\]

Run SAT solver to find model

Encode model improving constraints

SAT \quad UNSAT

Last found solution is optimal

Combinatorial Solving with Provably Correct Results

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

**VERIPB proof:**

<table>
<thead>
<tr>
<th>derived</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 + r_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( {\overline{x}_1, \ldots, \overline{x}_4, r_1, r_2, r_3 } )</td>
<td>Incumbent solution</td>
</tr>
<tr>
<td>( \sum_i r_i \leq 1 )</td>
<td>Objective Improvement Rule</td>
</tr>
<tr>
<td>( j \cdot \overline{p}_j + \sum_i r_i \geq j )</td>
<td>Fresh variable (RBS)</td>
</tr>
<tr>
<td>( (4 - j) \cdot p_j + \sum_i \overline{r}_i \geq 4 - j )</td>
<td>Explicit CP derivation</td>
</tr>
<tr>
<td>( \text{CNF}(p_j \iff (\sum_i r_i \geq j)) )</td>
<td>Implicit CP derivation</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\overline{x}_1 & \lor x_2 \\
\overline{x}_1 & \lor \overline{x}_2 \lor r_1 \\
x_1 & \lor \overline{x}_2 \lor r_2 \\
\overline{x}_2 & \lor x_3 \\
x_2 & \lor x_4 \lor r_3 \\
\overline{x}_3 & \lor x_4 \\
x_2 & \lor r_2 \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j)) \\
\overline{p}_2 & \geq 1 \\
x_4 & \geq 1 \\
\{\overline{x}_1, \overline{x}_2, x_3, x_4, \overline{r}_1, r_2, \overline{r}_3 \} & \text{Incumbent solution} \\
\sum_i r_i & \leq 0 \\
\overline{p}_1 & \geq 1 \\
\end{align*} \]

- Run SAT solver to find model
- Encode model improving constraints
- Last found solution is optimal
MaxSAT example (LSU search)

Objective: $\min \sum_i r_i$

\text{VERIPB} proof:

\begin{align*}
\text{derived} & \quad \text{justification} \\
\sum_i r_i & \geq 1 \quad \text{Reverse Unit Propagation} \\
\{\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3\} & \quad \text{Incumbent solution} \\
\sum_i r_i & \leq 1 \quad \text{Objective Improvement Rule} \\
4 - j & \cdot \overline{p}_j + \sum_i r_i \geq j \quad \text{Fresh variable (RBS)} \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j)) & \\
\overline{p}_2 & \geq 1 \quad \text{Explicit CP derivation} \\
x_4 & \geq 1 \quad \text{Explicit CP derivation} \\
\{\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{r}_1, r_2, \overline{r}_3\} & \quad \text{Reverse Unit Propagation} \\
\sum_i r_i & \leq 0 \quad \text{Incumbent solution} \\
\overline{p}_1 & \geq 1 \quad \text{Objective Improvement Rule} \\
0 & \geq 1 \quad \text{Explicit CP derivation} \\
\end{align*}

\begin{align*}
\overline{x}_1 \vee x_2 & \quad \overline{x}_1 \vee \overline{x}_2 \vee r_1 \\
x_1 \vee \overline{x}_2 & \quad x_1 \vee x_2 \vee r_2 \\
\overline{x}_2 \vee x_3 & \quad x_2 \vee x_4 \vee r_3 \\
\overline{x}_3 \vee x_4 & \quad r_2 \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j)) & \\
\overline{p}_2 & \quad x_4 \\
\overline{p}_1 & \quad \bot \\
\end{align*}

Run SAT solver to find model

\[ \text{SAT} \quad \text{UNSAT} \]

Encode model improving constraints

Last found solution is optimal

Combinatorial Solving with Provably Correct Results

Bart Bogaerts, Ciaran McCreesh, Jakob Nordström
MaxSAT example (LSU search)

Objective: \( \min \sum_i r_i \)

**VERIPB proof:**

<table>
<thead>
<tr>
<th>derived</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 + r_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( {\overline{x}_1, \ldots, \overline{x}_4, \overline{r}_1, r_2, r_3 } )</td>
<td>Incumbent solution</td>
</tr>
<tr>
<td>( \sum_i r_i \leq 1 )</td>
<td>Objective Improvement Rule</td>
</tr>
<tr>
<td>( j \cdot \overline{p}_j + \sum_i r_i \geq j )</td>
<td>Fresh variable (RBS)</td>
</tr>
<tr>
<td>( (4 - j) \cdot \overline{p}_j + \sum_i r_i \geq 4 - j )</td>
<td>Explicit CP derivation</td>
</tr>
<tr>
<td>( \text{CNF}(p_j \iff (\sum_i r_i \geq j)) )</td>
<td>Explicit CP derivation</td>
</tr>
<tr>
<td>( \overline{p}_2 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( x_4 \geq 1 )</td>
<td>Incumbent solution</td>
</tr>
<tr>
<td>( {\overline{x}_1, \overline{x}_2, \overline{x}_3, x_4, \overline{r}_1, r_2, r_3 } )</td>
<td>Objective Improvement Rule</td>
</tr>
<tr>
<td>( \sum_i r_i \leq 0 )</td>
<td>Explicit CP derivation</td>
</tr>
<tr>
<td>( \overline{p}_1 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
<tr>
<td>( 0 \geq 1 )</td>
<td>Reverse Unit Propagation</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\overline{x}_1 \lor x_2 & \quad \overline{x}_1 \lor \overline{x}_2 \lor r_1 \\
x_1 \lor \overline{x}_2 & \quad x_1 \lor x_2 \lor r_2 \\
\overline{x}_2 \lor x_3 & \quad x_2 \lor x_4 \lor r_3 \\
\overline{x}_3 \lor x_4 & \quad x_2 \lor r_2 \\
\text{CNF}(p_j \iff (\sum_i r_i \geq j)) \\
\end{align*}
\]
Progress So Far

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

We’ve learned about

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

Coming next, some worked examples from dedicated graph solvers
The Maximum Clique Problem
The Maximum Clique Problem
Maximum Clique Solvers

There are a lot of dedicated solvers for clique problems

But there are issues:

- “State-of-the-art” solvers have been buggy.
- Often undetected: error rate of around 0.1 [MPP19]

Often used inside other solvers

- An off-by-one result can cause much larger errors
A Brief and Incomplete Guide to Clique Solving (1/4)

Recursive maximum clique algorithm:

- Pick a vertex $v$
- Either $v$ is in the clique...
  - Throw away every vertex not adjacent to $v$
  - If vertices remain, recurse
- ...or $v$ is not in the clique
  - Throw $v$ away and pick another vertex
A Brief and Incomplete Guide to Clique Solving (2/4)

Key data structures:

- Growing clique $C$
- Set of potential vertices $P$
  - All the vertices we haven’t thrown away yet
  - Every $v \in P$ is adjacent to every $w \in C$
A Brief and Incomplete Guide to Clique Solving (2/4)

Key data structures:

- Growing clique $C$
- Set of potential vertices $P$
  - All the vertices we haven’t thrown away yet
  - Every $v \in P$ is adjacent to every $w \in C$

Branch and bound:

- Remember the biggest clique $C^*$ found so far
- If $|C| + |P| \leq |C^*|$, no need to keep going
A Brief and Incomplete Guide to Clique Solving (3/4)

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

We can use $|C| + \#\text{colours}(P)$ as a bound, for any colouring.
This brings us to 1997

Many improvements since then
- better bound functions
- clever vertex selection heuristics
- efficient data structures
- local search
- …

But key ideas for proof logging can be explained without worrying about such things
Making a Proof Logging Clique Solver

1. Output a pseudo-Boolean encoding of the problem
   - Clique problems have several standard file formats

2. Make the solver log its search tree
   - Output a small header
   - Output something on every backtrack
   - Output something every time a solution is found
   - Output a small footer

3. Figure out how to log the bound function
A Slightly Different Proof Logging Workflow

1. Run combinatorial solving algorithm on problem input
A Slightly Different Proof Logging Workflow

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
A Slightly Different Proof Logging Workflow

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

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with 0–1 ILP encoding of input
A Slightly Different Proof Logging Workflow

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with 0–1 ILP encoding of input
4. Verify that proof checker says answer is correct
A Pseudo-Boolean Encoding for Clique (in OPB Format)

* #variable= 12 #constraint= 41
min: -1 x1 -1 x2 -1 x3 -1 x4 ... and so on... -1 x11 -1 x12 ;
1 ~x3 1 ~x1 >= 1 ;
1 ~x3 1 ~x2 >= 1 ;
1 ~x4 1 ~x1 >= 1 ;
* ... and a further 38 similar lines for the remaining non-edges
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

Start with a header
Load the 41 problem axioms
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 \sim x12 \ \sim x7 \ >= \ 1 ;
rup 1 \sim x12 \ >= \ 1 ;
rup 1 \sim x11 \ \sim x10 \ >= \ 1 ;
rup 1 \sim x11 \ >= \ 1 ;
soli x1 x2 x5 x8
rup 1 \sim x8 \ \sim x5 \ >= \ 1 ;
rup 1 \sim x8 \ >= \ 1 ;
rup \ >= \ 1 ;
output NONE
conclusion BOUNDS \ -4 \ -4
end pseudo-Boolean proof

Branch accepting 12
Throw away non-adjacent vertices
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

Branch also accepting 7
Throw away non-adjacent vertices
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

Branch also accepting 9
Throw away non-adjacent vertices
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

We branched on 12, 7, 9
Found a new incumbent
Now looking for a ≥ 4 vertex clique
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
dep pseudo-Boolean proof

Backtrack from 12, 7
9 explored already, only 6 feasible
No ≥ 4 vertex clique possible
Effectively this deletes the 7–12 edge
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
\[ \textcolor{red}{rup 1 ~x12 >= 1 ;} \]
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

Backtrack from 12
Only 1, 6 and 9 feasible (1-colourable)
No ≥ 4 vertex clique possible
Effectively this deletes vertex 12
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;

rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

Branch on 11 then 10
Only 1, 3 and 9 feasible (1-colourable)
No ≥ 4 vertex clique possible
Backtrack, deleting the edge
First Attempt at a Proof

pseudo-Boolean proof version 2.0

f 41
soli x7 x9 x12
rup 1 ¬x12 1 ¬x7 >= 1 ;
rup 1 ¬x12 >= 1 ;
rup 1 ¬x11 1 ¬x10 >= 1 ;
rup 1 ¬x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ¬x8 1 ¬x5 >= 1 ;
rup 1 ¬x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

Backtrack from 11
2-colourable, so no ≥ 4 clique
Delete the vertex
First Attempt at a Proof

pseudo-Boolean proof version 2.0
f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

Branch on 8, 5, 1, 2
Find a new incumbent
Now looking for a ≥ 5 vertex clique
First Attempt at a Proof

pseudo-Boolean proof version 2.0

\( f_{41} \)

\( \text{soli } x_7 \ x_9 \ x_{12} \)

\( \text{rup}_1 \ \sim x_{12} \ 1 \ \sim x_7 \ \geq \ 1 \) ;

\( \text{rup}_1 \ \sim x_{12} \ \geq \ 1 \) ;

\( \text{rup}_1 \ \sim x_{11} \ 1 \ \sim x_{10} \ \geq \ 1 \) ;

\( \text{rup}_1 \ \sim x_{11} \ \geq \ 1 \) ;

\( \text{soli} \ x_1 \ x_2 \ x_5 \ x_8 \)

\( \text{rup}_1 \ \sim x_8 \ 1 \ \sim x_5 \ \geq \ 1 \) ;

\( \text{rup}_1 \ \sim x_8 \ \geq \ 1 \) ;

\( \text{rup} \ \geq \ 1 \) ;

output NONE

conclusion BOUNDS -4 -4

end pseudo-Boolean proof

Backtrack from 8, 5

Only 4 vertices; can’t have a \( \geq 5 \) clique

Delete the edge
First Attempt at a Proof

pseudo-Boolean proof version 2.0

f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
\textbf{rup 1 ~x8 >= 1 ;} 
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
def pseudo-Boolean proof

Backtrack from 8
Still not enough vertices
Delete the vertex
First Attempt at a Proof

pseudo-Boolean proof version 2.0

\[ f(41) \]
\[ \text{soli } x_7 \ x_9 \ x_{12} \]
\[ \text{rup } 1 \ \neg x_{12} \ 1 \ \neg x_7 >= 1 ; \]
\[ \text{rup } 1 \ \neg x_{12} >= 1 ; \]
\[ \text{rup } 1 \ \neg x_{11} \ 1 \ \neg x_{10} >= 1 ; \]
\[ \text{rup } 1 \ \neg x_{11} >= 1 ; \]
\[ \text{soli } x_1 \ x_2 \ x_5 \ x_8 \]
\[ \text{rup } 1 \ \neg x_8 \ 1 \ \neg x_5 >= 1 ; \]
\[ \text{rup } 1 \ \neg x_8 >= 1 ; \]
\[ \text{rup} >= 1 ; \]
\[ \text{output} \ \text{NONE} \]
\[ \text{conclusion} \ \text{BOUNDS} \ -4 -4 \]
\[ \text{end} \ \text{pseudo-Boolean proof} \]

Remaining graph is 3-colourable
Backtrack from root node
First Attempt at a Proof

pseudo-Boolean proof version 2.0

f 41
soli x7 x9 x12
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
soli x1 x2 x5 x8
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE
conclusion BOUNDS -4 -4
end pseudo-Boolean proof

Finish with what we’ve concluded
We specify a lower and an upper bound
Remember we’re minimising \( \sum_v -1 \times v \), so a 4-clique has an objective value of \(-4\).
Verifying This Proof (Or Not…) 

$ veripb clique.opb clique-attempt-one.veripb
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
Verifying This Proof (Or Not…)

$\texttt{veripb clique.opb clique-attempt-one.veripb}$

Verification failed.

Failed in proof file line 6.

Hint: Failed to show $'1 \sim x_10 1 \sim x_{11} >= 1'$ by reverse unit propagation.
Verifying This Proof (Or Not...)

$ veripb --trace clique.opb clique-attempt-one.veripb
line 002: f 41
   ConstraintId 001: 1 ~x1 1 ~x3 >= 1
   ConstraintId 002: 1 ~x2 1 ~x3 >= 1
...  
   ConstraintId 041: 1 ~x11 1 ~x12 >= 1
line 003: soli x7 x9 x12 ~x1 ~x2 ~x3 ~x4 ~x5 ~x6 ~x8 ~x10 ~x11
   ConstraintId 042: 1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 >= 4
line 004: rup 1 ~x12 1 ~x7 >= 1 ;
   ConstraintId 043: 1 ~x7 1 ~x12 >= 1
line 005: rup 1 ~x12 >= 1 ;
   ConstraintId 044: 1 ~x12 >= 1
line 006: rup 1 ~x11 1 ~x10 >= 1 ;
Verification failed.
Failed in proof file line 6.
Hint: Failed to show '1 ~x10 1 ~x11 >= 1' by reverse unit propagation.
Dealing With Colourings

The colour bound doesn’t follow by RUP…

But we can lazily recover at-most-one constraints for each colour class!
Dealing With Colourings

The colour bound doesn’t follow by RUP…

But we can lazily recover at-most-one constraints for each colour class!

\[
(\overline{x}_1 + \overline{x}_6 \geq 1) \\
+ (\overline{x}_1 + \overline{x}_9 \geq 1) \\
+ (\overline{x}_6 + \overline{x}_9 \geq 1) \\
/ 2
\]

\[
= 2\overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2
\]

\[
= 2\overline{x}_1 + 2\overline{x}_6 + 2\overline{x}_9 \geq 3
\]

\[
= \overline{x}_1 + \overline{x}_6 + \overline{x}_9 \geq 2
\]

i.e. \( x_1 + x_6 + x_9 \leq 1 \)
Dealing With Colourings

The colour bound doesn’t follow by RUP...

But we can lazily recover at-most-one constraints for each colour class!

\[
\begin{align*}
(x_1 + \overline{x}_6 &\geq 1) \\
+ (x_1 + \overline{x}_9 &\geq 1) \\
+ (\overline{x}_6 + \overline{x}_9 &\geq 1)
\end{align*}
\]

= \frac{2x_1 + x_6 + x_9}{2} \geq 2

= \frac{2x_1 + 2x_6 + 2x_9}{2} \geq 3

= x_1 + x_6 + x_9 \leq 1

This generalises to colour classes of any size \( v \)

- Each non-edge is used exactly once, \( v(v - 1) \) additions
- \( v - 3 \) multiplications and \( v - 2 \) divisions

Solvers don’t need to “understand” cutting planes to write this derivation to proof log
pseudo-Boolean proof version 2.0

\[ f(41) \]

\[ \text{rup} 1 \sim x12 \ 1 \sim x7 \geq 1 ; \]
* bound, colour classes [ x1 x6 x9 ]

\[ \text{pol} \ 7_{1\sim6} \ 19_{1\sim9} + 24_{6\sim9} + 2 \ d \]
\[ \text{pol} \ 42_{\text{obj}} \ -1 + \]

\[ \text{rup} 1 \sim x12 \geq 1 ; \]
* bound, colour classes [ x1 x3 x9 ]

\[ \text{pol} \ 1_{1\sim3} \ 19_{1\sim9} + 21_{3\sim9} + 2 \ d \]
\[ \text{pol} \ 42_{\text{obj}} \ -1 + \]

\[ \text{rup} 1 \sim x11 \ 1 \sim x10 \geq 1 ; \]
* bound, colour classes [ x1 x3 x7 ]
* [ x9 ]

\[ \text{pol} \ 1_{1\sim3} \ 10_{1\sim7} + 12_{3\sim7} + 2 \ d \]
\[ \text{pol} \ 42_{\text{obj}} \ -1 + \]

\[ \text{rup} 1 \sim x11 \geq 1 ; \]

\[ \text{rup} 1 \sim x8 \ 1 \sim x5 \geq 1 ; \]
* bound, colour classes [ x1 x9 ] [ x2 ]

\[ \text{pol} \ 53_{\text{obj}} \ 19_{1\sim9} + \]
\[ \text{rup} 1 \sim x8 \geq 1 ; \]
* bound, colour classes [ x1 x3 x7 ]
* [ x2 x4 x9 ] [ x5 x6 x10 ]

\[ \text{pol} \ 1_{1\sim3} \ 10_{1\sim7} + 12_{3\sim7} + 2 \ d \]
\[ \text{pol} \ 53_{\text{obj}} \ -1 + \]
\[ \text{pol} \ 42_{\text{obj}} \ -3 + -1 + \]
\[ \text{pol} \ 53_{\text{obj}} \ -5 + -3 + -1 + \]

\[ \text{rup} \geq 1 ; \]
output NONE

conclusion BOUNDS \(-4 \ -4\)
end pseudo-Boolean proof
Verifying This Proof (For Real, This Time)

```
$ veripb --trace clique.opb clique-attempt-two.veripb

=== begin trace ===
line 002: f 41
  ConstraintId 001: 1 ~x1 1 ~x3 >= 1
  ConstraintId 002: 1 ~x2 1 ~x3 >= 1
...
  ConstraintId 041: 1 ~x11 1 ~x12 >= 1
line 003: soli x7 x9 x12 ~x1 ~x2 ~x3 ~x4 ~x5 ~x6 ~x8 ~x10 ~x11
  ConstraintId 042: 1 x1 1 x2 1 x3 1 x4 1 x5 1 x6 1 x7 1 x8 1 x9 1 x10 1 x11 1 x12 >= 4
line 004: rup 1 ~x12 1 ~x7 >= 1;
  ConstraintId 043: 1 ~x7 1 ~x12 >= 1
line 005: * bound, colour classes [ x1 x6 x9 ]
line 006: pol 7 19 + 24 + 2 d
  ConstraintId 044: 1 ~x1 1 ~x6 1 ~x9 >= 2
line 007: pol 42 43 +
  ConstraintId 045: 1 x8 1 x11 1 x12 >= 2
...
  ConstraintId 061: 1 ~x5 1 ~x6 1 ~x10 >= 2
line 028: pol 53 57 + 59 + 61 +
  ConstraintId 062: 1 x8 1 x11 1 x12 >= 2
line 029: rup >= 1;
  ConstraintId 063: >= 1
line 030: output NONE
line 031: conclusion BOUNDS -4 -4
line 032: end pseudo-Boolean proof

=== end trace ===
```

Verification succeeded.
Different Clique Algorithms

Different search orders?
- ✓ Irrelevant for proof logging

Using local search to initialise?
- ✓ Just log the incumbent

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

Weighted cliques?
- ✓ Multiply a colour class by its largest weight
- ✓ Also works for vertices “split between colour classes”
Subgraph Isomorphism

- Find the **pattern** inside the **target**
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find **all** matches
Subgraph Isomorphism

- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches
Subgraph Isomorphism

- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches
Subgraph Isomorphism

- Find the **pattern** inside the **target**
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find **all** matches
Subgraph Isomorphism

- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches
Subgraph Isomorphism

- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches
Subgraph Isomorphism

- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, ...
- Often want to find all matches
Subgraph Isomorphism

- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches
Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a target vertex:

$$\sum_{t \in V(T)} x_{p,t} = 1 \quad p \in V(P)$$
Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a target vertex:

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

Each target vertex may be used at most once:

$$\sum_{p \in V(P)} -x_{p,t} \geq -1 \quad t \in V(T)$$
Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a target vertex:

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

Each target vertex may be used at most once:

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

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

$$\overline{x}_{p,t} + \sum_{u \in N(t)} x_{q,u} \geq 1 \quad p \in V(P), \; q \in N(p), \; t \in V(T)$$
Degree Reasoning in Cutting Planes

Pattern vertex $p$ of degree $\text{deg}(p)$ can never be mapped to target vertex $t$ of degree $< \text{deg}(p)$ in any subgraph isomorphism.

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

We wish to derive $\bar{x}_{p,t} \geq 1$
Degree Reasoning in Cutting Planes

Adjacency:
\[
\overline{x}_{p,t} + x_{q,u} + x_{q,v} \geq 1 \\
\overline{x}_{p,t} + x_{r,u} + x_{r,v} \geq 1 \\
\overline{x}_{p,t} + x_{s,u} + x_{s,v} \geq 1 
\]

Injectivity:
\[
-x_{o,u} - x_{p,u} - x_{q,u} - x_{r,u} - x_{s,u} \geq -1 \\
-x_{o,v} - x_{p,v} - x_{q,v} - x_{r,v} - x_{s,v} \geq -1 
\]

Literal axioms:
\[
x_{o,u} \geq 0 \\
x_{o,v} \geq 0 \\
x_{p,u} \geq 0 \\
x_{p,v} \geq 0 
\]

Add these together …
\[
3 \cdot \overline{x}_{p,t} \geq 1 
\]
Degree Reasoning in Cutting Planes

Adjacency:
\[ \overline{x}_{p,t} + x_{q,u} + x_{q,v} \geq 1 \]
\[ \overline{x}_{p,t} + x_{r,u} + x_{r,v} \geq 1 \]
\[ \overline{x}_{p,t} + x_{s,u} + x_{s,v} \geq 1 \]

Injectivity:
\[ -x_{o,u} - x_{p,u} - x_{q,u} - x_{r,u} - x_{s,u} \geq -1 \]
\[ -x_{o,v} - x_{p,v} - x_{q,v} - x_{r,v} - x_{s,v} \geq -1 \]

Literal axioms:
\[ x_{o,u} \geq 0 \]
\[ x_{o,v} \geq 0 \]
\[ x_{p,u} \geq 0 \]
\[ x_{p,v} \geq 0 \]

Add these together and divide by 3 to get
\[ \overline{x}_{p,t} \geq 1 \]
Degree Reasoning in VERiPB

\[
\begin{align*}
\text{pol } 18_{p \sim q} & \quad 19_{p \sim r} + 20_{p \sim s} + \\
& \quad 12_{inj(u)} + 13_{inj(v)} + \\
& \quad xo_u + xo_v + \\
& \quad xp_u + xp_v + \\
3 \; d \\
\end{align*}
\]

* sum adjacency constraints

* sum injectivity constraints

* cancel stray \( xo_* \)

* cancel stray \( xp_* \)

* divide, and we're done

Or we can ask VERiPB to do the last bit of simplification automatically:

\[
\begin{align*}
\text{pol } 18_{p \sim q} & \quad 19_{p \sim r} + 20_{p \sim s} + \\
& \quad 12_{inj(u)} + 13_{inj(v)} + \\
& \quad \text{ia } -1 : 1 \sim xp_t >= 1 ; \\
\end{align*}
\]

* sum adjacency constraints

* sum injectivity constraints

* desired conclusion is implied
Other Forms of Reasoning

We can also log all of the other things state of the art subgraph solvers do:

- Injectivity reasoning and filtering
- Distance filtering
- Neighbourhood degree sequences
- Path filtering
- Supplemental graphs
Other Forms of Reasoning

We can also log all of the other things state of the art subgraph solvers do:

- Injectivity reasoning and filtering
- Distance filtering
- Neighbourhood degree sequences
- Path filtering
- Supplemental graphs

Proof steps are “efficient” using cutting planes

- Length of proof $\approx$ time complexity of the reasoning algorithms
- Most proof steps require only trivial additional computations
Limitations

Why trust the encoding?

- Correctness of encoding can be formally verified! Work in progress…
Limitations

Why trust the encoding?
- Correctness of encoding can be formally verified! Work in progress…

Proof logging can introduce large slowdowns
- Writing to disk is much slower than bit-parallel algorithms
Limitations

Why trust the encoding?

- Correctness of encoding can be formally verified! Work in progress…

Proof logging can introduce large slowdowns

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

Verification can be even slower

- Unit propagation is much slower than bit-parallel algorithms
Limitations

Why trust the encoding?
- Correctness of encoding can be formally verified! Work in progress…

Proof logging can introduce large slowdowns
- Writing to disk is much slower than bit-parallel algorithms

Verification can be even slower
- Unit propagation is much slower than bit-parallel algorithms

Works up to moderately-sized hard instances
- Even an $O(n^3)$ encoding is painful
- Particularly bad when the pseudo-Boolean encoding talks about “non-edges” but large sparse graphs are “easy”
Proof Logging for Subgraph Isomorphism Solvers

Code for Proof Logging Subgraph Solver

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

Released under MIT Licence
Recap (1/2)

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

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with 0–1 ILP encoding of input
Recap (1/2)

1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with 0–1 ILP encoding of input
4. Verify that proof checker says answer is correct
Recap (2/2)

**Proof logging implementation**
- Don’t change solver
- Just add proof logging statements (plus some book-keeping)

**Performance goals**
Want linear(ish) scaling in terms of solver running time for
- proof size
- proof checking time
What About Constraint Programming?

Non-Boolean variables?

Constraints?
  - Encoding constraints in pseudo-Boolean form?
  - Justifying inferences?

Reformulations?
Compiling CP Variables (1/2)

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

\[
\begin{align*}
    a_{-3} + a_{-2} + a_{-1} + a_0 + a_1 + a_2 + a_3 \\
    + a_4 + a_5 + a_6 + a_7 + a_8 + a_9 &= 1
\end{align*}
\]
Compiling CP Variables (1/2)

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

$$a_{-3} + a_{-2} + a_{-1} + a_0 + a_1 + a_2 + a_3$$

$$+ a_4 + a_5 + a_6 + a_7 + a_8 + a_9 = 1$$

This doesn’t work for large domains…
Compiling CP Variables (1/2)

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

\[
a_{-3} + a_{-2} + a_{-1} + a_0 + a_1 + a_2 + a_3 \\
+ a_4 + a_5 + a_6 + a_7 + a_8 + a_9 = 1
\]

This doesn’t work for large domains…

We could use a binary encoding:

\[
-16a_{\text{neg}} + 1a_{b0} + 2a_{b1} + 4a_{b2} + 8a_{b3} \geq -3 \quad \text{and} \\
16a_{\text{neg}} - 1a_{b0} - 2a_{b1} - 4a_{b2} - 8a_{b3} \geq -9
\]

This doesn’t propagate much, but that isn’t a problem for proof logging
Compiling CP Variables (1/2)

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

$$a_{-3} + a_{-2} + a_{-1} + a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 + a_9 = 1$$

This doesn’t work for large domains…

We could use a binary encoding:

$$-16a_{\text{neg}} + 1a_{b0} + 2a_{b1} + 4a_{b2} + 8a_{b3} \geq -3 \quad \text{and}$$

$$16a_{\text{neg}} - 1a_{b0} - 2a_{b1} - 4a_{b2} - 8a_{b3} \geq -9$$

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

Convention in what follows:

- Upper-case $A, B, C$ are CP variables;
- Lower-case $a, b, c$ are corresponding Boolean variables in PB encoding
Compiling CP Variables (2/2)

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

\[
\begin{align*}
a_{\geq 4} &\iff -16a_{\text{neg}} + 1a_{b0} + 2a_{b1} + 4a_{b2} + 8a_{b3} \geq 4 \\
a_{\geq 5} &\iff -16a_{\text{neg}} + 1a_{b0} + 2a_{b1} + 4a_{b2} + 8a_{b3} \geq 5 \\
a_{=}4 &\iff a_{\geq 4} \land \bar{a}_{\geq 5}
\end{align*}
\]
Compiling CP Variables (2/2)

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

\[
\begin{align*}
a_{\geq 4} & \iff -16a_{\text{neg}} + 1a_{b_0} + 2a_{b_1} + 4a_{b_2} + 8a_{b_3} \geq 4 \\
a_{\geq 5} & \iff -16a_{\text{neg}} + 1a_{b_0} + 2a_{b_1} + 4a_{b_2} + 8a_{b_3} \geq 5 \\
a_{= 4} & \iff a_{\geq 4} \land \bar{a}_{\geq 5}
\end{align*}
\]

When creating \(a_{\geq i}\), also introduce pseudo-Boolean constraints encoding

\[
a_{\geq i} \implies a_{\geq j} \quad \text{and} \quad a_{\geq k} \implies a_{\geq i}
\]

for the closest values \(j < i < k\) that already exist.
Compiling CP Variables (2/2)

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

\[ a_{\geq 4} \iff -16a_{\text{neg}} + 1a_{b0} + 2a_{b1} + 4a_{b2} + 8a_{b3} \geq 4 \]
\[ a_{\geq 5} \iff -16a_{\text{neg}} + 1a_{b0} + 2a_{b1} + 4a_{b2} + 8a_{b3} \geq 5 \]
\[ a_{= 4} \iff a_{\geq 4} \land \bar{a}_{\geq 5} \]

When creating \( a_{\geq i} \), also introduce pseudo-Boolean constraints encoding

\[ a_{\geq i} \implies a_{\geq j} \quad \text{and} \quad a_{\geq h} \implies a_{\geq i} \]

for the closest values \( j < i < h \) that already exist

We can do this:

- Inside the pseudo-Boolean model, where needed
- Otherwise lazily during proof logging
Compiling Constraints

- Also need to compile every constraint to pseudo-Boolean form
- Doesn’t need to be a propagating encoding
- Can use additional variables
Compiling Linear Inequalities

Given inequality

\[ 2A + 3B + 4C \geq 42 \]

where \( A, B, C \in \{-3 \ldots 9\} \)
Compiling Linear Inequalities

Given inequality

\[ 2A + 3B + 4C \geq 42 \]

where \( A, B, C \in \{-3 \ldots 9\} \)

Encode in pseudo-Boolean form as

\[
-32a_{\text{neg}} + 2a_{b0} + 4a_{b1} + 8a_{b2} + 16a_{b3} \\
+ - 48b_{\text{neg}} + 3b_{b0} + 6b_{b1} + 12b_{b2} + 24b_{b3} \\
+ - 64c_{\text{neg}} + 4c_{b0} + 8c_{b1} + 16c_{b2} + 32c_{b3} \geq 42
\]
Compiling Table Constraints

Constraints can be specified \textit{extensionally} as list of feasible tuples, called a \textit{table}.
Variable assignments must match some row in table.
Compiling Table Constraints

Constraints can be specified **extensionally** as list of feasible tuples, called a table.
Variable assignments must match some row in table.

Given table constraint

\[(A, B, C) \in [(1, 2, 3), (1, 3, 4), (2, 2, 5)]\]

define

\[3\bar{t}_1 + a_1 + b_2 + c_3 \geq 3\]  \[\text{i.e., } t_1 \Rightarrow (a_1 \land b_2 \land c_3)\]

\[3\bar{t}_2 + a_1 + b_4 + c_4 \geq 3\]  \[\text{i.e., } t_2 \Rightarrow (a_1 \land b_4 \land c_4)\]

\[3\bar{t}_3 + a_2 + b_2 + c_5 \geq 3\]  \[\text{i.e., } t_3 \Rightarrow (a_2 \land b_2 \land c_5)\]

using tuple selector variables

\[t_1 + t_2 + t_3 = 1\]
Encoding Constraint Definitions

Already know how to do it for any constraint with a sane encoding using some combination of

- CNF
- Integer linear inequalities
- Table constraints
- Auxiliary variables

Simplicity is important, propagation strength isn’t
Justifying Search

Mostly this works as in earlier examples

Restarts are easy

No need to justify guesses or decisions — only justify backtracking
Justifying Inference

**Key idea**

Anything the constraint programming solver knows must follow from *unit propagation* of guessed assignments on *constraints in proof log*
Justifying Inference

Key idea

Anything the constraint programming solver knows must follow from unit propagation of guessed assignments on constraints in proof log

If it follows from unit propagation on the encoding, nothing needed

Some propagators and encodings need RUP steps for inferences
  - A lot of propagators are effectively “doing a little bit of lookahead” but in an efficient way
Proof Logging for the CP Solver

Justifying Inference

Key idea

Anything the constraint programming solver knows must follow from unit propagation of guessed assignments on constraints in proof log

If it follows from unit propagation on the encoding, nothing needed

Some propagators and encodings need RUP steps for inferences

- A lot of propagators are effectively “doing a little bit of lookahead” but in an efficient way

A few need explicit cutting planes justifications written to the proof log

- Linear inequalities just need to multiply and add
- All-different needs a bit more
Justifying All-Different Failures

\[ V \in \{ 1, 4, 5 \} \]
\[ W \in \{ 1, 2, 3 \} \]
\[ X \in \{ 2, 3 \} \]
\[ Y \in \{ 1, 3 \} \]
\[ Z \in \{ 1, 3 \} \]
Justifying All-Different Failures

\[ V \in \{1, 4, 5\} \]
\[ W \in \{1, 2, 3\} \]
\[ X \in \{2, 3\} \]
\[ Y \in \{1, 3\} \]
\[ Z \in \{1, 3\} \]
Justifying All-Different Failures

\[
V \in \{1, 4, 5\} \\
W \in \{1, 2, 3\} \quad w_1 + w_2 + w_3 \geq 1 \quad [\text{\textit{W} takes some value}] \\
X \in \{2, 3\} \\
Y \in \{1, 3\} \\
Z \in \{1, 3\}
\]
Justifying All-Different Failures

\[ V \in \{1, 4, 5\} \]
\[ W \in \{1, 2, 3\} \quad \begin{array}{c} w_1 + w_2 + w_3 \\ \geq 1 \end{array} [\text{\textit{W takes some value}}] \]
\[ X \in \{2, 3\} \quad \begin{array}{c} x_2 + x_3 \\ \geq 1 \end{array} [\text{\textit{X takes some value}}] \]
\[ Y \in \{1, 3\} \quad \begin{array}{c} y_1 + y_3 \\ \geq 1 \end{array} [\text{\textit{Y takes some value}}] \]
\[ Z \in \{1, 3\} \quad \begin{array}{c} z_1 + z_3 \\ \geq 1 \end{array} [\text{\textit{Z takes some value}}] \]
Justifying All-Different Failures

\[
V \in \{1, 4, 5\} \\
W \in \{1, 2, 3\} \quad \{w_1 + w_2 + w_3 \geq 1 \quad [W \text{ takes some value}]\} \\
X \in \{2, 3\} \quad \{x_2 + x_3 \geq 1 \quad [X \text{ takes some value}]\} \\
Y \in \{1, 3\} \quad \{y_1 + y_3 \geq 1 \quad [Y \text{ takes some value}]\} \\
Z \in \{1, 3\} \quad \{z_1 + z_3 \geq 1 \quad [Z \text{ takes some value}]\}
\]

\[
\rightarrow -v_1 -w_1 + -y_1 -z_1 \geq -1 \quad [\text{At most one variable} = 1]\n\rightarrow -w_2 -x_2 \geq -1 \quad [\text{At most one variable} = 2]\n\rightarrow -w_3 -x_3 -y_3 -z_3 \geq -1 \quad [\text{At most one variable} = 3]
\]
Justifying All-Different Failures

\( V \in \{1, 4, 5\} \)
\( W \in \{1, 2, 3\} \quad w_1 + w_2 + w_3 \geq 1 \quad [W \text{ takes some value}] \)
\( X \in \{2, 3\} \quad x_2 + x_3 \geq 1 \quad [X \text{ takes some value}] \)
\( Y \in \{1, 3\} \quad y_1 + y_3 \geq 1 \quad [Y \text{ takes some value}] \)
\( Z \in \{1, 3\} \quad z_1 + z_3 \geq 1 \quad [Z \text{ takes some value}] \)

\[\begin{align*}
- v_1 + - w_1 + - y_1 + - z_1 & \geq -1 \\
- w_2 + - x_2 & \geq -1 \\
- w_3 + - x_3 + - y_3 + - z_3 & \geq -1 \\
- v_1 & \geq 1
\end{align*}\]  
[At most one variable = 1]
[At most one variable = 2]
[At most one variable = 3]

[Sum all constraints so far]
Justifying All-Different Failures

\[ V \in \{ 1, 4, 5 \} \]
\[ W \in \{ 1, 2, 3 \} \quad w_1 + w_2 + w_3 \geq 1 \quad [W \text{ takes some value}] \]
\[ X \in \{ 2, 3 \} \quad x_2 + x_3 \geq 1 \quad [X \text{ takes some value}] \]
\[ Y \in \{ 1, 3 \} \quad y_1 + y_3 \geq 1 \quad [Y \text{ takes some value}] \]
\[ Z \in \{ 1, 3 \} \quad z_1 + z_3 \geq 1 \quad [Z \text{ takes some value}] \]

\[ \rightarrow \quad -v_1 + -w_1 + -y_1 + -z_1 \geq -1 \quad [ \text{At most one variable } = 1 ] \]
\[ \rightarrow \quad -w_2 + -x_2 \geq -1 \quad [ \text{At most one variable } = 2 ] \]
\[ \rightarrow \quad -w_3 + -x_3 + -y_3 + -z_3 \geq -1 \quad [ \text{At most one variable } = 3 ] \]

\[ -v_1 \geq 1 \quad [\text{Sum all constraints so far}] \]
\[ v_1 \geq 0 \quad [\text{Variable } v_1 \text{ non-negative}] \]
Justifying All-Different Failures

\[ V \in \{1, 4, 5\} \]
\[ W \in \{1, 2, 3\} \quad w_1 + w_2 + w_3 \geq 1 \quad [W \text{ takes some value}] \]
\[ X \in \{2, 3\} \quad x_2 + x_3 \geq 1 \quad [X \text{ takes some value}] \]
\[ Y \in \{1, 3\} \quad y_1 + y_3 \geq 1 \quad [Y \text{ takes some value}] \]
\[ Z \in \{1, 3\} \quad z_1 + z_3 \geq 1 \quad [Z \text{ takes some value}] \]

\[ \rightarrow -v_1 + -w_1 + -y_1 + -z_1 \geq -1 \quad [\text{At most one variable} = 1] \]
\[ \rightarrow -w_2 + -x_2 \geq -1 \quad [\text{At most one variable} = 2] \]
\[ \rightarrow -w_3 + -x_3 + -y_3 + -z_3 \geq -1 \quad [\text{At most one variable} = 3] \]

\[ -v_1 \geq 1 \quad [\text{Sum all constraints so far}] \]
\[ v_1 \geq 0 \quad [\text{Variable } v_1 \text{ non-negative}] \]

\[ 0 \geq 1 \quad [\text{Sum above two constraints}] \]
Reformulation

Auto-tabulation is possible

- Heavy use of extension variables

Can re-encode maximum common subgraph as a clique problem, without changing pseudo-Boolean encoding
High Level Modelling Languages?

High level modelling languages like MINIZINC and ESSENCE have complicated compilers.

How do we know we’re giving a proof for the problem the user actually specified?

This would need a modelling language with formally specified semantics…
Code

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

Released under MIT Licence

Supports proof logging for global constraints including:

- All-different
- Integer linear inequality (including for very large domains)
- Smart table and regular
- Minimum / maximum of an array
- Element
- Absolute value
- (Hamiltonian) Circuit

Details in [EGMN20, GMN22, MM23]
Strengthening Rules (And Truth About Extension Variables)

When is it allowed to derive a new constraint? If it is (clear that it is) implied?
Strengthening Rules (And Truth About Extension Variables)

When is it allowed to derive a new constraint? If it is (clear that it is) implied?
Sometimes weaker criterion needed — recall that to get variable $a$ encoding

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

we introduced pseudo-Boolean constraints

$$3\bar{a} + 3x + 2y + z + w \geq 3 \quad 5a + 3\bar{x} + 2\bar{y} + \bar{z} + \bar{w} \geq 5$$

Cutting planes method inherently cannot certify such constraints — they are not implied!
Strengthening Rules (And Truth About Extension Variables)

When is it allowed to derive a new constraint? If it is (clear that it is) implied?
Sometimes weaker criterion needed — recall that to get variable $a$ encoding

\[ a \iff (3x + 2y + z + w \geq 3)(x \land y) \]

we introduced pseudo-Boolean constraints

\[ 3\bar{a} + 3x + 2y + z + w \geq 3 \quad 5a + 3\bar{x} + 2\bar{y} + \bar{z} + \bar{w} \geq 5 \]

Cutting planes method inherently cannot certify such constraints — they are not implied!

Wish to allow without-loss-of-generality arguments that can derive non-implied constraints
Redundance-Based Strengthening

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

Adding redundant constraints should be OK
Redundance-Based Strengthening

C is **redundant** with respect to F if F and F ∧ C are **equisatisfiable**

Adding redundant constraints should be OK

---

Redundance-based strengthening [BT19, GN21] (extending RAT rule of SAT prof logging)

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

\[
F \land \neg C \models (F \land C)^{\uparrow \omega}
\]
Redundance-Based Strengthening

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

Adding redundant constraints should be OK

**Redundance-based strengthening** [BT19, GN21] (extending RAT rule of SAT prof logging)

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

$$F \land \neg C \models (F \land C) \upharpoonright_\omega$$
Redundance-Based Strengthening

C is redundant with respect to F if F and \( F \land C \) are equisatisfiable.

Adding redundant constraints should be OK.

**Fact**
\( \alpha \) satisfies \( \phi \upharpoonright_\omega \) iff \( \alpha \circ \omega \) satisfies \( \phi \).

Redundance-based strengthening [BT19, GN21] (extending RAT rule of SAT prof logging)

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

\[
F \land \neg C \models (F \land C) \upharpoonright_\omega
\]

Proof sketch for interesting direction: If \( \alpha \) satisfies F but falsifies C, then \( \alpha \circ \omega \) satisfies \( F \land C \).
Redundance-Based Strengthening

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

Adding redundant constraints should be OK

Fact

$\alpha$ satisfies $\phi \upharpoonright \omega$ iff $\alpha \circ \omega$ satisfies $\phi$

Redundance-based strengthening [BT19, GN21] (extending RAT rule of SAT prof logging)

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

$$F \land \neg C \models (F \land C) \upharpoonright \omega$$

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

Witness $\omega$ should be specified, and implication should be efficiently verifiable, which is the case for constraints in $(F \land C) \upharpoonright \omega$ that are, e.g.,

- Reverse unit propagation (RUP) constraints w.r.t. $F \land \neg C$
- Obviously implied by a single constraint among $F \land \neg C$
Toy example of Redundance Rule

Choose binary encoding of two integers in $[0, 15]$ that sum up to 25 and are equal modulo two.
Toy example of Redundance Rule

Choose binary encoding of two integers in $[0, 15]$ that sum up to 25 and are equal modulo two

\[1 \cdot x_0 + 2 \cdot x_1 + 4 \cdot x_2 + 8 \cdot x_3\]
\[+ 1 \cdot y_0 + 2 \cdot y_1 + 4 \cdot y_2 + 8 \cdot y_3 = 25\]
\[x_0 = y_0\]
Toy example of Redundance Rule

Choose binary encoding of two integers in [0, 15] that sum up to 25 and are equal modulo two

\[1 \cdot x_0 + 2 \cdot x_1 + 4 \cdot x_2 + 8 \cdot x_3\]
\[+ 1 \cdot y_0 + 2 \cdot y_1 + 4 \cdot y_2 + 8 \cdot y_3 \geq 25\]
\[-1 \cdot x_0 - 2 \cdot x_1 - 4 \cdot x_2 - 8 \cdot x_3\]
\[-1 \cdot y_0 - 2 \cdot y_1 - 4 \cdot y_2 - 8 \cdot y_3 \geq -25\]
\[x_0 - y_0 \geq 0\]
\[y_0 - x_0 \geq 0\]
Redundance-Based Strengthening

Toy example of Redundance Rule

Choose binary encoding of two integers in [0, 15] that sum up to 25 and are equal modulo two

\[ \begin{align*}
1 \cdot x_0 + 2 \cdot x_1 + 4 \cdot x_2 + 8 \cdot x_3 \\
+ 1 \cdot y_0 + 2 \cdot y_1 + 4 \cdot y_2 + 8 \cdot y_3 & \geq 25 \\
-1 \cdot x_0 - 2 \cdot x_1 - 4 \cdot x_2 - 8 \cdot x_3 \\
-1 \cdot y_0 - 2 \cdot y_1 - 4 \cdot y_2 - 8 \cdot y_3 & \geq -25 \\
x_0 - y_0 & \geq 0 \\
y_0 - x_0 & \geq 0
\end{align*} \]

To derive without loss of generality \( x \leq y \) (argument: we can always swap them)

\[ 1 \cdot x_0 + 2 \cdot x_1 + 4 \cdot x_2 + 8 \cdot x_3 \leq 1 \cdot y_0 + 2 \cdot y_1 + 4 \cdot y_2 + 8 \cdot y_3 \]

pseudo-Boolean proof version 2.0

\[ \begin{array}{llllllllllll}
\text{f 4} & \text{red 1 y0 2 y1 4 y2 8 y3} \\
\rightarrow & -1 x0 -2 x1 -4 x2 -8 x3 & \geq & 0 \\
\rightarrow & y0 -> x0 x0 -> y0 y1 -> x1 x1 -> y1 \\
\rightarrow & y2 -> x2 x2 -> y2 y3 -> x3 x3 -> y3
\end{array} \]
Toy example of Redundance Rule

Choose binary encoding of two integers in \([0, 15]\) that sum up to 25 and are equal modulo two

\[
1 \cdot x_0 + 2 \cdot x_1 + 4 \cdot x_2 + 8 \cdot x_3 \\
+ 1 \cdot y_0 + 2 \cdot y_1 + 4 \cdot y_2 + 8 \cdot y_3 \geq 25 \\
-1 \cdot x_0 - 2 \cdot x_1 - 4 \cdot x_2 - 8 \cdot x_3 \\
-1 \cdot y_0 - 2 \cdot y_1 - 4 \cdot y_2 - 8 \cdot y_3 \leq -25 \\
x_0 - y_0 \geq 0 \\
y_0 - x_0 \geq 0
\]

To derive without loss of generality \(x \leq y\) (argument: we can always swap them)

\[
1 \cdot x_0 + 2 \cdot x_1 + 4 \cdot x_2 + 8 \cdot x_3 \leq 1 \cdot y_0 + 2 \cdot y_1 + 4 \cdot y_2 + 8 \cdot y_3
\]
Deriving \( a \iff (3x + 2y + z + w \geq 3) \) Using the Redundance Rule

Want to derive

\[
3\overline{a} + 3x + 2y + z + w \geq 3 \quad 5a + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5
\]

using condition \( F \land \neg C \models (F \land C) \upharpoonright_\omega \)
Deriving $a \iff (3x + 2y + z + w \geq 3)$ Using the Redundance Rule

Want to derive

$$3\bar{a} + 3x + 2y + z + w \geq 3 \quad 5a + 3\bar{x} + 2\bar{y} + z + \bar{w} \geq 5$$

using condition $F \land \neg C \models (F \land C)\upharpoonright_\omega$

1. $F \land \neg(3\bar{a} + 3x + 2y + z + w \geq 3) \models (F \land (3\bar{a} + 3x + 2y + z + w \geq 3))\upharpoonright_\omega$

Choose $\omega = \{a \mapsto 0\} - F$ untouched; new constraint satisfied
Deriving $a \iff (3x + 2y + z + w \geq 3)$ Using the Redundance Rule

Want to derive

\[ 3\bar{a} + 3x + 2y + z + w \geq 3 \quad 5a + 3\bar{x} + 2\bar{y} + \bar{z} + \bar{w} \geq 5 \]

using condition $F \land \neg C \models (F \land C)^\upharpoonright_\omega$

1. $F \land \neg (3\bar{a} + 3x + 2y + z + w \geq 3) \models (F \land (3\bar{a} + 3x + 2y + z + w \geq 3))^\upharpoonright_\omega$
   Choose $\omega = \{a \mapsto 0\} - F$ untouched; new constraint satisfied

2. $F \land (3\bar{a} + 3x + 2y + z + w \geq 3) \land \neg (5a + 3\bar{x} + 2\bar{y} + \bar{z} + \bar{w} \geq 5) \models$
   $(F \land (3\bar{a} + 3x + 2y + z + w \geq 3) \land (5a + 3\bar{x} + 2\bar{y} + \bar{z} + \bar{w} \geq 5))^\upharpoonright_\omega$
   Choose $\omega = \{a \mapsto 1\} - F$ untouched; new constraint satisfied
   $\neg (5a + 3\bar{x} + 2\bar{y} + \bar{z} + \bar{w} \geq 5)$ forces $3\bar{x} + 2\bar{y} + \bar{z} + \bar{w} \leq 4$
   This is the same constraint as $3\bar{a} + 3x + 2y + z + w \geq 3$
   And VERIPB can automatically detect this implication
Redundance and Dominance Rules for Optimisation

**Redundance-based strengthening, optimisation version**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

$$F \land \neg C \models (F \land C)^\omega \land f^\omega \leq f$$
Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

Add constraint \( C \) to formula \( F \) if exists witness substitution \( \omega \) s.t.

\[
F \land \neg C \models (F \land C)^\omega \land f^\omega \leq f
\]

Can be more aggressive if witness \( \omega \) strictly improves solution
Redundance and Dominance Rules for Optimisation

**Redundance-based strengthening, optimisation version**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

$$F \land \neg C \models (F \land C)\uparrow_\omega \land f\uparrow_\omega \leq f$$

Can be more aggressive if witness $\omega$ strictly improves solution

**Dominance-based strengthening (simplified)**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

$$F \land \neg C \models F\uparrow_\omega \land f\uparrow_\omega < f$$
Redundance and Dominance Rules for Optimisation

**Redundance-based strengthening, optimisation version**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

$$F \land \neg C \models (F \land C)^\uparrow_\omega \land f^\uparrow_\omega \leq f$$

Can be more aggressive if witness $\omega$ strictly improves solution.

**Dominance-based strengthening (simplified)**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

$$F \land \neg C \models F^\uparrow_\omega \land f^\uparrow_\omega < f$$

- Applying $\omega$ should strictly decrease $f$
- If so, don’t need to show that $C^\uparrow_\omega$ holds!
Soundness of Dominance Rule

**Dominance-based strengthening (simplified)**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

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

Why is this sound?
Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

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

Why is this sound?

1. Suppose $\alpha$ 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 $\omega$ s.t.

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

Why is this sound?

1. Suppose $\alpha$ 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 $\omega$ s.t.

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

Why is this sound?

1. Suppose $\alpha$ 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 \( \omega \) s.t.

\[
F \land \neg C \models F \uparrow \omega \land f \uparrow \omega < f
\]

Why is this sound?

1. Suppose \( \alpha \) 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 \( f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega) \)
Soundness of Dominance Rule

**Dominance-based strengthening (simplified)**

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

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

**Why is this sound?**

1. Suppose $\alpha$ 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 $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
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 $\omega$ s.t.

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

Why is this sound?

1. Suppose $\alpha$ 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 $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies $C$, we’re done
6. Otherwise $((\alpha \circ \omega) \circ \omega) \circ \omega$ satisfies $F$ and $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f((\alpha \circ \omega) \circ \omega)$
Redundance-Based and Dominance-Based Strengthening for Optimisation

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

$$F \land \lnot C \models F \uparrow_{\omega} \land f \uparrow_{\omega} < f$$

Why is this sound?

1. Suppose $\alpha$ satisfies $F$ but falsifies $C$ (i.e., satisfies $\lnot C$)
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 $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies $C$, we’re done
6. Otherwise $((\alpha \circ \omega) \circ \omega) \circ \omega$ satisfies $F$ and $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f((\alpha \circ \omega) \circ \omega)$
7. ...
Soundness of Dominance Rule

Dominance-based strengthening (simplified)

Add constraint $C$ to formula $F$ if exists witness substitution $\omega$ s.t.

$$F \land \lnot C \models F^{\uparrow \omega} \land f^{\uparrow \omega} < f$$

Why is this sound?

1. Suppose $\alpha$ satisfies $F$ but falsifies $C$ (i.e., satisfies $\lnot C$)
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 $f((\alpha \circ \omega) \circ \omega) < f(\alpha \circ \omega)$
5. If $(\alpha \circ \omega) \circ \omega$ satisfies $C$, we’re done
6. Otherwise $((\alpha \circ \omega) \circ \omega) \circ \omega$ satisfies $F$ and $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f((\alpha \circ \omega) \circ \omega)$
7. …
8. Can’t go on forever, so finally reach $\alpha'$ 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 $\omega$ 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 $\omega$ s.t.

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

Only consider $F$ — no need to show that any $C_i \downarrow_{\omega}$ 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 $\omega$ s.t.

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

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

Now why is *this* sound?

- Same inductive proof as before, but nested
- Or pick solution $\alpha$ 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 $\omega$ 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!

Now why is this sound?
- Same inductive proof as before, but nested
- Or pick solution $\alpha$ 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 [BGMN23] for details
Using the Dominance Rule for Symmetry Handling

Dominance rule very powerful; can be used for symmetry and dominance breaking
Using the Dominance Rule for Symmetry Handling

Dominance rule very powerful; can be used for symmetry and dominance breaking

Examples:

1. Symmetries in constraint programming (manual symmetry breaking)
2. Vertex dominance in clique solving (automatic dominance breaking during search)
3. Symmetries in SAT solving (automatic symmetry breaking in preprocessing)
Symmetry in Constraint Programming

Symmetry Elimination (CP)

The Crystal Maze Puzzle

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

Human modellers might add:

- (mirror vertically)
- (mirror horizontally)
- \( \leq 4 \) (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)

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)

Human modellers might add:

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

Are these valid simultaneously?

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

- Use permutation of variable-values as the witness $\omega$
- The constraints give us the order
- No group theory required!

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?

Can introduce these constraints inside the proof, rather than as part of the pseudo-Boolean model!
- Use permutation of variable-values as the witness \( \omega \)
- The constraints give us the order
- No group theory required!

Research challenge: Constraint programming 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 symmetry, but dominance
Lazy Global Domination for Maximum Clique [MP16]

Can ignore vertex 2b
- Every neighbour of 2b is also a neighbour of 2
- Not symmetry, but dominance

Dominance rule can justify this
- Even when detected dynamically during search
Strategy for SAT Symmetry Breaking in SAT Solving

1 Pretend to solve optimisation problem minimizing $f \doteq \sum_{i=1}^{n} 2^{n-i} \cdot x_i$
(search for lexicographically smallest assignment satisfying formula)
Strategy for SAT Symmetry Breaking in SAT Solving

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

2. Derive (for proof log only) pseudo-Boolean version of lex-leader constraint

\[
C_{\sigma} \doteq f \leq f \mid_{\sigma} \doteq \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0
\]
Strategy for SAT Symmetry Breaking in SAT Solving

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

2. Derive (for proof log only) pseudo-Boolean version of lex-leader constraint

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

3. Derive CNF encoding of lex-leader constraint used by SAT solver from pseudo-Boolean constraint (in same spirit as [GMNO22])

   $$y_0$$
   $$\overline{y}_{j-1} \lor \overline{x}_j \lor \sigma(x_j)$$
   $$\overline{y}_j \lor y_{j-1}$$
   $$\overline{y}_j \lor \sigma(x_j) \lor x_j$$
   $$y_j \lor \overline{y}_{j-1} \lor \overline{x}_j$$
   $$y_j \lor \overline{y}_{j-1} \lor \sigma(x_j)$$
Strategy for SAT Symmetry Breaking in SAT Solving

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

2. Derive (for proof log only) pseudo-Boolean version of lex-leader constraint

$$C_\sigma \equiv f \leq f_{\uparrow \sigma} \equiv \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(x_i) - x_i) \geq 0$$

3. Derive CNF encoding of lex-leader constraint used by SAT solver from pseudo-Boolean constraint (in same spirit as [GMNO22])

$$y_0 \geq 1$$  \hspace{2cm} \overline{y}_j + \overline{\sigma(x_j)} + x_j \geq 1$$

$$\overline{y}_{j-1} + \overline{x}_j + \sigma(x_j) \geq 1$$  \hspace{2cm} y_j + \overline{y}_{j-1} + \overline{x}_j \geq 1$$

$$\overline{y}_j + y_{j-1} \geq 1$$  \hspace{2cm} y_j + \overline{y}_{j-1} + \sigma(x_j) \geq 1$$
Symmetry Breaking: Example

Example: Pigeonhole principle (PHP) 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 (PHP) 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 (PHP) 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: “Pick smallest hole for pigeon 1, then smallest for pigeon 2, …”

$$f = 2^{11} \cdot p_{13} + 2^{10} \cdot p_{12} + 2^9 \cdot p_{11} + 2^8 \cdot p_{23} + \cdots + 1 \cdot p_{41}$$
Breaking a Single Simple Symmetry (Example)

- $F$ is a formula expressing PHP constraints with $F|_{\sigma(12)} = F$
- Add constraint $C_{12}$ breaking $\sigma_{(12)}$ — should be satisfied by $\alpha$ iff $\alpha$ “at least as good” as $\sigma(12)(\alpha)$
Breaking a Single Simple Symmetry (Example)

- $F$ is a formula expressing PHP constraints with $F \upharpoonright_{\sigma(12)} = F$
- Add constraint $C_{12}$ breaking $\sigma(12)$ — should be satisfied by $\alpha$ iff $\alpha$ “at least as good” as $\sigma(12)(\alpha)$

$$C_{12} \doteq f \leq f \upharpoonright_{\sigma(12)}$$
Breaking a Single Simple Symmetry (Example)

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

$$C_{12} \triangleq f \leq f \upharpoonright \sigma_{(12)}$$

$$\triangleq \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma_{(12)}(x_i) - x_i) \geq 0$$
Breaking a Single Simple Symmetry (Example)

- $F$ is a formula expressing PHP constraints with $F \upharpoonright_{\sigma(12)} = F$

- Add constraint $C_{12}$ breaking $\sigma(12)$ — should be satisfied by $\alpha$ iff $\alpha$ “at least as good” as $\sigma(12)(\alpha)$

\[
C_{12} \triangleq f \leq f \upharpoonright_{\sigma(12)}
\leq \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(12)(x_i) - x_i) \geq 0
\leq (2^{11} - 2^{8})(p_{23} - p_{13}) + (2^{10} - 2^{7})(p_{22} - p_{12}) + (2^9 - 2^6)(p_{21} - p_{11}) \geq 0
\]

“Pigeon 1 in smaller hole than pigeon 2”
Breaking a Single Simple Symmetry (Example)

- $F$ is a formula expressing PHP constraints with $F\upharpoonright_{\sigma(12)} = F$
- Add constraint $C_{12}$ breaking $\sigma(12)$ — should be satisfied by $\alpha$ iff $\alpha$ “at least as good” as $\sigma(12)(\alpha)$

\[
C_{12} \triangleq f \leq f\upharpoonright_{\sigma(12)} \\
\triangleq \sum_{i=1}^{n} 2^{n-i} \cdot (\sigma(12)(x_i) - x_i) \geq 0 \\
\triangleq (2^{11} - 2^{8})(p_{23} - p_{13}) + (2^{10} - 2^{7})(p_{22} - p_{12}) + (2^{9} - 2^{6})(p_{21} - p_{11}) \geq 0
\]

“Pigeon 1 in smaller hole than pigeon 2”

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

\[
F \land \neg C_{12} \models F\upharpoonright_{\sigma(12)} \land C_{12}\upharpoonright_{\sigma(12)} \land f\upharpoonright_{\sigma(12)} \leq f \\
F \land \neg(f \leq f\upharpoonright_{\sigma(12)}) \models F\upharpoonright_{\sigma(12)} \land (f \leq f\upharpoonright_{\sigma(12)})\upharpoonright_{\sigma(12)} \land f\upharpoonright_{\sigma(12)} \leq f
\]
Breaking a Single Simple Symmetry (Example)

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

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

“Pigeon 1 in smaller hole than pigeon 2”

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

$$F \land \neg C_{12} \models F\upharpoonright_{\sigma_{12}} \land C_{12}\upharpoonright_{\sigma_{12}} \land f\upharpoonright_{\sigma_{12}} \leq f$$
$$F \land f > f\upharpoonright_{\sigma_{12}} \models F\upharpoonright_{\sigma_{12}} \land f\upharpoonright_{\sigma_{12}} \leq f \land f\upharpoonright_{\sigma_{12}} \leq f$$
Breaking a Single Simple Symmetry (Example)

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

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

“Pigeon 1 in smaller hole than pigeon 2”

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

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

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

\[ F \land C_{12} \land \neg C_{23} \not\models \bar{f} \mid \sigma_{(23)} \land C_{12} \mid \sigma_{(23)} \land C_{23} \mid \sigma_{(23)} \land f \mid \sigma_{(23)} \leq f \]

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

- Breaking complex symmetries
Breaking More/Other Symmetries

**Problem**

*This idea does not generalize*

- Breaking two symmetries

\[ F \land C_{12} \land \neg C_{23} \not\models F \upharpoonright \sigma_{(23)} \land C_{12} \upharpoonright \sigma_{(23)} \land C_{23} \upharpoonright \sigma_{(23)} \land f \downarrow \sigma_{(23)} \leq f \]

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

  \[ F \land C_{12} \land \neg C_{23} \not\models F_{\sigma(23)} \land C_{12} \upharpoonright_{\sigma(23)} \land C_{23} \upharpoonright_{\sigma(23)} \land f \upharpoonright_{\sigma(23)} \leq f \]

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

- Breaking complex symmetries

  \[ F \land \neg C_{1234} \models F_{\sigma_{(1234)}} \land C_{1234} \upharpoonright_{\sigma_{(1234)}} \land f \upharpoonright_{\sigma_{(1234)}} \leq f \]

  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

\[ F \land C_{12} \land \neg C_{23} \not\models F\models_{\sigma_{(23)}} \land C_{12}\models_{\sigma_{(23)}} \land C_{23}\models_{\sigma_{(23)}} \land f\models_{\sigma_{(23)}} \leq f \]

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

- Breaking complex symmetries

\[ F \land \neg C_{1234} \models F\models_{\sigma_{(1234)}} \land C_{1234}\models_{\sigma_{(1234)}} \land f\models_{\sigma_{(1234)}} \leq f \]

Intuitively, \( C_{1234} \) holds if shifting all the pigeons results in a worse assignment
Can satisfy this constraint by applying \( \sigma_{(1234)} \) once, twice, or thrice
Breaking Symmetries with the Dominance Rule (1/2)

Definition

Given a symmetry $\sigma$, the (pseudo-Boolean) breaking constraint of $\sigma$ is

$$C_\sigma \triangleq f \leq f \upharpoonright \sigma$$
Breaking Symmetries with the Dominance Rule (1/2)

**Definition**
Given a symmetry $\sigma$, the (pseudo-Boolean) breaking constraint of $\sigma$ is

$$C_\sigma \doteq f \leq f \upharpoonright \sigma$$

**Theorem ([BGMN23])**
$C_\sigma$ can be derived from $F$ using dominance with witness $\sigma$

$$F \land \neg C_\sigma \models F \upharpoonright \sigma \land f \upharpoonright \sigma < f$$
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 (can ignore previously derived symmetry breaking constraints)

\[ F \land C_{12} \land \neg C_{23} \models F\upharpoonright_{\sigma_{(23)}} \land f\upharpoonright_{\sigma_{(23)}} < f \]
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 (can ignore previously derived symmetry breaking constraints)

\[
F \land C_{12} \land \neg C_{23} \models F\upharpoonright_{\sigma(23)} \land f\upharpoonright_{\sigma(23)} < f
\]

Why does it work?

- Witness need not satisfy all derived constraints
- Sufficient to just produce “better” assignment
Strategy for SAT Symmetry Breaking in SAT Solving

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

2. Derive (for proof log only) pseudo-Boolean version of lex-leader constraint

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

3. Derive CNF encoding of lex-leader constraint used by SAT solver from pseudo-Boolean constraint (in same spirit as [GMNO22])

\[
\begin{align*}
y_0 & \geq 1 \\
\bar{y}_{j-1} + \bar{x}_j + \sigma(x_j) & \geq 1 \\
\bar{y}_j + y_{j-1} & \geq 1 \\
\bar{y}_j + \bar{y}_{j-1} + x_j & \geq 1 \\
y_j + \bar{y}_{j-1} + \bar{x}_j & \geq 1 \\
y_j + \bar{y}_j + \sigma(x_j) & \geq 1
\end{align*}
\]
Symmetry Breaking in CNF

- In SAT symmetry breaking tools, symmetry is broken by adding clausal constraints
Symmetry Breaking in CNF

- In SAT symmetry breaking tools, symmetry is broken by adding clausal constraints
- Need to show how to derive this CNF encoding
Symmetry Breaking in CNF

- In SAT symmetry breaking tools, symmetry is broken by adding clausal constraints.
- Need to show how to derive this CNF encoding.
- We use the encoding of *BreakID* [DBBD16]:

\[
\begin{align*}
y_0 & \geq 1 \\
\overline{y}_{j-1} + \overline{x}_j + \sigma(x_j) & \geq 1 \\
\overline{y}_j + y_{j-1} & \geq 1 \\
\overline{y}_j + \sigma(x_j) + x_j & \geq 1 \\
y_j + \overline{y}_{j-1} + \overline{x}_j & \geq 1 \\
y_j + \overline{y}_{j-1} + \sigma(x_j) & \geq 1
\end{align*}
\]
Symmetry Breaking in CNF

- In SAT symmetry breaking tools, symmetry is broken by adding clausal constraints.
- Need to show how to derive this CNF encoding.
- We use the encoding of *BreakID* [DBBD16]:

\[
\begin{align*}
    y_0 & \geq 1 \\
    \bar{y}_{j-1} + \bar{x}_j + \sigma(x_j) & \geq 1 \\
    \bar{y}_j + y_{j-1} & \geq 1 \\
    \bar{y}_j + \sigma(x_j) + x_j & \geq 1 \\
    y_j + \bar{y}_{j-1} + \bar{x}_j & \geq 1 \\
    y_j + \bar{y}_{j-1} + \sigma(x_j) & \geq 1
\end{align*}
\]

Define \( y_j \) true if \( x_k \) equals \( \sigma(x_k) \) for all \( k \leq j \):

\[
y_k \iff y_{k-1} \land (x_k \iff \sigma(x_k))
\]

(derivable with redundancy rule)
Symmetry Breaking in CNF

- In SAT symmetry breaking tools, symmetry is broken by adding clausal constraints.
- Need to show how to derive this CNF encoding.
- We use the encoding of BreakID [DBBD16]:

\[
\begin{align*}
y_0 & \geq 1 \\
\overline{y}_{j-1} + \overline{x} + \sigma(x_j) & \geq 1 \\
\overline{y}_j + y_{j-1} & \geq 1 \\
\overline{y}_j + \sigma(x_j) + x_j & \geq 1 \\
y_j + \overline{y}_{j-1} + \overline{x}_j & \geq 1 \\
y_j + \overline{y}_{j-1} + \sigma(x_j) & \geq 1
\end{align*}
\]

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

\[
y_k \iff y_{k-1} \land (x_k \iff \sigma(x_k))
\]

(derivable with redundancy rule)

If \( y_{k-1} \) is true, \( x_k \) is at most \( \sigma(x_k) \)
(derivable from the PB breaking constraint)
Back to Our Pigeons — Setting up the Pretend Optimisation Problem

Start the proof and load input formula

```pseudo-Boolean proof version 2.0
f 22
pre_order exp
vars
  left u1 u2 u3 u4 u5 u6 u7 u8 u9 u10 u11 u12
  right v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12
  aux
end
def
  -1 u12 1 v12 -2 u11 2 v11 [...] -1024 u2 1024 v2 -2048 u1 2048 v1 >= 0;
end
transitivity
vars
  fresh_right w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12
end
proof
proofgoal #1
  pol 1 2 + 3 +
  qed -1
  qed
end
load_order exp p13 p12 p11 p23 p22 p21 p31 p32 p33 p41 p42 p43
```
Back to Our Pigeons — Setting up the Pretend Optimisation Problem

Start the proof and load input formula

1. Pretend to solve optimisation problem

minimizing $f = 2^{11} \cdot p_{13} + 2^{10} \cdot p_{12} + 2^9 \cdot p_{11} + 2^8 \cdot p_{23} + \cdots + 2 \cdot p_{42} + 1 \cdot p_{41}$

pseudo-Boolean proof version 2.0

```
f 22
pre_order exp
  vars
    left u1 u2 u3 u4 u5 u6 u7 u8 u9 u10 u11 u12
    right v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12
    aux
  end
  def
    -1 u12 1 v12 -2 u11 2 v11 [...] -1024 u2 1024 v2 -2048 u1 2048 v1 >= 0;
  end
  transitivity
    vars
      fresh_right w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12
    end
  proof
    proofgoal #1
      pol 1 2 + 3 +
    qed -1
    qed
    end
end
load_order exp p13 p12 p11 p23 p22 p21 p31 p32 p33 p41 p42 p43
```
Back to Our Pigeons — Setting up the Pretend Optimisation Problem

Start the proof and load input formula

1. Pretend to solve optimisation problem

   minimizing \( f = \sum \left(2^{11} \cdot p_{13} + 2^{10} \cdot p_{12} + 2^9 \cdot p_{11} + 2^8 \cdot p_{23} + \cdots + 2 \cdot p_{42} + 1 \cdot p_{41}\right)\)

   (Actually defining an order — see [BGMN23] for details)

pseudo-Boolean proof version 2.0

```plaintext
f 22
pre_order exp
vars
  left u1 u2 u3 u4 u5 u6 u7 u8 u9 u10 u11 u12
  right v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12
  aux
end
def
-1 u12 1 v12 -2 u11 2 v11 […] -1024 u2 1024 v2 -2048 u1 2048 v1 >= 0;
end
transitivity
vars
  fresh_right w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12
end
proof
proofgoal #1
  pol 1 2 + 3 +
  qed -1
  qed
end
load_order exp p13 p12 p11 p23 p22 p21 p31 p32 p33 p41 p42 p43
```
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($D$):

$$2^{11} \cdot (p_{23} - p_{13}) +$$
$$2^{10} \cdot (p_{22} - p_{12}) +$$
$$\ldots \geq 0$$
Back to Our Pigeons — Deriving the Constraints

Derived constraints (\(\mathcal{D}\)):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\ldots & \geq 0
\end{align*}
\]

Pseudo-Boolean breaking constraint

Use dominance with witness \(\sigma = (p_{11}p_{21})(p_{12}p_{22})(p_{13}p_{23})\)
Symmetry Breaking for SAT

Back to Our Pigeons — Deriving the Constraints

Derived constraints ($D$):

\[
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots \geq 0
\]

Pseudo-Boolean breaking constraint

Use dominance with witness \(\sigma = (p_{11}p_{21})(p_{12}p_{22})(p_{13}p_{23})\)

\[
F \land \neg C_{12} \models F|_{\omega} \land (f|_{\omega} < f)
\]

\textsc{VeriPB} fills in all missing subproofs except for \(\neg C_{12} \land C_{12} \models \bot\)
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots & \geq 0
\end{align*}
\]

\[y_0 \geq 1\]

Derivable by redundancy with witness $\omega = \{y_0 \mapsto 1\}$

\[
F \land \mathcal{D} \land \neg(y_0 \geq 1) \models (F \land \mathcal{D})\upharpoonright_\omega \land (y_0 \geq 1)\upharpoonright_\omega
\]
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots \geq 0
\]

\[y_0 \geq 1\]

Derivable by redundancy with witness $\omega = \{y_0 \rightarrow 1\}$

\[
F \land \mathcal{D} \land \neg (y_0 \geq 1) \models (F \land \mathcal{D}) \upharpoonright_\omega \land (y_0 \geq 1) \upharpoonright_\omega \\
F \land \mathcal{D} \land (\bar{y}_0 \geq 1) \models (F \land \mathcal{D}) \land (1 \geq 1)
\]
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots & \geq 0
\end{align*}
\]

\[
y_0 \geq 1
\]

\[
\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1
\]

Derivable by RUP

\[
F \land \mathcal{D} \land \neg(\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1)
\]
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($D$):

\[ 2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots \geq 0 \]

\[ y_0 \geq 1 \]

\[ \overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1 \]

Derivable by RUP

\[ F \land D \land \neg(\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1) \models F \land D \land (y_0 \geq 1) \land (p_{13} \geq 1) \land (\overline{p}_{23} \geq 1) \]
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\ldots & \geq 0
\end{align*}
\]

\[
y_0 \geq 1
\]

\[
\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1
\]

Derivable by **RUP**

\[
F \land \mathcal{D} \land \neg(\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1) \models F \land \mathcal{D} \land (y_0 \geq 1) \land (p_{13} \geq 1) \land (\overline{p}_{23} \geq 1)
\]

\[
2^{11} \cdot (p_{23} - p_{13}) + 2^{10} \cdot (p_{22} - p_{12}) + \ldots \geq 0
\]
Back to Our Pigeons — Deriving the Constraints

**Derived constraints** \((D)\):

\[
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots \geq 0
\]

\[
y_0 \geq 1
\]

\[
\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1
\]

Derivable by RUP

\[
F \land D \land \neg(\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1) \models F \land D \land (y_0 \geq 1) \land (p_{13} \geq 1) \land (\overline{p}_{23} \geq 1)
\]

\[
2^{11} \cdot (1) + 2^{10} \cdot (p_{22} - p_{12}) + \cdots \geq 0
\]
Back to Our Pigeons — Deriving the Constraints

**Derived constraints (D):**

\[
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots \geq 0
\]

\[
y_0 \geq 1
\]

\[
\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1
\]

Derivable by **RUP**

\[
F \land D \land \neg (\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1) \models F \land D \land (y_0 \geq 1) \land (p_{13} \geq 1) \land (\overline{p}_{23} \geq 1)
\]

\[
2^{11} \cdot (-1) + 2^{10} \cdot (p_{22} - p_{12}) + \cdots \geq 0
\]

where \( \sum_{i=1}^{10} 2^i < 2^{11} \)
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots \geq 0
\]

\[y_0 \geq 1\]

\[\bar{y}_0 + \overline{p_{13}} + \sigma(p_{13}) \geq 1\]

\[\bar{y}_1 + y_0 \geq 1\]

Derivable by redundancy with witness $\omega = \{y_1 \mapsto 0\}$

\[
F \land \mathcal{D} \land \neg(\bar{y}_1 + y_0 \geq 1) \\
\models (F \land \mathcal{D}) \upharpoonright_\omega \land (\bar{y}_1 + y_0 \geq 1) \upharpoonright_\omega
\]
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots & \geq 0
\end{align*}
\]

\[
\begin{align*}
y_0 & \geq 1 \\
\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) & \geq 1 \\
\overline{y}_1 + y_0 & \geq 1
\end{align*}
\]

Derivable by redundancy with witness \( \omega = \{y_1 \mapsto 0\} \)

\[
F \land \mathcal{D} \land \neg(\overline{y}_1 + y_0 \geq 1) \\
\models (F \land \mathcal{D})\upharpoonright_\omega \land (\overline{y}_1 + y_0 \geq 1)\upharpoonright_\omega
\]

\[
F \land \mathcal{D} \land (y_1 + \overline{y}_0 \geq 2) \\
\models (F \land \mathcal{D}) \land (1 + y_0 \geq 1)
\]
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($D$):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots & \geq 0 \\
y_0 & \geq 1 \\
\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) & \geq 1 \\
\overline{y}_1 + y_0 & \geq 1 \\
\overline{y}_1 + \sigma(p_{13}) + p_{13} & \geq 1
\end{align*}
\]

Derivable by redundancy with witness $\omega = \{y_1 \mapsto 0\}$
(essentially same argument)
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($D$):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots & \geq 0 \\
y_0 & \geq 1 \\
\bar{y}_0 + \bar{p}_{13} + \sigma(p_{13}) & \geq 1 \\
\bar{y}_1 + y_0 & \geq 1 \\
\bar{y}_1 + \sigma(p_{13}) + p_{13} & \geq 1 \\
y_1 + \bar{y}_0 + \bar{p}_{13} & \geq 1
\end{align*}
\]

Derivable by redundancy with witness $\omega = \{y_1 \mapsto 1\}$

\[
F \land D \land \neg(y_1 + \bar{y}_0 + \bar{p}_{13} \geq 1) \\
\models (F \land D) \upharpoonright_\omega \land (y_1 + \bar{y}_0 + \bar{p}_{13} \geq 1) \upharpoonright_\omega
\]
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[ 2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots \geq 0 \]
\[ y_0 \geq 1 \]
\[ \overline{y}_0 + \overline{p}_{13} + \sigma (p_{13}) \geq 1 \]
\[ \overline{y}_1 + y_0 \geq 1 \]
\[ \overline{y}_1 + \frac{\sigma (p_{13})}{p_{13}} + p_{13} \geq 1 \]
\[ y_1 + \overline{y}_0 + \overline{p}_{13} \geq 1 \]

Derivable by redundancy with witness $\omega = \{ y_1 \mapsto 1 \}$

\[
F \wedge \mathcal{D} \wedge \neg (y_1 + \overline{y}_0 + \overline{p}_{13} \geq 1) \\
\models (F \wedge \mathcal{D}) \mid_{\omega} \wedge (y_1 + \overline{y}_0 + \overline{p}_{13} \geq 1) \mid_{\omega} \\
F \wedge \mathcal{D} \wedge (\overline{y}_1 + y_0 + p_{13} \geq 3) \\
\models \cdots \wedge \mathcal{D} \mid_{\omega} \wedge \cdots
\]
Symmetry Breaking for SAT

Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots & \geq 0 \\
y_0 & \geq 1 \\
\bar{y}_0 + \bar{p}_{13} + \sigma(p_{13}) & \geq 1 \\
\frac{\bar{y}_1 + y_0}{y_0} & \geq 1 \\
\bar{y}_1 + \sigma(p_{13}) + p_{13} & \geq 1 \\
y_1 + \bar{y}_0 + \bar{p}_{13} & \geq 1
\end{align*}
\]

Derivable by redundancy with witness $\omega = \{y_1 \mapsto 1\}$

\[
\begin{align*}
F \land \mathcal{D} \land \neg(y_1 + \bar{y}_0 + \bar{p}_{13} \geq 1) & \\
\models (F \land \mathcal{D}) \upharpoonright_\omega \land (y_1 + \bar{y}_0 + \bar{p}_{13} \geq 1) \upharpoonright_\omega \\
F \land \mathcal{D} \land (\bar{y}_1 + y_0 + p_{13} \geq 3) & \\
\models \cdots \land \mathcal{D} \upharpoonright_\omega \land \cdots
\end{align*}
\]
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots & \geq 0 \\
y_0 & \geq 1 \\
\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) & \geq 1 \\
\overline{y}_1 + y_0 & \geq 1 \\
\overline{y}_1 + \sigma(p_{13}) + p_{13} & \geq 1 \\
y_1 + \overline{y}_0 + \overline{p}_{13} & \geq 1 \\
\end{align*}
\]

Derivable by *redundance* with witness $\omega = \{y_1 \mapsto 1\}$

\[
F \land \mathcal{D} \land \neg(y_1 + \overline{y}_0 + \overline{p}_{13} \geq 1) \\
\models (F \land \mathcal{D}) \upharpoonright_\omega \land (y_1 + \overline{y}_0 + \overline{p}_{13} \geq 1) \upharpoonright_\omega \\
F \land \mathcal{D} \land (\overline{y}_1 + y_0 + p_{13} \geq 3) \\
\models \cdots \land \mathcal{D} \upharpoonright_\omega \land \cdots
\]
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[ 2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\cdots \geq 0 \]

\[ y_0 \geq 1 \]
\[ \bar{y}_0 + \bar{p}_{13} + \sigma(p_{13}) \geq 1 \]
\[ \bar{y}_1 + y_0 \geq 1 \]
\[ \bar{y}_1 + \sigma(p_{13}) + p_{13} \geq 1 \]
\[ y_1 + \bar{y}_0 + \bar{p}_{13} \geq 1 \]
\[ y_1 + \bar{y}_0 + \sigma(p_{13}) \geq 1 \]

Derivable by redundancy with witness $\omega = \{ y_1 \mapsto 1 \}$ (same argument)
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\ldots \geq 0
\]

\[
y_0 \geq 1
\]

\[
\overline{y}_0 + \overline{p}_{13} + \sigma(p_{13}) \geq 1
\]

\[
\overline{y}_1 + y_0 \geq 1
\]

\[
\overline{y}_1 + \sigma(p_{13}) + p_{13} \geq 1
\]

\[
y_1 + \overline{y}_0 + \overline{p}_{13} \geq 1
\]

\[
y_1 + \overline{y}_0 + \sigma(p_{13}) \geq 1
\]

\[
2^{11} \cdot \overline{y}_1 + 2^{10} \cdot (p_{22} - p_{12}) \ldots \geq 1
\]

Simplify the pseudo-Boolean breaking constraint and delete original constraint
Back to Our Pigeons — Deriving the Constraints

Derived constraints ($\mathcal{D}$):

\[
\begin{align*}
2^{11} \cdot (p_{23} - p_{13}) + \\
2^{10} \cdot (p_{22} - p_{12}) + \\
\ldots & \geq 0 \\
y_0 & \geq 1 \\
\overline{y}_0 + p_{13} + \sigma(p_{13}) & \geq 1 \\
\overline{y}_1 + y_0 & \geq 1 \\
\overline{y}_{1} + \sigma(p_{13}) + p_{13} & \geq 1 \\
y_1 + \overline{y}_0 + p_{13} & \geq 1 \\
y_1 + \overline{y}_0 + \sigma(p_{13}) & \geq 1 \\
2^{11} \cdot \overline{y}_1 + 2^{10} \cdot (p_{22} - p_{12}) \ldots & \geq 1 \\
\overline{y}_1 + p_{12} + \sigma(p_{22}) & \geq 1
\end{align*}
\]
Future Research Directions

**Performance and reliability of pseudo-Boolean proof logging**

- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker (*work in progress* [BMM+23])
Future Research Directions

Performance and reliability of pseudo-Boolean proof logging
- Trim proof while verifying (as in DRAT-Trim [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker (work in progress [BMM+23])

Proof logging for other combinatorial problems and techniques
- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming (some work on SCIP in [CGS17, EG21])
- Satisfiability modulo theories (SMT) solving (some work by Bjørner and others)
- High-level modelling languages
Future Research Directions

**Performance and reliability of pseudo-Boolean proof logging**
- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker (*work in progress* [BMM+23])

**Proof logging for other combinatorial problems and techniques**
- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming (*some work on SCIP* in [CGS17, EG21])
- Satisfiability modulo theories (SMT) solving (*some work by Bjørner and others*)
- High-level modelling languages

**And more...**
- Use proof logs for algorithm analysis or explainability purposes
- Lots of other challenging problems and interesting ideas
Future Research Directions

**Performance and reliability of pseudo-Boolean proof logging**
- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Design formally verified proof checker *(work in progress [BMM+23]*)

**Proof logging for other combinatorial problems and techniques**
- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(some work on SCIP in [CGS17, EG21]*)
- Satisfiability modulo theories (SMT) solving *(some work by Bjørner and others*)
- High-level modelling languages

**And more...**
- Use proof logs for algorithm analysis or explainability purposes
- Lots of other challenging problems and interesting ideas
- Talk to us if you want to join the proof logging revolution! 😊
  We’re happy to collaborate, and we’re hiring
Summing up

- **Combinatorial solving and optimization** is a true success story.

- But **ensuring correctness** is a crucial, and not yet satisfactorily addressed, concern.

- **Certifying solvers producing machine-verifiable proofs** of correctness seems like most promising approach.

- **Cutting planes reasoning** with **pseudo-Boolean constraints** seems to hit a sweet spot between simplicity and expressivity.
Summing up

- **Combinatorial solving and optimization** is a true success story

- But **ensuring correctness** is a crucial, and not yet satisfactorily addressed, concern

- **Certifying solvers** producing **machine-verifiable proofs** of correctness seems like most promising approach

- **Cutting planes reasoning** with **pseudo-Boolean constraints** seems to hit a sweet spot between simplicity and expressivity

- **Action point:** What problems can VERIPB solve for you?
Summing up

- **Combinatorial solving and optimization** is a true success story

- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern

- **Certifying solvers** producing **machine-verifiable proofs** of correctness seems like most promising approach

- Cutting planes reasoning with **pseudo-Boolean constraints** seems to hit a sweet spot between simplicity and expressivity

- **Action point**: What problems can VErIPB solve for you?

The end.
Summing up

- **Combinatorial solving and optimization** is a true success story

- But **ensuring correctness** is a crucial, and not yet satisfactorily addressed, concern

- **Certifying solvers** producing **machine-verifiable proofs** of correctness seems like most promising approach

- **Cutting planes reasoning** with **pseudo-Boolean constraints** seems to hit a sweet spot between simplicity and expressivity

- **Action point:** What problems can V̂ER̂IPB solve for you?

The end. Or rather, the beginning!
References for Getting Started with VeriPB

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

Released under MIT Licence

Various features to help development:
- Extended variable name syntax allowing human-readable names
- Proof tracing
- “Trust me” assertions for incremental proof logging

Documentation:
- Specific details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23]
- Lots of concrete example files at https://gitlab.com/MIAOresearch/software/VeriPB


References II


References III


References IV


References V


References VI


References VII


References VIII


Parity Reasoning: Experimental Evaluation

Implemented parity reasoning and PB proof logging engine\(^2\)

Also DRAT proof logging for XOR constraints as described in [PR16]

Experiments with MINISAT\(^3\)

Set-up:\(^4\)

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

---

\(^2\)https://gitlab.com/MIAOresearch/tools-and-utlities/xorengine
\(^3\)http://minisat.se/
\(^4\)Tools, benchmarks, data and evaluation scripts available at https://doi.org/10.5281/zenodo.7083485
Parity Reasoning: Proof Size for DRAT and PB Proof Logging

Proof sizes for Tseitin formulas using DRAT and pseudo-Boolean proof logging
Parity Reasoning: Solving and Proof Checking Time

Solving and proof checking time for Tseitin formulas using DRAT and PB proof logging
Parity Reasoning: Crypto Track of SAT 2021 Competition

Cumulative plot for the crypto track of the SAT Competition 2021
Parity Reasoning: Crypto Track Proof Size

DRAT and PB proof sizes for crypto track of SAT Competition 2021
Parity Reasoning: Crypto Track Solving & Proof Checking Time

![Graph showing solving and proof checking times for cryptographic instances]

Time required for solving and proof checking for cryptographic instances.
PB-to-CNF Translation: Experimental Evaluation

- Certified translations for CNF encodings with VeritasPBLib\(^5\)
  - Sequential counter [Sin05]
  - Totalizer [BB03]
  - Generalized totalizer [JMM15]
  - Adder network [ES06]
- Proofs verified by proof checker VERIPB
- Formulas solved with fork of KISSAT\(^6\) syntactically modified to output VERIPB proofs
- Benchmarks from PB 2016 Evaluation\(^7\) in 3 categories
  - Only cardinality constraints (sequential counter, totalizer)
  - Only general 0-1 ILP constraints (generalized totalizer, adder network)
  - Mixed cardinality & general 0-1 ILP constraints (sequential counter + adder network)

---

\(^5\)https://github.com/forge-lab/VeritasPBLib
\(^6\)https://gitlab.com/MIAOresearch/tools-and-utilities/kissat_fork
\(^7\)http://www.cril.univ-arthois.fr/PB16/
PB-to-CNF: CNF Size vs Proof Size in KiB

- Nice scaling for proof size in terms of original CNF formula size
- Except for some sequential encoding cases (which is not such a great encoding anyway)
PB-to-CNF: Translation Time vs Proof Checking Time in Seconds

- Translation faster — only has to generate clauses and proof
- Proof checking slower — has to verify full proof
PB-to-CNF: Solving Time vs Proof Checking Time in Seconds

- Room for improvement of end-to-end proof checking process
- But even first proof-of-concept implementation shows our approach is viable
Clique Solving: Experimental Evaluation

- Implemented in the *Glasgow Subgraph Solver*
  - Bit-parallel, can perform a colouring and recursive call in under a microsecond
- 59 of the 80 DIMACS instances take under 1,000 seconds to solve without logging
- Produced and verified proofs for 57 of these 59 instances (the other two reached 1TByte disk space)
- Mean slowdown from proof logging is 80.1 (due to disk I/O)
- Mean verification slowdown a further 10.1
- Approximate implementation effort: one Masters student
The Pseudo-Boolean models can be large: had to restrict to instances with no more than 260 vertices in the target graph.

Took enumeration instances which could be solved without proof logging in under ten seconds.

1,227 instances from Solnon’s benchmark collection:
- 789 unsatisfiable, up to 50,635,140 solutions in the rest
- 498 instances solved without guessing
- Hardest solved satisfiable and unsatisfiable instances required 53,605,482 and 2,074,386 recursive calls.
Subgraph Isomorphism Solving: Experimental Evaluation (2/3)

![Graph showing instances solved vs time](image)
Subgraph Isomorphism Solving: Experimental Evaluation (3/3)

![Graph showing the relationship between OPB + Proof Log Size and Time with Proof Logging (ms). The x-axis represents Time with Proof Logging in milliseconds, ranging from 1ms to 10s. The y-axis represents OPB + Proof Log Size, ranging from 1K to 100G. The graph includes color coding for Time without Proof Logging, with colors ranging from 1ms to 10s.](image-url)
Constraint Programming: How Expensive is Proof Logging? (1/2)

- Five benchmark problems allowing comparison of solvers “doing the same thing”:
  - Simple models
  - Fixed search order and well-defined propagation consistency levels
  - Few global constraints
- Probably close to the worst case for proof logging performance
- Also: Crystal Maze and World’s Hardest Sudoku
Constraint Programming: How Expensive is Proof Logging? (2/2)

- Our solver: faster than the fastest of MiniCP, OscaR, and Choco
- Proof logging slowdown: between 8.4 and 61.1 factor
  - 800,000 to 3,000,000 inferences per second
  - Proof logs can be hundreds of GBytes
  - No effort put into making the proof-writing code run fast
- Verification slowdown: a further factor 10 to 100
  - Probably possible to reduce this substantially if we are prepared to put more care into writing proofs
SAT Symmetry Breaking: Experimental Evaluation

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

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

![Graph showing BreakID and proof logging times](image)