The Joy of Pseudo-Boolean Proof Logging

Jakob Nordström

University of Copenhagen and Lund University

ACP Summer School 2026
CP: Explanation, Verification, and Applications
Changchun, China
August 24–28, 2026

The Joy of Pseudo-Boolean Proof Logging

Or: An Introduction to VeriPB

Jakob Nordström

University of Copenhagen and Lund University

ACP Summer School 2026
CP: Explanation, Verification, and Applications
Changchun, China
August 24–28, 2026

Brief Presentation

Jakob Nordström
University of Copenhagen
and Lund University
jn@di.ku.dk
www.jakobnordstrom.se

Research interests:

- Computational complexity theory
- Applied combinatorial optimization
- Certifying algorithms / proof logging

Brief Presentation

Jakob Nordström
University of Copenhagen
and Lund University
jn@di.ku.dk
www.jakobnordstrom.se

Research interests:

- *Proving exponential lower bounds for* NP*-hard problems*
- *Designing algorithms to solve* NP*-hard problems in (ideally) linear time*
- *Generating machine-verifiable proofs that such algorithms reason correctly*

Tutorial Based on Research Contributions By…

- Berhan Oumer Adame
- Markus Anders
- Jeremias Berg
- Bart Bogaerts
- Benjamin Bogø
- Wolf De Wulf
- Emir Demirović
- Simon Dold
- Jan Elffers
- Ambros Gleixner
- Stephan Gocht
- Arthur Gontier
- Malte Helmert
- Alexander Hoen
- Hannes Ihalainen
- Matti Järvisalo
- George Katsirelos
- Wietze Koops
- Daniel Le Berre
- Ruben Martins
- Ross McBride
- Ciaran McCreesh
- Matthew McIlree
- Magnus O. Myreen
- Andy Oertel
- Tobias Paxian
- Patrick Prosser
- Adrián Rebola-Pardo
- Philip Rodgers
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- Yong Kiam Tan
- James Trimble
- Mark Turnbull
- Dieter Vandesande
- Marc Vinyals
- Wei-Lin Wu

Tutorial Based on Research Contributions By…

- Berhan Oumer Adame
- Markus Anders
- Jeremias Berg
- **Bart Bogaerts**
- Benjamin Bogø
- Wolf De Wulf
- Emir Demirović
- Simon Dold
- Jan Elffers
- Ambros Gleixner
- **Stephan Gocht**
- Arthur Gontier
- Malte Helmert
- Alexander Hoen
- Hannes Ihalainen
- Matti Järvisalo
- George Katsirelos
- Wietze Koops
- Daniel Le Berre
- Ruben Martins
- Ross McBride
- **Ciaran McCreesh**
- Matthew McIlree
- Magnus O. Myreen
- Andy Oertel
- Tobias Paxian
- Patrick Prosser
- Adrián Rebola-Pardo
- Philip Rodgers
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- Yong Kiam Tan
- James Trimble
- Mark Turnbull
- Dieter Vandesande
- Marc Vinyals
- Wei-Lin Wu

Tutorial Based on Research Contributions By…

- Berhan Oumer Adame
- Markus Anders
- Jeremias Berg
- **Bart Bogaerts**
- **Benjamin Bogø**
- Wolf De Wulf
- Emir Demirović
- Simon Dold
- Jan Elffers
- Ambros Gleixner
- **Stephan Gocht**
- Arthur Gontier
- Malte Helmert
- Alexander Hoen
- Hannes Ihalainen
- Matti Järvisalo
- George Katsirelos
- **Wietze Koops**
- Daniel Le Berre
- Ruben Martins
- Ross McBride
- **Ciaran McCreesh**
- **Matthew McIlree**
- Magnus O. Myreen
- **Andy Oertel**
- Tobias Paxian
- Patrick Prosser
- Adrián Rebola-Pardo
- **Philip Rodgers**
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- Yong Kiam Tan
- James Trimble
- Mark Turnbull
- Dieter Vandesande
- Marc Vinyals
- Wei-Lin Wu

Tutorial Based on Research Contributions By…

- Berhan Oumer Adame
- Markus Anders
- Jeremias Berg
- **Bart Bogaerts**
- **Benjamin Bogø**
- Wolf De Wulf
- Emir Demirović
- Simon Dold
- Jan Elffers
- Ambros Gleixner
- **Stephan Gocht**
- Arthur Gontier
- Malte Helmert
- Alexander Hoen
- Hannes Ihalainen
- Matti Järvisalo
- George Katsirelos
- **Wietze Koops**
- Daniel Le Berre
- Ruben Martins
- Ross McBride
- **Ciaran McCreesh**
- **Matthew McIlree**
- **Magnus O. Myreen**
- **Andy Oertel**
- Tobias Paxian
- Patrick Prosser
- Adrián Rebola-Pardo
- **Philip Rodgers**
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- **Yong Kiam Tan**
- James Trimble
- Mark Turnbull
- Dieter Vandesande
- Marc Vinyals
- **Wei-Lin Wu**

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, BMN22, GCS23]
- Solvers can propose infeasible "solutions" (but such errors can be detected)
- More challenging: How to achieve reliable claims of infeasibility?
- Or that a solution is optimal? (Even off-by-one mistakes can snowball into large errors if solver used as subroutine)

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

What Can Be Done About Solver Bugs?

- **Software testing**
  Very useful, but bugs slip through even with careful domain-specific testing
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence

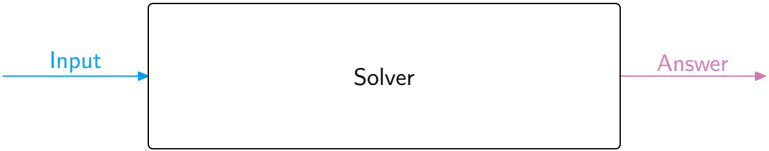
What Can Be Done About Solver Bugs?

- **Software testing**
  Very useful, but bugs slip through even with careful domain-specific testing
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence
- **Formal methods**
  Prove that solver implementation adheres to formal specification
  Current formal verification techniques cannot scale to level of complexity in modern solvers
  (Despite valiant efforts in, e.g., [Fle20])

What Can Be Done About Solver Bugs?

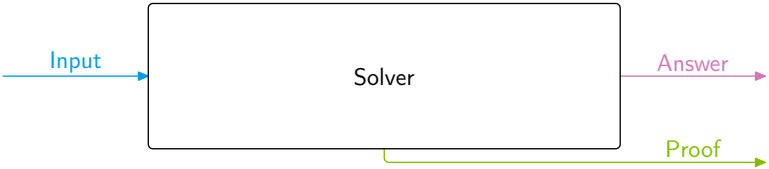
- **Software testing**
  Very useful, but bugs slip through even with careful domain-specific testing
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence
- **Formal methods**
  Prove that solver implementation adheres to formal specification
  Current formal verification techniques cannot scale to level of complexity in modern solvers
  (Despite valiant efforts in, e.g., [Fle20])
- **Proof logging**
  Make solver certifying [ABM+11, MMNS11] by adding code so that it outputs
  1. not only answer but also
  2. simple, machine-verifiable proof that answer is correct

Proof Logging with Certifying Solvers: Workflow



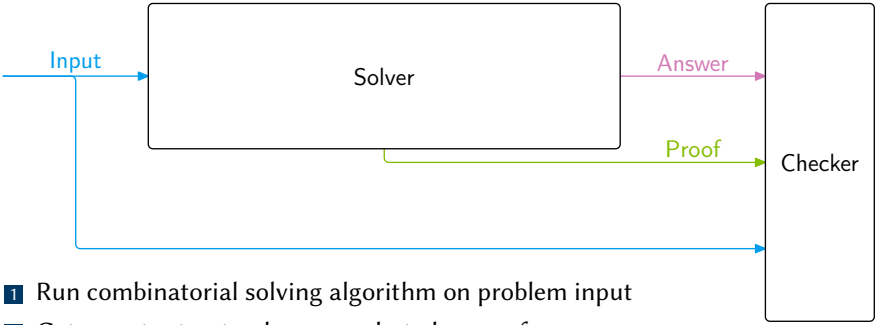
1. Run combinatorial solving algorithm on problem input

Proof Logging with Certifying Solvers: Workflow



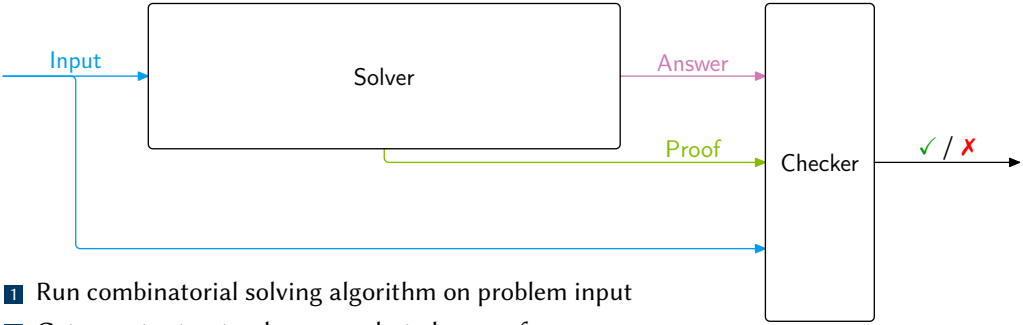
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

Proof Logging with Certifying Solvers: Workflow



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

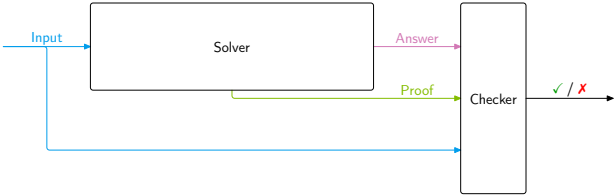
Proof Logging with Certifying Solvers: Workflow



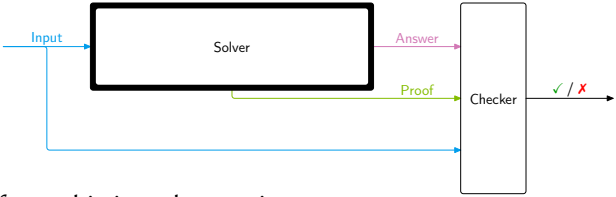
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

Proof Logging Desiderata

Proof format for certifying solver should be



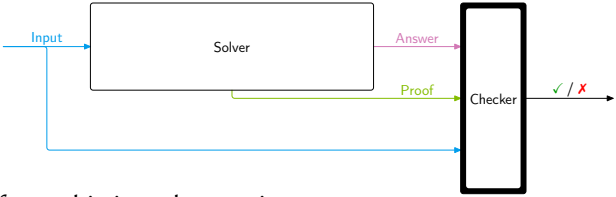
Proof Logging Desiderata



Proof format for certifying solver should be

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

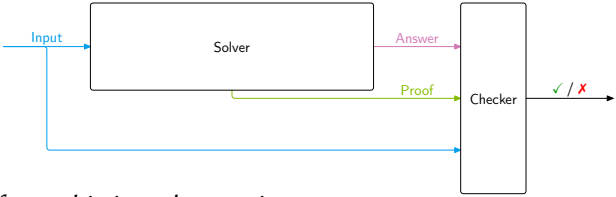
Proof Logging Desiderata



Proof format for certifying solver should be

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

Proof Logging Desiderata

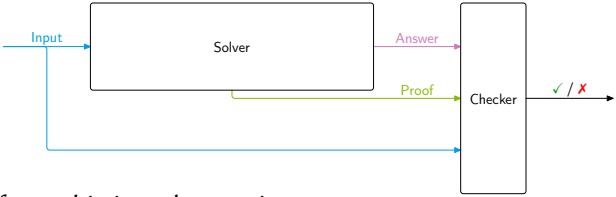


Proof format for certifying solver should be

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

Clear conflict expressivity vs. simplicity!

Proof Logging Desiderata



Proof format for certifying solver should be

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

Clear conflict expressivity vs. simplicity!

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

Some Previous Proof Logging Work

Boolean satisfiability (SAT) solving

- Well established since over decade with several proof formats such as DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], …
- But no efficient support for most advanced techniques such as
  - Gaussian elimination
  - symmetry breaking
- And only for decision problems, not optimization (MaxSAT)

Some Previous Proof Logging Work

Boolean satisfiability (SAT) solving

- Well established since over decade with several proof formats such as DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], …
- But no efficient support for most advanced techniques such as
  - Gaussian elimination
  - symmetry breaking
- And only for decision problems, not optimization (MaxSAT)

Constraint programming

- Either have to trust that propagations done correctly [DFS12, OSC09, VS10]
- Or suffer from exponential slow-down to generate verifiable proofs [GCS23]

Some Previous Proof Logging Work

Boolean satisfiability (SAT) solving

- Well established since over decade with several proof formats such as DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], …
- But no efficient support for most advanced techniques such as
  - Gaussian elimination
  - symmetry breaking
- And only for decision problems, not optimization (MaxSAT)

Constraint programming

- Either have to trust that propagations done correctly [DFS12, OSC09, VS10]
- Or suffer from exponential slow-down to generate verifiable proofs [GCS23]

Mixed integer linear programming

- Work on proof format VIPR [CGS17, EG23]
- But only for exact solving and without support for advanced techniques

Take-Away Message from This Tutorial

Proof logging for sophisticated combinatorial solvers is possible!

Take-Away Message from This Tutorial

Proof logging for sophisticated combinatorial solvers is possible!

- With single, unified method!
- Producing proofs in simple format calculating with 0–1 integer linear inequalities (a.k.a. pseudo-Boolean constraints)
- Implemented in VeriPB (`https://VeriPB.org`)

Take-Away Message from This Tutorial

Proof logging for sophisticated combinatorial solvers is possible!

- With single, unified method!
- Producing proofs in simple format calculating with 0–1 integer linear inequalities (a.k.a. pseudo-Boolean constraints)
- Implemented in VeriPB (`https://VeriPB.org`)

Purpose of this presentation:

1. Marketing pitch ☺

Take-Away Message from This Tutorial

Proof logging for sophisticated combinatorial solvers is possible!

- With single, unified method!
- Producing proofs in simple format calculating with 0–1 integer linear inequalities (a.k.a. pseudo-Boolean constraints)
- Implemented in VeriPB (`https://VeriPB.org`)

Purpose of this presentation:

1. Marketing pitch ☺
2. Explain pseudo-Boolean proof logging basics
   (to prepare for Ciaran McCreesh's talk on CP proof logging tomorrow)

Take-Away Message from This Tutorial

Proof logging for sophisticated combinatorial solvers is possible!

- With single, unified method!
- Producing proofs in simple format calculating with 0–1 integer linear inequalities (a.k.a. pseudo-Boolean constraints)
- Implemented in VeriPB (`https://VeriPB.org`)

Purpose of this presentation:

1. Marketing pitch ☺
2. Explain pseudo-Boolean proof logging basics
   (to prepare for Ciaran McCreesh's talk on CP proof logging tomorrow)
3. Provide pointers for further reading

Take-Away Message from This Tutorial

Proof logging for sophisticated combinatorial solvers is possible!

- With single, unified method!
- Producing proofs in simple format calculating with 0–1 integer linear inequalities (a.k.a. pseudo-Boolean constraints)
- Implemented in VeriPB (`https://VeriPB.org`)

Purpose of this presentation:

1. Marketing pitch ☺
2. Explain pseudo-Boolean proof logging basics
   (to prepare for Ciaran McCreesh's talk on CP proof logging tomorrow)
3. Provide pointers for further reading

These slides with references are online at `https://jakobnordstrom.se/presentations/`

Software for Exercises Tomorrow Thursday

Kindling: See `README.md` and files in `slides-proofs/`

https://github.com/ciaranm/kindling

VeriPB (official version): See `README.md` and try to run `veripb justified.opb justified.pbp`

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

VeriPB (experimental): Interactive mode (Linux and Mac only?); see `README.md` and then use `cargo run -p veripb-repl -m path-to-veripb-repl/Cargo.toml`

https://github.com/ciaranm/veripb-repl

The Sales Pitch For Proof Logging

1. Certifies correctness of computed results

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

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 software development [GMM+20, KM21, BBN+23, EG23, KLM+25, DKK+26]

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 software development [GMM+20, KM21, BBN+23, EG23, KLM+25, DKK+26]
4. Facilitates performance analysis

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 software development [GMM+20, KM21, BBN+23, EG23, KLM+25, DKK+26]
4. Facilitates performance analysis
5. Helps identify potential for further improvements

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 software development [GMM+20, KM21, BBN+23, EG23, KLM+25, DKK+26]
4. Facilitates performance analysis
5. Helps identify potential for further improvements
6. Enables auditability

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 software development [GMM+20, KM21, BBN+23, EG23, KLM+25, DKK+26]
4. Facilitates performance analysis
5. Helps identify potential for further improvements
6. Enables auditability
7. Serves as stepping stone towards explainability

Design Principles for Proof Logging

Proof logging implementation

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

Design Principles for Proof Logging

Proof logging implementation

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

Performance goals

- Proof logging overhead small constant fraction of running time ($\lessapprox 10\%$)
- Proof checking time within constant factor of solving time (current aim $\lessapprox \times 10$)

Design Principles for Proof Logging

Proof logging implementation

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

Performance goals

- Proof logging overhead small constant fraction of running time ($\lessapprox 10\%$)
- Proof checking time within constant factor of solving time (current aim $\lessapprox \times 10$)

Proof system

- Keep language simple — no XOR constraints, CP propagators, symmetry groups, …
- But reason efficiently about such notions using power of proof system
- Combine proof logging with formally verified proof checker

But Let Us Start from the Beginning…

Review of some basic concepts:

- Satisfiability (SAT) problem
- Unit propagation
- DPLL and CDCL algorithms
- Resolution proof system

The Satisfiability (SAT) Problem

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

The SAT Problem

Given a CNF formula F, is it satisfiable?

For instance, what about:

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

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

Proof checker should know how to unit propagate until saturation

Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

Davis-Putman-Logemann-Loveland (DPLL)

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

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

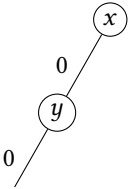
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

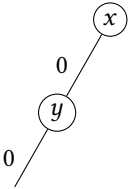


Davis-Putman-Logemann-Loveland (DPLL)

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

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

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



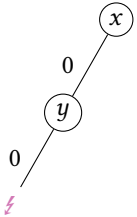
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

1 $x \vee y$



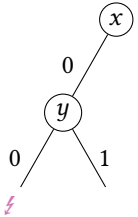
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

1 $x \vee y$



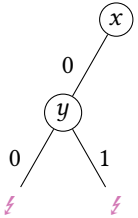
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



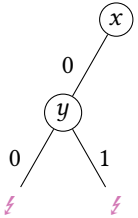
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



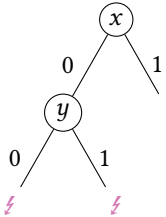
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



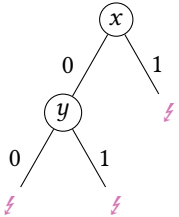
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



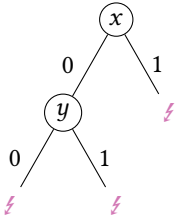
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



Reverse Unit Propagation (RUP)

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

Reverse Unit Propagation (RUP)

To make this into 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 into 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 proof consisting of RUP clauses

What About Conflict-Driven Clause Learning (CDCL)?

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

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

What About Conflict-Driven Clause Learning (CDCL)?

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

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

Decision

Free choice to assign value to variable

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

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

Decision

Free choice to assign value to variable

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

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

decision level 3

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

Conflict Analysis

Time to analyse this conflict and learn from it!

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

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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

decision level 3

Conflict Analysis

Time to analyse this conflict and learn from it!

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

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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

$\bot$

decision level 3

Could backtrack by erasing conflict level & flipping last decision

Conflict Analysis

Time to analyse this conflict and learn from it!

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

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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

$\bot$

decision level 3

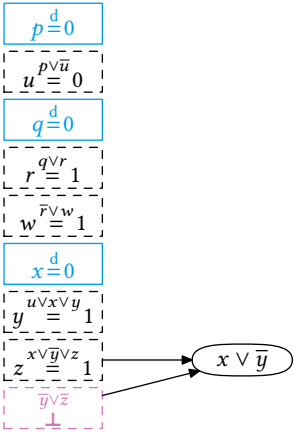
Could backtrack by erasing conflict level & flipping last decision

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!

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



Could backtrack by erasing conflict level & flipping last decision

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

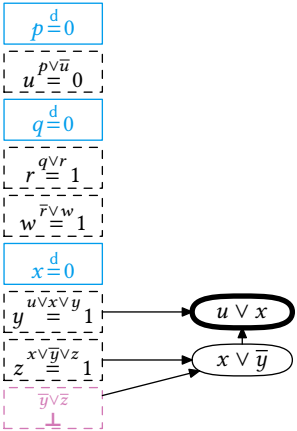
Case analysis over z for last two clauses:

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

Conflict Analysis

Time to analyse this conflict and learn from it!

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



Could backtrack by erasing conflict level & flipping last decision

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

Case analysis over z for last two clauses:

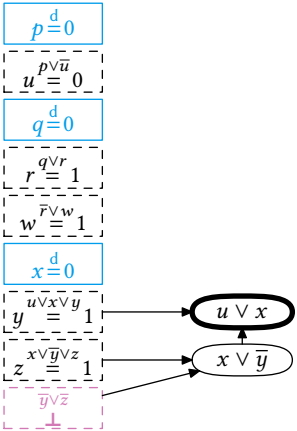
- $x \vee \overline{y} \vee z$ wants $z = 1$
- $\overline{y} \vee \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \vee \overline{y}$

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

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

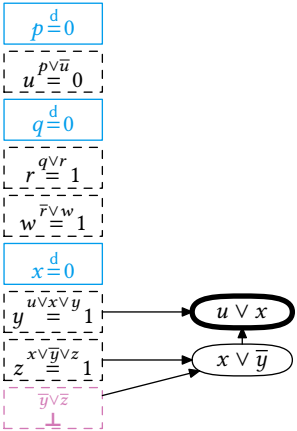
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

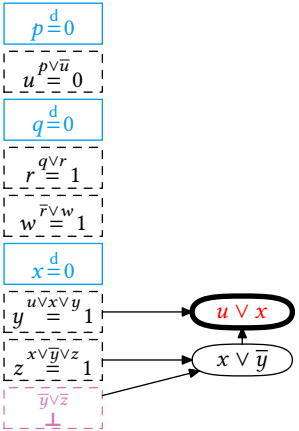


Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



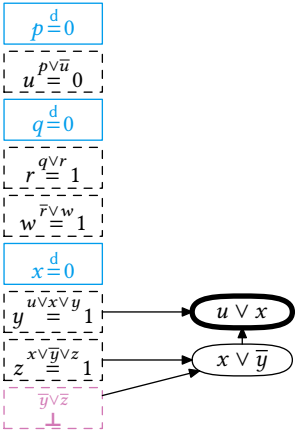
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

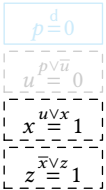
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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





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

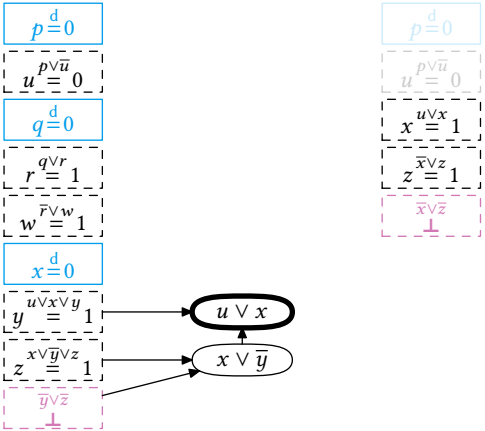
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Then continue as before…

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

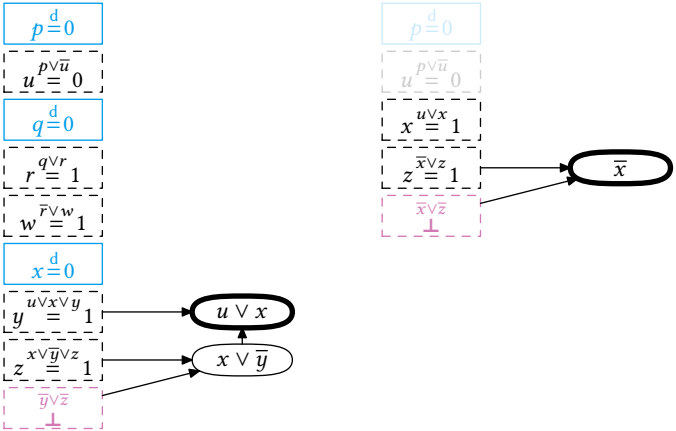
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

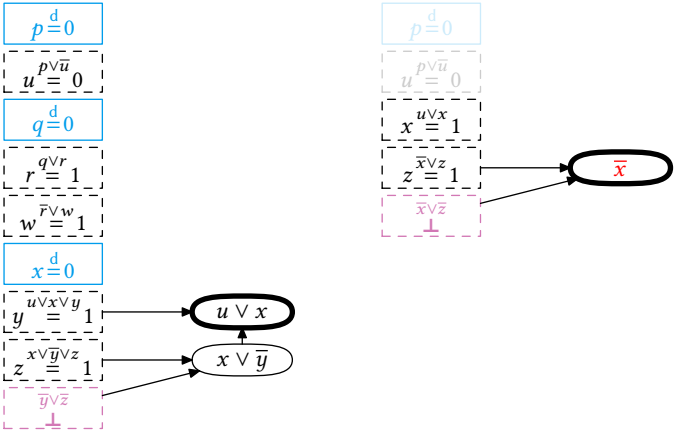
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

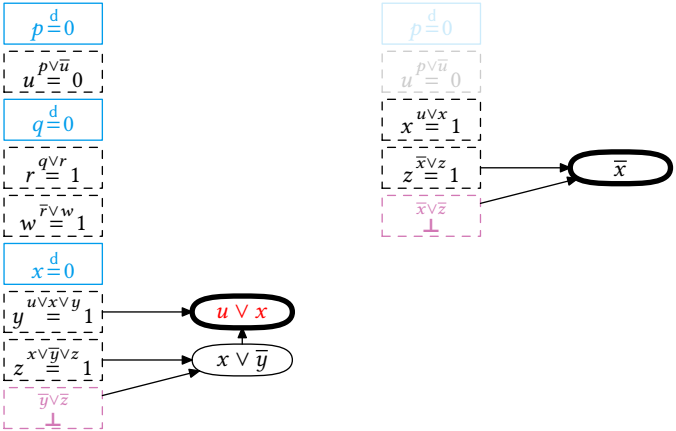
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

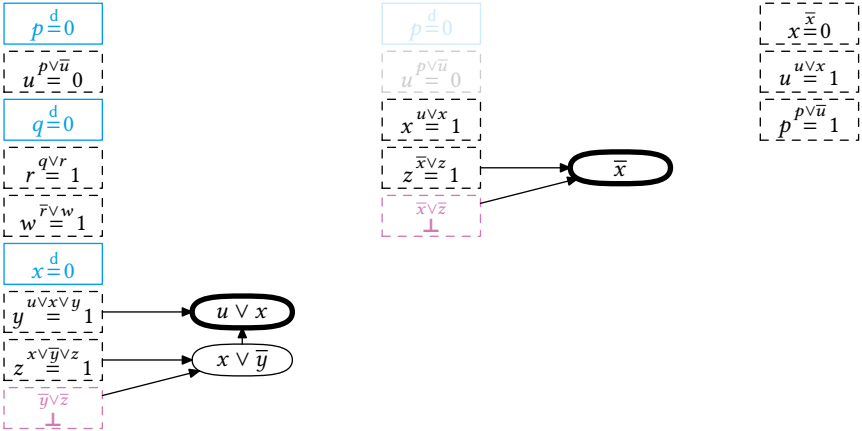
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

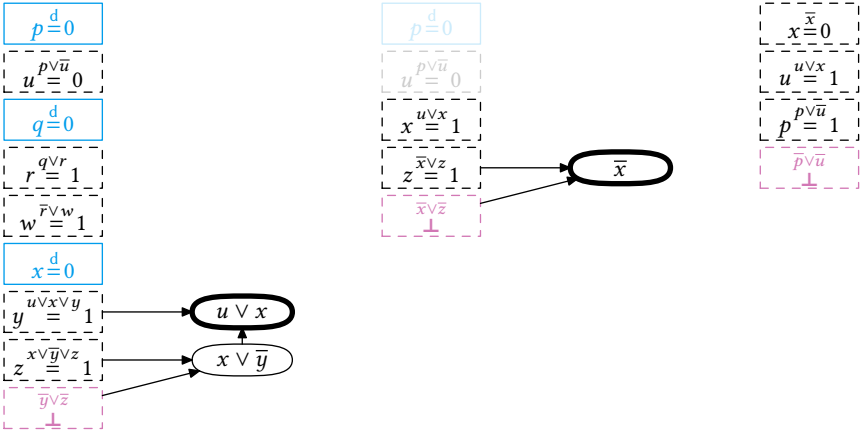
$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

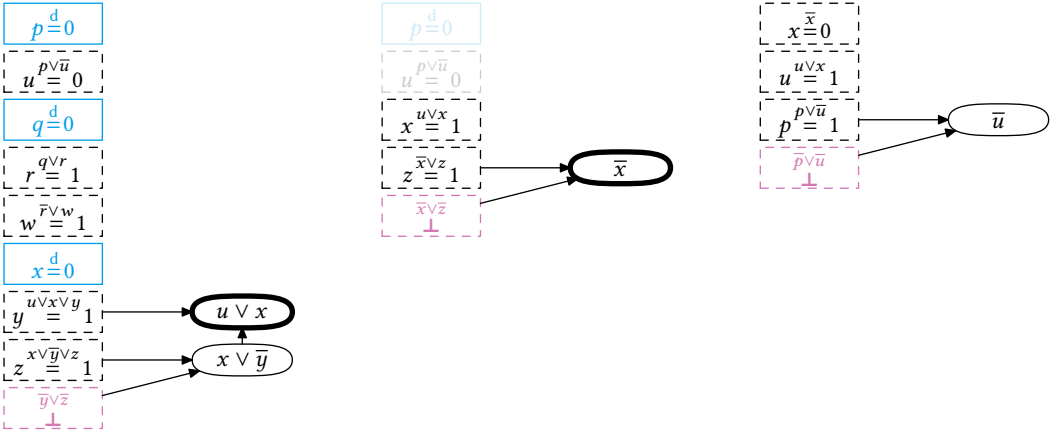
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

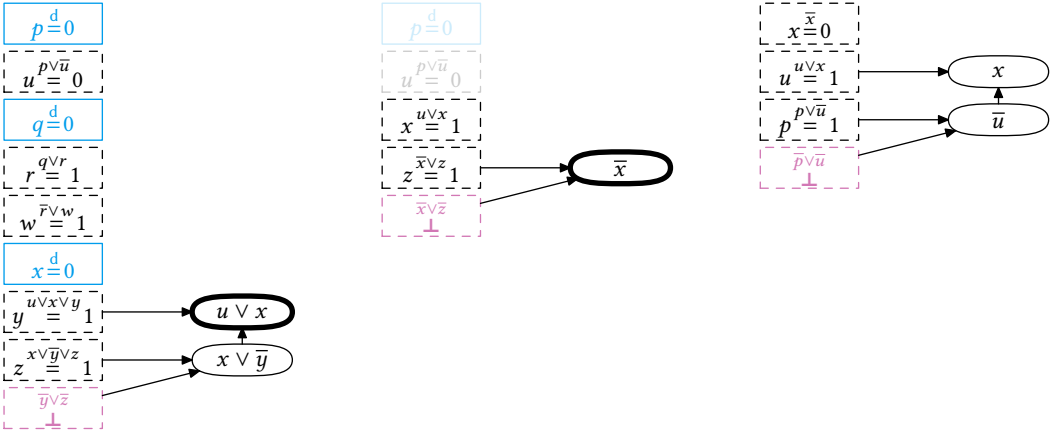
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

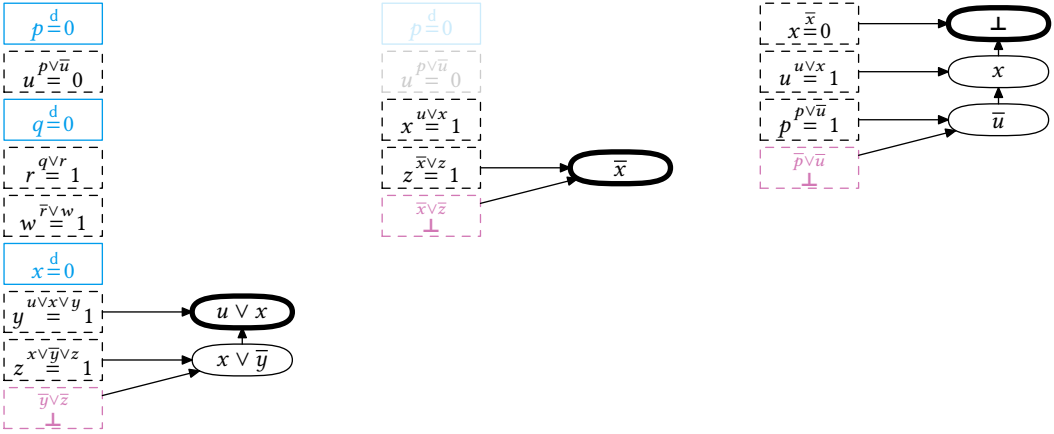
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



CDCL Reasoning and the Resolution Proof System

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

CDCL Reasoning and the Resolution Proof System

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

Resolution proof system [Bla37, Rob65]

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

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

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

CDCL Reasoning and the Resolution Proof System

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

Resolution proof system [Bla37, Rob65]

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

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

- 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

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

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

When run on unsatisfiable formula, CDCL generates resolution proof*

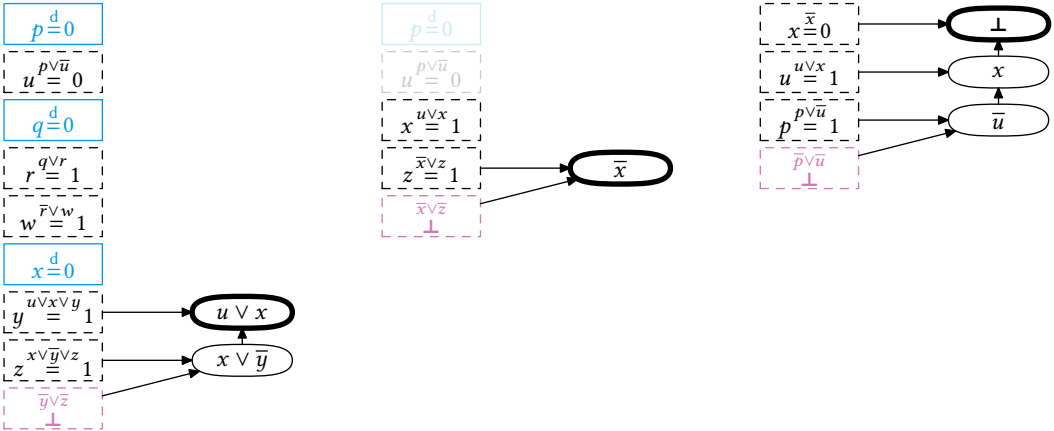
(*) Ignores pre- and inprocessing, but we will get there…

Resolution Proofs from CDCL Executions

Obtain resolution proof…

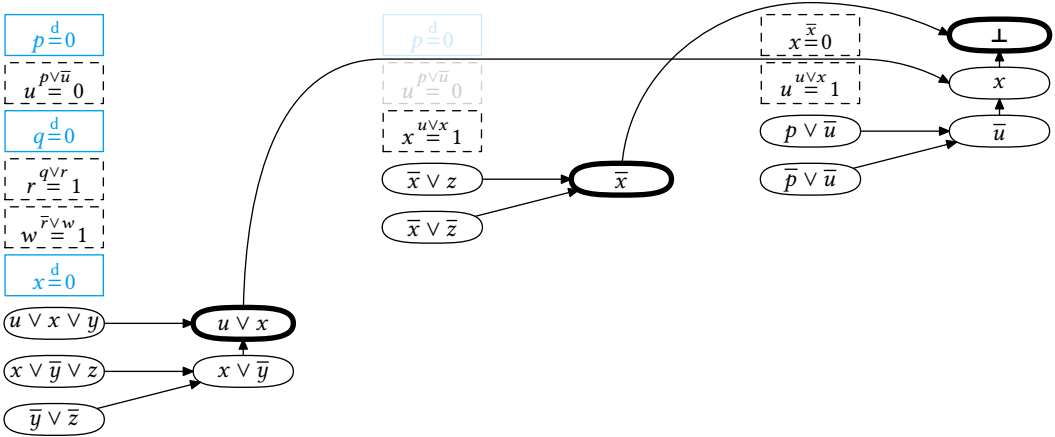
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution…



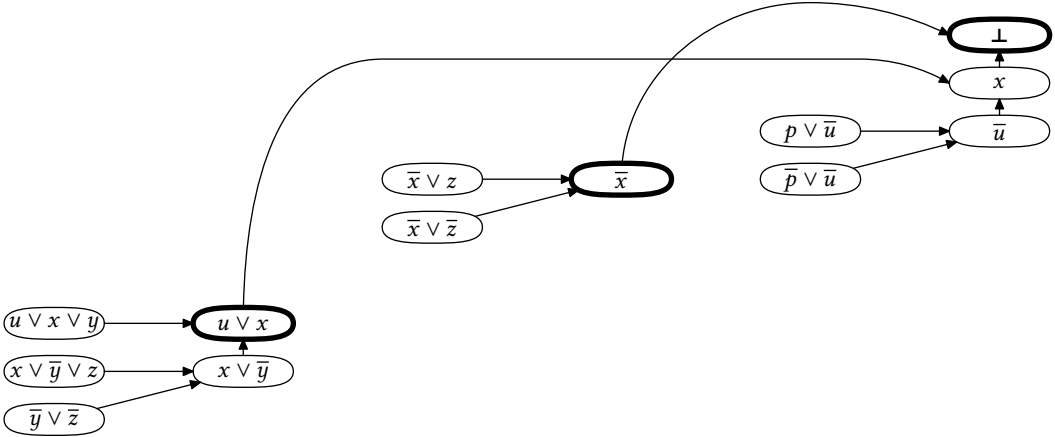
Resolution Proofs from CDCL Executions

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

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 proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

More Ingredients in Proof Logging for SAT

Fact

RUP proofs can be viewed as shorthand for resolution proofs

See proof complexity and SAT solving survey [BN21] for more on this

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

Extension Variables, Part 1

Suppose we want a reified variable r encoding

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

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

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

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

Extension Variables, Part 1

Suppose we want a reified variable r encoding

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

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

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

for fresh variable r (this is fine since r 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 [KRH18]

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!

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

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)

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

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

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
- And more…

Pseudo-Boolean Constraints

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

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

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

Pseudo-Boolean Constraints

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

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

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

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

Some Types of Pseudo-Boolean Constraints

1. Clauses

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

2. Cardinality constraints

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

3. General pseudo-Boolean constraints

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms From the input

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms From the input

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms From the input

Literal axioms

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

Addition

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms From the input

Literal axioms

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

Addition

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

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

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms From the input

Literal axioms

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

Addition

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

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

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

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

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

Input/model axioms — From the input

Literal axioms

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

Addition

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

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

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

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

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

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

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \min(a_i, A) \cdot \ell_i \geq A}$$

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

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

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

Cutting Planes Toy Example

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

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

$$\text{Constraint 1} \doteq 2x + y + w \geq 2$$

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

$$\sim\text{z} \doteq \overline{z} \geq 0$$

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

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

Cutting Planes Toy Example

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

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

$$\text{Constraint 1} \doteq 2x + y + w \geq 2$$

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

$$\sim\text{z} \doteq \overline{z} \geq 0$$

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

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

(*) Except VeriPB variables have to be at least two characters long, but we abuse syntax slightly here for simplicity

Resolution and Cutting Planes

To simulate resolution step such as

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

we can perform the cutting planes steps

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

Resolution and Cutting Planes

To simulate resolution step such as

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

we can perform the cutting planes steps

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

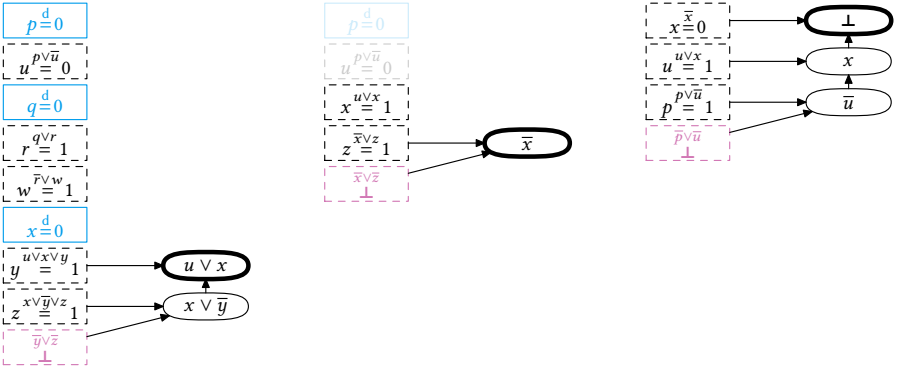
Given that the premises are clauses 7 and 5 in our example CNF formula, using references

$$\text{Constraint 7} \doteq \overline{y} + \overline{z} \geq 1$$
$$\text{Constraint 5} \doteq x + \overline{y} + z \geq 1$$

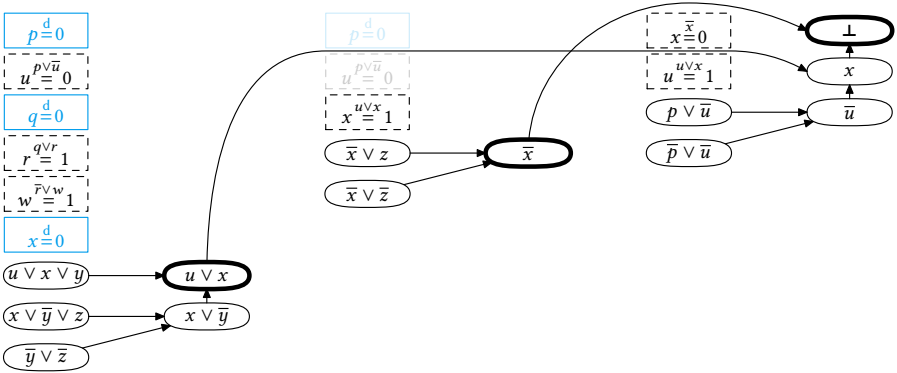
we can write this in the proof log as

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

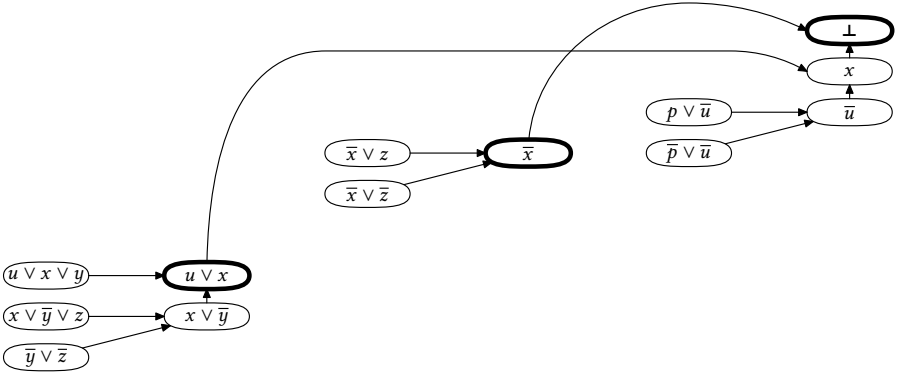
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



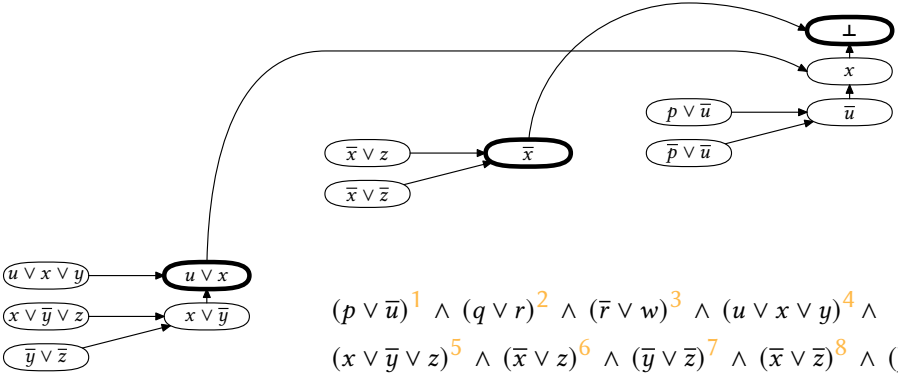
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses

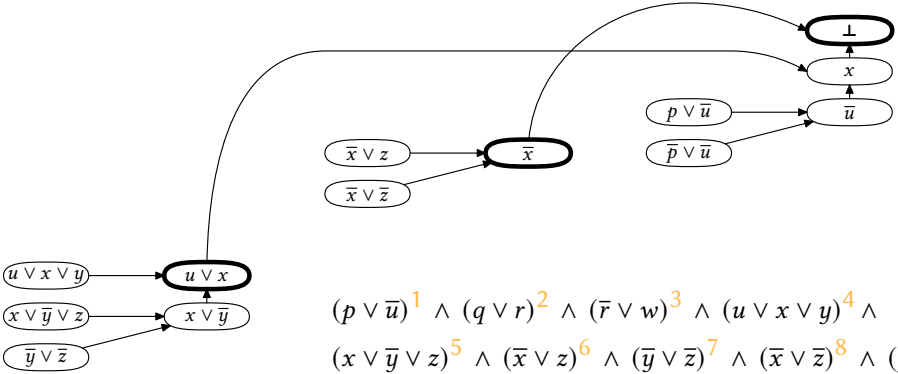


Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



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

Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



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

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

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

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

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

RUP Revisited

Can define (reverse) unit propagation in a pseudo-Boolean setting [EGMN20]

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

E.g., if x_5 is false,

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

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

RUP Revisited

Can define (reverse) unit propagation in a pseudo-Boolean setting [EGMN20]

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

E.g., if x_5 is false,

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

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

Risk for confusion!

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

Propagation, Conflict, and Slack

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

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

Propagation, Conflict, and Slack

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

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

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

ρ	$slack(C; \rho)$	comment

Propagation, Conflict, and Slack

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

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

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

ρ	$slack(C;\rho)$	comment
$\{\}$	8	

Propagation, Conflict, and Slack

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

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

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

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

Propagation, Conflict, and Slack

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

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

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

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

Propagation, Conflict, and Slack

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

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

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

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

Propagation, Conflict, and Slack

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

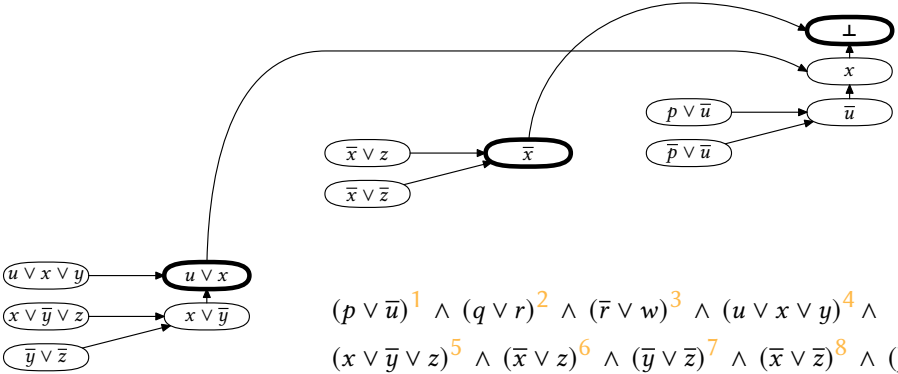
$$slack\left(\sum_i a_i \ell_i \geq A; \rho\right) = \sum_{i\,:\,\rho(\ell_i) \neq 0} a_i \; - A$$

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

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

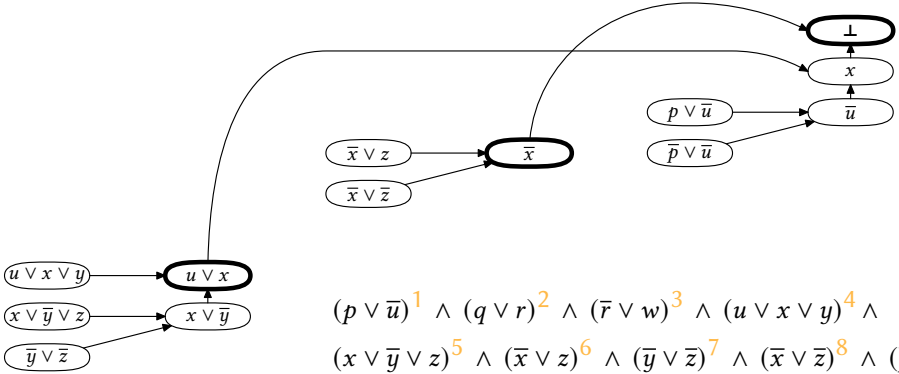
Note: constraint can be conflicting though not all variables assigned

Pseudo-Boolean Proof Logging for Example CDCL Execution with RUP



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

Pseudo-Boolean Proof Logging for Example CDCL Execution with RUP



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

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

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

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

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

Extension Variables, Part 2

Suppose we want new, fresh reified variable r encoding

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

This time, introduce constraints

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

Again, needs support from the proof system in the form of strengthening rules

Proof Logs for "Cutting Planes with Strengthening"

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 "Cutting Planes with Strengthening"

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 — more details in a few slides …

Optimisation and Enumeration Problems

Optimisation

- Define an objective function $f = \sum_i w_i \ell_i + k$, $w_i, k \in \mathbb{Z}$, to minimise subject to the contraints in the formula
- To maximise, negate objective
- Log solution α with `soli` rule $\Rightarrow$ objective function value $UB = \sum_i w_i \alpha(\ell_i) + k$
- Add objective-improving constraint $f \leq UB - 1$
- Semantics for proof of optimality: "infeasible to find better solution than best so far"
- Can also derive and certify (potentially non-tight) lower bound $f \geq LB$

Optimisation and Enumeration Problems

Optimisation

- Define an objective function $f = \sum_i w_i \ell_i + k$, $w_i, k \in \mathbb{Z}$, to minimise subject to the contraints in the formula
- To maximise, negate objective
- Log solution α with `soli` rule $\Rightarrow$ objective function value $UB = \sum_i w_i \alpha(\ell_i) + k$
- Add objective-improving constraint $f \leq UB - 1$
- Semantics for proof of optimality: "infeasible to find better solution than best so far"
- Can also derive and certify (potentially non-tight) lower bound $f \geq LB$

Enumeration

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

Deleting Constraints

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

Unsatisfiability proofs: fairly straightforward to deal with from point of view of proof logging

Optimisation and enumeration proofs: significantly more delicate

We will mostly ignore deletions during this tutorial for simplicity and clarity

Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging [basically: add print statements to solver code]

Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging [basically: add print statements to solver code]

Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging [basically: add print statements to solver code]

Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

Goldilocks compromise between expressivity and simplicity:

1. 0-1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments
3. Efficient reification using big-M constraints

Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging [basically: add print statements to solver code]

Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

Goldilocks compromise between expressivity and simplicity:

1. 0-1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments
3. Efficient reification using big-M constraints — another example to get used to this:

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

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

Pseudo-Boolean Proof Logging — How and Why?

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

- just do proof logging [basically: add print statements to solver code]

Otherwise

- do trusted or verified translation to 0-1 ILP
- do proof logging for 0-1 ILP formulation [but solver still works with original input]

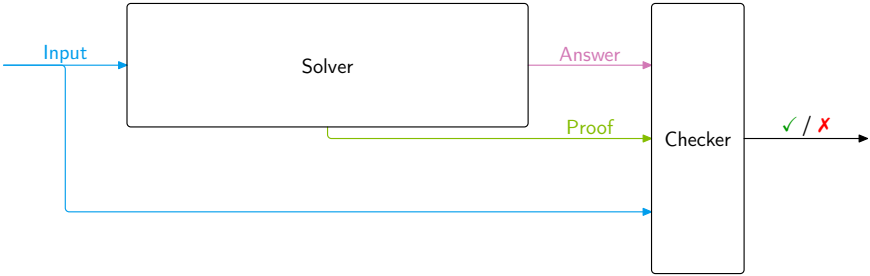
Goldilocks compromise between expressivity and simplicity:

1. 0-1 ILP expressive formalism for combinatorial problems (including objective)
2. Powerful reasoning capturing many combinatorial arguments
3. Efficient reification using big-M constraints — another example to get used to this:

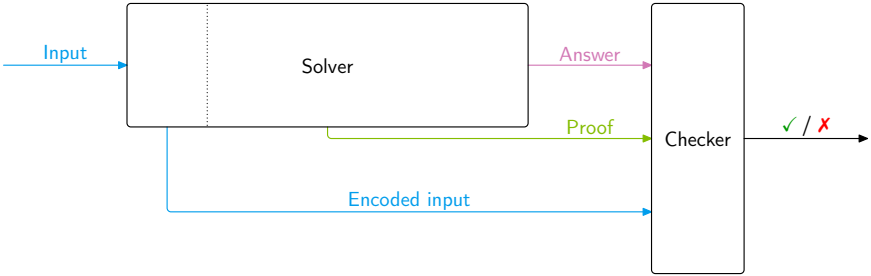
$$r \Rightarrow x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7 \qquad 7\overline{r} + x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

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

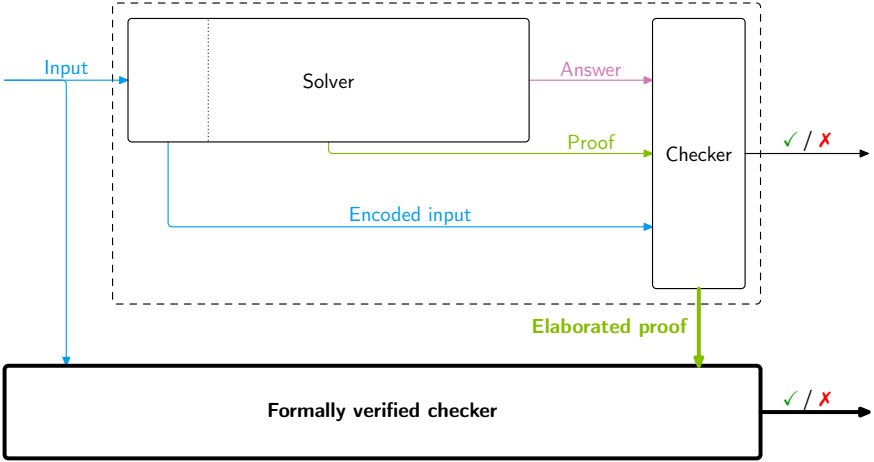
Proof Logging with Formally Verified Checking: Full Workflow



Proof Logging with Formally Verified Checking: Full Workflow



Proof Logging with Formally Verified Checking: Full Workflow



Strengthening Rules (And the Truth About Extension Variables)

When is it allowed to derive a new constraint? If it is (clear that it is) implied?

Strengthening Rules (And the 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 r encoding

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

we introduced pseudo-Boolean constraints

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

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

Strengthening Rules (And the 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 r encoding

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

we introduced pseudo-Boolean constraints

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

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

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

Redundance-Based Strengthening

C is "redundant" with respect to F if F and $F \cup \{C\}$ are equisatisfiable

[apologies for the terminology — this is inherited from SAT proof logging]

Redundance-Based Strengthening

C is "redundant" with respect to F if F and $F \cup \{C\}$ are equisatisfiable

[apologies for the terminology — this is inherited from SAT proof logging]

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

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

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

Redundance-Based Strengthening

C is "redundant" with respect to F if F and $F \cup \{C\}$ are equisatisfiable

[apologies for the terminology — this is inherited from SAT proof logging]

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

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

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

Proof sketch for interesting direction:

- If α satisfies F but falsifies C, then α satisfies $(F \cup \{C\})\restriction_\omega$ *[notation $C\restriction_\omega$ means apply ω to C and simplify]*

Redundance-Based Strengthening

C is "redundant" with respect to F if F and $F \cup \{C\}$ are equisatisfiable

[apologies for the terminology — this is inherited from SAT proof logging]

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

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

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

Proof sketch for interesting direction:

- If α satisfies F but falsifies C, then α satisfies $(F \cup \{C\})\restriction_\omega$ *[notation $C\restriction_\omega$ means apply ω to C and simplify]*
- I.e., $\alpha \circ \omega$ satisfies $F \cup \{C\}$

Redundance-Based Strengthening

C is "redundant" with respect to F if F and $F \cup \{C\}$ are equisatisfiable

[apologies for the terminology — this is inherited from SAT proof logging]

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

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

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

Proof sketch for interesting direction:

- If α satisfies F but falsifies C, then α satisfies $(F \cup \{C\})\restriction_\omega$ *[notation $C\restriction_\omega$ means apply ω to C and simplify]*
- I.e., $\alpha \circ \omega$ satisfies $F \cup \{C\}$

Witness ω should be specified, and implication needs to be efficiently verifiable

Redundance-Based Strengthening

C is "redundant" with respect to F if F and $F \cup \{C\}$ are equisatisfiable

[apologies for the terminology — this is inherited from SAT proof logging]

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

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

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

Proof sketch for interesting direction:

- If α satisfies F but falsifies C, then α satisfies $(F \cup \{C\})\restriction_\omega$ *[notation $C\restriction_\omega$ means apply ω to C and simplify]*
- I.e., $\alpha \circ \omega$ satisfies $F \cup \{C\}$

Witness ω should be specified, and implication needs to be efficiently verifiable — every $D \in (F \cup \{C\})\restriction_\omega$ should follow from $F \cup \{\neg C\}$ either

1. "obviously" (e.g., by reverse unit propagation) or

Redundance-Based Strengthening

C is "redundant" with respect to F if F and $F \cup \{C\}$ are equisatisfiable

[apologies for the terminology — this is inherited from SAT proof logging]

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

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

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

Proof sketch for interesting direction:

- If α satisfies F but falsifies C, then α satisfies $(F \cup \{C\})\restriction_\omega$ *[notation $C\restriction_\omega$ means apply ω to C and simplify]*
- I.e., $\alpha \circ \omega$ satisfies $F \cup \{C\}$

Witness ω should be specified, and implication needs to be efficiently verifiable — every $D \in (F \cup \{C\})\restriction_\omega$ should follow from $F \cup \{\neg C\}$ either

1. "obviously" (e.g., by reverse unit propagation) or

Deriving $r \Leftrightarrow (3x + 2y + z + w \geq 3)$ Using the Redundance Rule

Want to derive

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

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

Deriving $r \Leftrightarrow (3x + 2y + z + w \geq 3)$ Using the Redundance Rule

Want to derive

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

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

1. $F \cup \{\neg(3\overline{r} + 3x + 2y + z + w \geq 3)\} \models \big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\}\big)\restriction_\omega$

Deriving $r \Leftrightarrow (3x + 2y + z + w \geq 3)$ Using the Redundance Rule

Want to derive

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

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

1. $F \cup \{\neg(3\overline{r} + 3x + 2y + z + w \geq 3)\} \models \big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\}\big)\restriction_\omega$
   Choose $\omega = \{r \mapsto 0\}$ — F untouched; new constraint satisfied

Deriving $r \Leftrightarrow (3x + 2y + z + w \geq 3)$ Using the Redundance Rule

Want to derive

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

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

1. $F \cup \{\neg(3\overline{r} + 3x + 2y + z + w \geq 3)\} \models \big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\}\big)\restriction_\omega$
   Choose $\omega = \{r \mapsto 0\}$ — F untouched; new constraint satisfied
2. $F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\} \cup \{\neg(5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5)\} \models$
   $\big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3,\ 5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5\}\big)\restriction_\omega$
   Choose $\omega = \{r \mapsto 1\}$ — F untouched; new constraint satisfied

Deriving $r \Leftrightarrow (3x + 2y + z + w \geq 3)$ Using the Redundance Rule

Want to derive

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

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

1. $F \cup \{\neg(3\overline{r} + 3x + 2y + z + w \geq 3)\} \models \big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\}\big)\restriction_\omega$
   Choose $\omega = \{r \mapsto 0\}$ — F untouched; new constraint satisfied
2. $F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\} \cup \{\neg(5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5)\} \models$
   $\big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3,\ 5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5\}\big)\restriction_\omega$
   Choose $\omega = \{r \mapsto 1\}$ — F untouched; new constraint satisfied
   $\neg(5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5)$ forces $3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \leq 4$

Deriving $r \Leftrightarrow (3x + 2y + z + w \geq 3)$ Using the Redundance Rule

Want to derive

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

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

1. $F \cup \{\neg(3\overline{r} + 3x + 2y + z + w \geq 3)\} \models \big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\}\big)\restriction_\omega$
   Choose $\omega = \{r \mapsto 0\}$ — F untouched; new constraint satisfied
2. $F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\} \cup \{\neg(5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5)\} \models$
   $\big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3,\ 5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5\}\big)\restriction_\omega$
   Choose $\omega = \{r \mapsto 1\}$ — F untouched; new constraint satisfied
   $\neg(5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5)$ forces $3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \leq 4$
   Same constraint as $3x + 2y + z + w \geq 3$; can be automatically detected

Deriving $r \Leftrightarrow (3x + 2y + z + w \geq 3)$ Using the Redundance Rule

Want to derive

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

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

1. $F \cup \{\neg(3\overline{r} + 3x + 2y + z + w \geq 3)\} \models \big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\}\big)\restriction_\omega$
   Choose $\omega = \{r \mapsto 0\}$ — F untouched; new constraint satisfied
2. $F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3\} \cup \{\neg(5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5)\} \models$
   $\big(F \cup \{3\overline{r} + 3x + 2y + z + w \geq 3,\ 5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5\}\big)\restriction_\omega$
   Choose $\omega = \{r \mapsto 1\}$ — F untouched; new constraint satisfied
   $\neg(5r + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5)$ forces $3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \leq 4$
   Same constraint as $3x + 2y + z + w \geq 3$; can be automatically detected

VeriPB code:

```
red 3 ~r 3 x 2 y 1 z 1 w >= 3 : r -> 0 ;
red 5 r 3 ~x 2 ~y 1 ~z 1 ~w >= 5 : r -> 1 ;
```

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

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

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

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

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

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

Can be more aggressive if witness ω strictly improves solution

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

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

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

Can be more aggressive if witness ω strictly improves solution

Dominance-based strengthening (simplified)

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

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

Redundance and Dominance Rules for Optimisation

Redundance-based strengthening, optimisation version

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

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

Can be more aggressive if witness ω strictly improves solution

Dominance-based strengthening (simplified)

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

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

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

Why is this sound?

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

Why is this sound?

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

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

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

$$F \cup \{C_1, \ldots, C_{m-1}\} \cup \{\neg C_m\} \models F\restriction_\omega \cup \{f\restriction_\omega < f\}$$

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

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

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

$$F \cup \{C_1, \ldots, C_{m-1}\} \cup \{\neg C_m\} \models F\restriction_\omega \cup \{f\restriction_\omega < f\}$$

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

Now why is *this* sound?

- Same inductive proof as before, but nested

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

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

$$F \cup \{C_1, \ldots, C_{m-1}\} \cup \{\neg C_m\} \models F\restriction_\omega \cup \{f\restriction_\omega < f\}$$

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

Now why is *this* sound?

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

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified)

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

$$F \cup \{C_1, \ldots, C_{m-1}\} \cup \{\neg C_m\} \models F\restriction_\omega \cup \{f\restriction_\omega < f\}$$

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

Now why is *this* sound?

- Same inductive proof as before, but nested
- Or pick solution α satisfying F and 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, ABB+26b] fore more details

Strengthening Rules: Syntax

```
red ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;

dom ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;
```

Strengthening Rules: Syntax

```
red ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;

dom ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;
```

- Witness ω should be explicitly specified in proof log

Strengthening Rules: Syntax

```
red ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;

dom ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;
```

- Witness ω should be explicitly specified in proof log
- Subproofs of proof goals should also be explicit

Strengthening Rules: Syntax

```
red ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;

dom ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;
```

- Witness ω should be explicitly specified in proof log
- Subproofs of proof goals should also be explicit
- Except can be skipped for proof goals that "obviously" follow, e.g.,
  - by reverse unit propagation (RUP)
  - by simple syntactic implication from other constraint (as in our reification example)
  - if the proof goal is not affected by the witness substitution ω

Successful Applications of Pseudo-Boolean Proof Logging with VeriPB

Surprisingly, pseudo-Boolean reasoning with strengthening rules sufficient to efficiently certify wide range of combinatorial solving techniques and paradigms:

Successful Applications of Pseudo-Boolean Proof Logging with VeriPB

Surprisingly, pseudo-Boolean reasoning with strengthening rules sufficient to efficiently certify wide range of combinatorial solving techniques and paradigms:

1. Boolean satisfiability (SAT) solving including advanced techniques such as
   - Gaussian elimination [GN21]
   - symmetry breaking [BGMN23, ABB+26b]
2. SAT-based optimization (MaxSAT) [VDB22, BBN+23, BBN+24, IOT+24, VCB26]
3. Pseudo-Boolean solving and optimization [GMNO22, KLM+25]
4. Graph solving [GMN20, GMM+20, GMM+24, DKK+26]
5. Dynamic programming and decision diagrams [DMM+24]
6. Presolving in 0–1 integer linear programming [HOGN24]
7. Constraint programming [EGMN20, GMN22, MM23, MMN24, MM25]
8. Automated planning [DHN+25]
9. Enumeration of solutions/models [MNOT26]

VeriPB Resources

VeriPB tutorials, technical documentation, and more information at
https://VeriPB.org

VeriPB Resources

VeriPB tutorials, technical documentation, and more information at
https://VeriPB.org

Tutorial series with videos and slides from *WHOOPS '25* workshop at
https://jakobnordstrom.se/WHOOPS25/

VeriPB Resources

VeriPB tutorials, technical documentation, and more information at
https://VeriPB.org

Tutorial series with videos and slides from *WHOOPS '25* workshop at
https://jakobnordstrom.se/WHOOPS25/

Technical details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23, BBN+24, DMM+24, GMM+24, HOGN24, IOT+24, MMN24, DHN+25, JBBJ25, KLM+25, MM25, ABB+26a, ABB+26b, DKK+26, MNOT26, VCB26] and PhD theses [Goc22, McI26, Oer26]

VeriPB Resources

VeriPB tutorials, technical documentation, and more information at
https://VeriPB.org

Tutorial series with videos and slides from *WHOOPS '25* workshop at
https://jakobnordstrom.se/WHOOPS25/

Technical details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23, BBN+24, DMM+24, GMM+24, HOGN24, IOT+24, MMN24, DHN+25, JBBJ25, KLM+25, MM25, ABB+26a, ABB+26b, DKK+26, MNOT26, VCB26] and PhD theses [Goc22, McI26, Oer26]

Lots of concrete example files at https://gitlab.com/MIAOresearch/software/VeriPB

Summing up the First Lecture

- Combinatorial solving and optimisation 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 the First Lecture

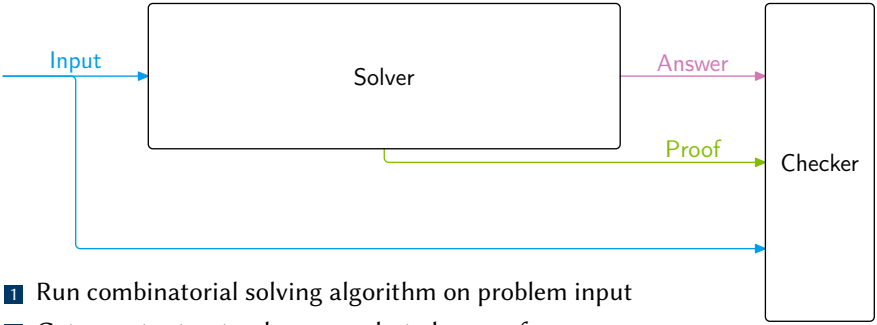
- Combinatorial solving and optimisation is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity
- **Up next:**
  - Break
  - And then more details and practical examples

Summing up the First Lecture

- Combinatorial solving and optimisation 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
- **Up next:**
  - Break
  - And then more details and practical examples

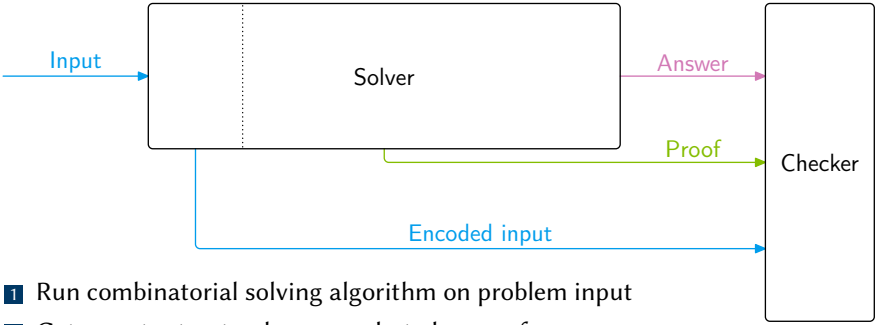
Questions so far?

Recap of Proof Logging Workflow (1/2)



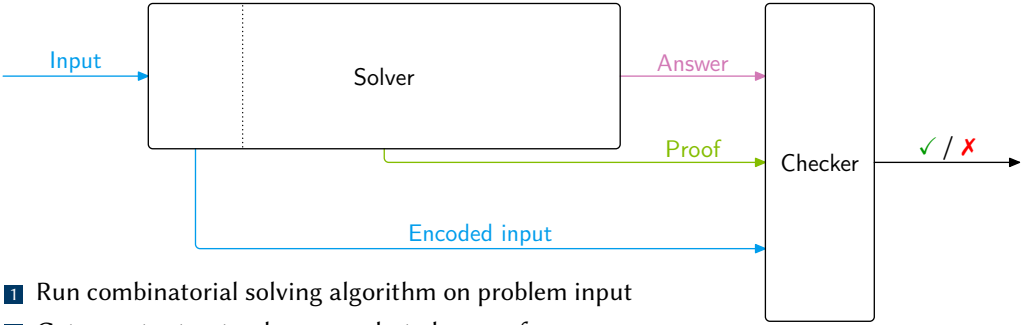
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with input

Recap of Proof Logging Workflow (1/2)



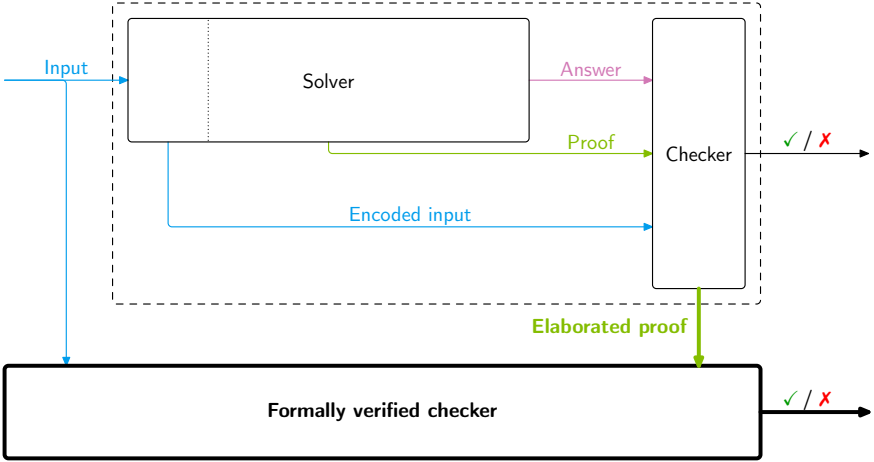
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed answer + proof to proof checker together with 0–1 ILP encoding of input

Recap of Proof Logging Workflow (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 of Proof Logging Workflow (2/2)



Implementation and Performance

Proof logging implementation

- Don't change solver
- Just add proof logging statements (plus some book-keeping)
- Proof generation should be very efficient

Performance goals for proof checking

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

- proof size
- proof checking time

VeriPB Proof Structure

1. **Preamble**
   Specify
   - VeriPB proof version
   - # input constraints (optional)

VeriPB Proof Structure

1. **Preamble**
   Specify
   - VeriPB proof version
   - # input constraints (optional)

2. **Derivation section**
   The actual proof
   Derivations of new constraints
   Logging of solutions

VeriPB Proof Structure

1. **Preamble**
   Specify
   - VeriPB proof version
   - # input constraints (optional)

2. **Derivation section**
   The actual proof
   Derivations of new constraints
   Logging of solutions

3. **Output section**
   List of constraints currently in database (optional)
   Input to next stage (or for debugging)

VeriPB Proof Structure

1. **Preamble**
   Specify
   - VeriPB proof version
   - # input constraints (optional)
2. **Derivation section**
   The actual proof
   Derivations of new constraints
   Logging of solutions
3. **Output section**
   List of constraints currently in database (optional)
   Input to next stage (or for debugging)
4. **Conclusions section**
   Specification of what was proven
   - satisfiability / unsatisfiability
   - optimality (or upper and lower bounds)
   - enumeration of solutions
   - other types of conclusions (future)

VeriPB Proof Structure: Syntax

```
pseudo-Boolean proof version 3.0
f ⟨M⟩ ;
⟨derivation part⟩
output ⟨output part⟩
conclusion ⟨conclusion part⟩
end pseudo-Boolean proof ;
```

VeriPB Proof Structure: Syntax

```
pseudo-Boolean proof version 3.0
f ⟨M⟩ ;
⟨derivation part⟩
output ⟨output part⟩
conclusion ⟨conclusion part⟩
end pseudo-Boolean proof ;
```

```
pseudo-Boolean proof version 3.0
f 18 ;
red 1 b1 >= 1 : b1 -> ~b1 b2 -> ~b2 b3 -> ~b3 b4 -> ~b4
                b5 -> ~b5 b6 -> ~b6 b7 -> ~b7 b8 -> ~b8 ;
rup >= 1 ;
output NONE ;
conclusion UNSAT ;
end pseudo-Boolean proof ;
```

VeriPB Proof Configuration

Core set $\mathscr{C}$

- Contains input formula at the start
- Input constraints can be modified (e.g., during preprocessing)
- But "equivalence" with input formula maintained

VeriPB Proof Configuration

Core set $\mathscr{C}$

- Contains input formula at the start
- Input constraints can be modified (e.g., during preprocessing)
- But "equivalence" with input formula maintained

Derived set $\mathscr{D}$

- All constraints derived during search
- Also intermediate constraints used in proof logging [but not used by solver]
- Any satisfying assignment to $\mathscr{C}$ can be extended to $\mathscr{D}$

VeriPB Proof Configuration

Core set $\mathscr{C}$

- Contains input formula at the start
- Input constraints can be modified (e.g., during preprocessing)
- But "equivalence" with input formula maintained

Objective $f = \sum_i w_i \ell_i + k$

- 0–1 linear function to minimize
- Or $f = 0$ for decision problem
- Keep track of best known bound; initialize to ∞

Derived set $\mathscr{D}$

- All constraints derived during search
- Also intermediate constraints used in proof logging [but not used by solver]
- Any satisfying assignment to $\mathscr{C}$ can be extended to $\mathscr{D}$

VeriPB Proof Configuration

Core set $\mathscr{C}$

- Contains input formula at the start
- Input constraints can be modified (e.g., during preprocessing)
- But "equivalence" with input formula maintained

Objective $f = \sum_i w_i \ell_i + k$

- 0–1 linear function to minimize
- Or $f = 0$ for decision problem
- Keep track of best known bound; initialize to ∞

Derived set $\mathscr{D}$

- All constraints derived during search
- Also intermediate constraints used in proof logging [but not used by solver]
- Any satisfying assignment to $\mathscr{C}$ can be extended to $\mathscr{D}$

Order $\mathscr{O}$

- Pseudo-Boolean formula encoding pre-order (reflexive and transitive)
- Syntactic proof of properties required
- Applied to specified variable set $\vec{z}$

Cutting Planes Rules [CCT87]

Input/model axioms — From the input

Literal axioms

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

Addition

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

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

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

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

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

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

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \min(a_i, A) \cdot \ell_i \geq A}$$

Strengthening Rules in Their (Almost) Full Formal Glory

Witness ω: substitution mapping variables to truth values or literals

Redundance-based strengthening (witness ω show how to "patch assignment")

Derive constraint C from $\mathscr{C} \cup \mathscr{D}$ if exists witness ω such that

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash (\mathscr{C} \cup \mathscr{D} \cup \{C\})\restriction_\omega \cup \{f\restriction_\omega \leq f\} \cup \mathscr{O}(\vec{z}\restriction_\omega, \vec{z})$$

Strengthening Rules in Their (Almost) Full Formal Glory

Witness ω: substitution mapping variables to truth values or literals

Redundance-based strengthening (witness ω show how to "patch assignment")

Derive constraint C from $\mathscr{C} \cup \mathscr{D}$ if exists witness ω such that

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash (\mathscr{C} \cup \mathscr{D} \cup \{C\})\restriction_\omega \cup \{f\restriction_\omega \leq f\} \cup \mathscr{O}(\vec{z}\restriction_\omega, \vec{z})$$

Dominance-based strengthening (witness ω "drives down potential")

Derive constraint C from $\mathscr{C} \cup \mathscr{D}$ if exists witness ω such that

$$\mathscr{C} \cup \mathscr{D} \cup \{\neg C\} \vdash \mathscr{C}\restriction_\omega \cup \{f\restriction_\omega \leq f\} \cup \mathscr{O}(\vec{z}\restriction_\omega, \vec{z}) \cup \neg\mathscr{O}(\vec{z}, \vec{z}\restriction_\omega)$$

Syntax for Strengthening Rules

```
red ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;

dom ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;
```

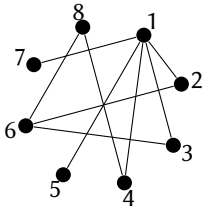
Syntax for Strengthening Rules

```
red ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;

dom ⟨Constraint C⟩ : ⟨var1⟩ -> ⟨val1⟩ . . . ⟨varN⟩ -> ⟨valN⟩ : subproof
    subproofs for proof goals
qed;
```

- Witness ω should be explicitly specified in proof log
- Subproofs of proof goals should also be explicit
- But can be skipped for proof goals "obvious" to proof checker

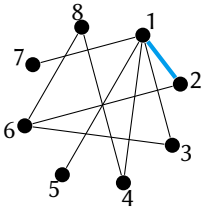
More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?

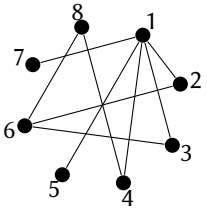


$$b_1 + b_2 \geq 1$$

$$\overline{b}_1 + \overline{b}_2 \geq 1$$

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



$$b_1 + b_2 \geq 1$$

$$\overline{b}_1 + \overline{b}_2 \geq 1$$

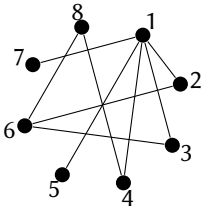
$$b_1 + b_3 \geq 1$$

$$\overline{b}_1 + \overline{b}_3 \geq 1$$

$$\ldots$$

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?

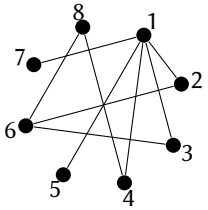


$$b_1 + b_2 \geq 1$$
$$\overline{b}_1 + \overline{b}_2 \geq 1$$
$$b_1 + b_3 \geq 1$$
$$\overline{b}_1 + \overline{b}_3 \geq 1$$
$$\ldots$$

Without loss of generality, node 1 is blue
(otherwise we can swap the two colours)

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



$$b_1 + b_2 \geq 1$$
$$\overline{b}_1 + \overline{b}_2 \geq 1$$
$$b_1 + b_3 \geq 1$$
$$\overline{b}_1 + \overline{b}_3 \geq 1$$
$$\ldots$$

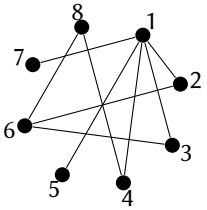
Without loss of generality, node 1 is blue (otherwise we can swap the two colours)

Derive $b_1 \geq 1$ with redundance; witness $\omega : b_i \mapsto \overline{b}_i$ for all i

$$F \doteq F\restriction_\omega \quad \text{and} \quad \neg(b_1 \geq 1) = (b_1 \geq 1)\restriction_\omega$$

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



$$b_1 + b_2 \geq 1$$
$$\overline{b}_1 + \overline{b}_2 \geq 1$$
$$b_1 + b_3 \geq 1$$
$$\overline{b}_1 + \overline{b}_3 \geq 1$$
$$\ldots$$

Without loss of generality, node 1 is blue (otherwise we can swap the two colours)

Derive $b_1 \geq 1$ with redundance; witness $\omega : b_i \mapsto \overline{b}_i$ for all i

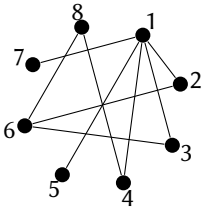
$$F \doteq F\restriction_\omega \quad \text{and} \quad \neg(b_1 \geq 1) = (b_1 \geq 1)\restriction_\omega$$

```
pseudo-Boolean proof version 3.0
f 18 ;
red 1 b1 >= 1 : b1 -> ~b1 b2 -> ~b2
↪ b3 -> ~b3 b4 -> ~b4 b5 -> ~b5
↪ b6 -> ~b6 b7 -> ~b7 b8 -> ~b8 ;
rup >= 1 ;
output NONE ;
conclusion UNSAT ;
end pseudo-Boolean proof ;
```

- Start the proof

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



$$b_1 + b_2 \geq 1$$
$$\overline{b}_1 + \overline{b}_2 \geq 1$$
$$b_1 + b_3 \geq 1$$
$$\overline{b}_1 + \overline{b}_3 \geq 1$$
$$\ldots$$

Without loss of generality, node 1 is blue (otherwise we can swap the two colours)

Derive $b_1 \geq 1$ with redundance; witness $\omega : b_i \mapsto \overline{b}_i$ for all i

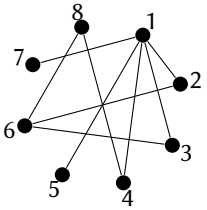
$$F \doteq F\restriction_\omega \quad \text{and} \quad \neg(b_1 \geq 1) = (b_1 \geq 1)\restriction_\omega$$

```
pseudo-Boolean proof version 3.0
f 18 ;
red 1 b1 >= 1 : b1 -> ~b1 b2 -> ~b2
↪ b3 -> ~b3 b4 -> ~b4 b5 -> ~b5
↪ b6 -> ~b6 b7 -> ~b7 b8 -> ~b8 ;
rup >= 1 ;
output NONE ;
conclusion UNSAT ;
end pseudo-Boolean proof ;
```

- Start the proof
- Derive $b_1 \geq 1$ with redundance rule

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



$$b_1 + b_2 \geq 1$$
$$\overline{b}_1 + \overline{b}_2 \geq 1$$
$$b_1 + b_3 \geq 1$$
$$\overline{b}_1 + \overline{b}_3 \geq 1$$
$$\ldots$$

Without loss of generality, node 1 is blue (otherwise we can swap the two colours)

Derive $b_1 \geq 1$ with redundance; witness $\omega : b_i \mapsto \overline{b}_i$ for all i

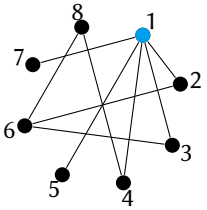
$$F \doteq F\restriction_\omega \quad \text{and} \quad \neg(b_1 \geq 1) = (b_1 \geq 1)\restriction_\omega$$

```
pseudo-Boolean proof version 3.0
f 18 ;
red 1 b1 >= 1 : b1 -> ~b1 b2 -> ~b2
↪ b3 -> ~b3 b4 -> ~b4 b5 -> ~b5
↪ b6 -> ~b6 b7 -> ~b7 b8 -> ~b8 ;
rup >= 1 ;
output NONE ;
conclusion UNSAT ;
end pseudo-Boolean proof ;
```

- Start the proof
- Derive $b_1 \geq 1$ with redundance rule
- Derive contradiction by RUP

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



$$b_1 + b_2 \geq 1$$
$$\overline{b}_1 + \overline{b}_2 \geq 1$$
$$b_1 + b_3 \geq 1$$
$$\overline{b}_1 + \overline{b}_3 \geq 1$$
$$\ldots$$

Without loss of generality, node 1 is blue (otherwise we can swap the two colours)

Derive $b_1 \geq 1$ with redundance; witness $\omega : b_i \mapsto \overline{b}_i$ for all i

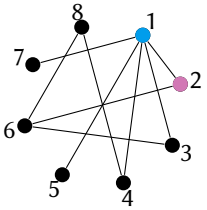
$$F \doteq F\restriction_\omega \quad \text{and} \quad \neg(b_1 \geq 1) = (b_1 \geq 1)\restriction_\omega$$

```
pseudo-Boolean proof version 3.0
f 18 ;
red 1 b1 >= 1 : b1 -> ~b1 b2 -> ~b2
↪ b3 -> ~b3 b4 -> ~b4 b5 -> ~b5
↪ b6 -> ~b6 b7 -> ~b7 b8 -> ~b8 ;
rup >= 1 ;
output NONE ;
conclusion UNSAT ;
end pseudo-Boolean proof ;
```

- Start the proof
- Derive $b_1 \geq 1$ with redundance rule
- Derive contradiction by RUP

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



$b_1 + b_2 \geq 1$

$\overline{b}_1 + \overline{b}_2 \geq 1$

$b_1 + b_3 \geq 1$

$\overline{b}_1 + \overline{b}_3 \geq 1$

$\ldots$

Without loss of generality, node 1 is blue (otherwise we can swap the two colours)

Derive $b_1 \geq 1$ with redundance; witness $\omega : b_i \mapsto \overline{b}_i$ for all i

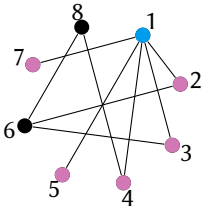
$$F \doteq F\restriction_\omega \quad \text{and} \quad \neg(b_1 \geq 1) = (b_1 \geq 1)\restriction_\omega$$

```
pseudo-Boolean proof version 3.0
f 18 ;
red 1 b1 >= 1 : b1 -> ~b1 b2 -> ~b2
↪ b3 -> ~b3 b4 -> ~b4 b5 -> ~b5
↪ b6 -> ~b6 b7 -> ~b7 b8 -> ~b8 ;
rup >= 1 ;
output NONE ;
conclusion UNSAT ;
end pseudo-Boolean proof ;
```

- Start the proof
- Derive $b_1 \geq 1$ with redundance rule
- Derive contradiction by RUP

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



$b_1 + b_2 \geq 1$

$\overline{b}_1 + \overline{b}_2 \geq 1$

$b_1 + b_3 \geq 1$

$\overline{b}_1 + \overline{b}_3 \geq 1$

$\ldots$

Without loss of generality, node 1 is blue (otherwise we can swap the two colours)

Derive $b_1 \geq 1$ with redundance; witness $\omega : b_i \mapsto \overline{b}_i$ for all i

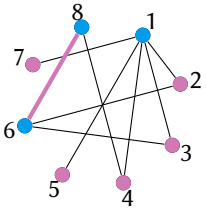
$$F \doteq F\restriction_\omega \quad \text{and} \quad \neg(b_1 \geq 1) = (b_1 \geq 1)\restriction_\omega$$

```
pseudo-Boolean proof version 3.0
f 18 ;
red 1 b1 >= 1 : b1 -> ~b1 b2 -> ~b2
↪ b3 -> ~b3 b4 -> ~b4 b5 -> ~b5
↪ b6 -> ~b6 b7 -> ~b7 b8 -> ~b8 ;
rup >= 1 ;
output NONE ;
conclusion UNSAT ;
end pseudo-Boolean proof ;
```

- Start the proof
- Derive $b_1 \geq 1$ with redundance rule
- Derive contradiction by RUP

More Advanced Example of Redundance-Based Strengthening

Is the following graph 2-colourable (bipartite)?



$$b_1 + b_2 \geq 1$$
$$\overline{b}_1 + \overline{b}_2 \geq 1$$
$$b_1 + b_3 \geq 1$$
$$\overline{b}_1 + \overline{b}_3 \geq 1$$
$$\ldots$$

Without loss of generality, node 1 is blue (otherwise we can swap the two colours)

Derive $b_1 \geq 1$ with redundance; witness $\omega : b_i \mapsto \overline{b}_i$ for all i

$$F \doteq F\restriction_\omega \quad \text{and} \quad \neg(b_1 \geq 1) = (b_1 \geq 1)\restriction_\omega$$

```
pseudo-Boolean proof version 3.0
f 18 ;
red 1 b1 >= 1 : b1 -> ~b1 b2 -> ~b2
↪ b3 -> ~b3 b4 -> ~b4 b5 -> ~b5
↪ b6 -> ~b6 b7 -> ~b7 b8 -> ~b8 ;
rup >= 1 ;
output NONE ;
conclusion UNSAT ;
end pseudo-Boolean proof ;
```

- Start the proof
- Derive $b_1 \geq 1$ with redundance rule
- Derive contradiction by RUP

The Problem of Deleting Constraints

Important to allow deletions of constraints from database

- Improves practical performance
- Affects power of strengthening rules

The Problem of Deleting Constraints

Important to allow deletions of constraints from database

- Improves practical performance
- Affects power of strengthening rules

But powerful strengthening rules create problems:

- Unsatisfiable formulas can turn satisfiable
- Satisfiable formulas can turn unsatisfiable(!)

The Problem of Deleting Constraints

Important to allow deletions of constraints from database

- Improves practical performance
- Affects power of strengthening rules

But powerful strengthening rules create problems:

- Unsatisfiable formulas can turn satisfiable
- Satisfiable formulas can turn unsatisfiable(!)

Optimisation: deletion can create too-good-to-be-true solutions

Enumeration: deletion can reintroduce already enumerated solutions

The Problem of Deleting Constraints

Important to allow deletions of constraints from database

- Improves practical performance
- Affects power of strengthening rules

But powerful strengthening rules create problems:

- Unsatisfiable formulas can turn satisfiable
- Satisfiable formulas can turn unsatisfiable(!)

Optimisation: deletion can create too-good-to-be-true solutions

Enumeration: deletion can reintroduce already enumerated solutions

Solution: distinguish between deletion from core set $\mathscr{C}$ and derived set $\mathscr{D}$

Deletion, Core Transfer, and Order Change

Deletion

1. Deletion of constraint C always OK from derived set $\mathscr{D}$
2. OK from core set $\mathscr{C}$ only if C can be rederived from $\mathscr{C} \setminus \{C\}$ with redundance rule (otherwise unchecked deletion — special conditions apply)

Deletion, Core Transfer, and Order Change

Deletion

1. Deletion of constraint C always OK from derived set $\mathscr{D}$
2. OK from core set $\mathscr{C}$ only if C can be rederived from $\mathscr{C} \setminus \{C\}$ with redundance rule (otherwise unchecked deletion — special conditions apply)

Similar to deletion rule in [JHB12] (but not implemented in DRAT)

Deletion, Core Transfer, and Order Change

Deletion

1. Deletion of constraint C always OK from derived set $\mathscr{D}$
2. OK from core set $\mathscr{C}$ only if C can be rederived from $\mathscr{C} \setminus \{C\}$ with redundance rule (otherwise unchecked deletion — special conditions apply)

Similar to deletion rule in [JHB12] (but not implemented in DRAT)

Core transfer

Constraints from $\mathscr{D}$ can be moved to $\mathscr{C}$

Deletion, Core Transfer, and Order Change

Deletion

1. Deletion of constraint C always OK from derived set $\mathscr{D}$
2. OK from core set $\mathscr{C}$ only if C can be rederived from $\mathscr{C} \setminus \{C\}$ with redundance rule (otherwise unchecked deletion — special conditions apply)

Similar to deletion rule in [JHB12] (but not implemented in DRAT)

Core transfer

Constraints from $\mathscr{D}$ can be moved to $\mathscr{C}$

Change of order

Possible to change order only if $\mathscr{D} = \emptyset$

Conclusions for Decision Problems

`NONE`
Status is undetermined

`SAT [ : ⟨assignment⟩]`
Check provided assignment against original formula
If no assignment given, then

- solution should have been logged
- no unchecked deletion must have occurred before this

`UNSAT [ : ⟨constraint ID⟩]`
Only valid if no solution has been logged
Check that specified constraint is contradictory (technically: negative slack)
If no constraint specified, check that database contains contradictory constraint

Optimisation Problems

Any solution α found is logged with `soli` "log solution and improve" command

- Provided solution α checked against current core set $\mathscr{C}$
- Objective-improving constraint $f \leq -1 + \sum_i w_i \cdot \alpha(\ell_i) + k$ added to core set (forces search for better solutions)

Optimisation Problems

Any solution α found is logged with `soli` "log solution and improve" command

- Provided solution α checked against current core set $\mathscr{C}$
- Objective-improving constraint $f \leq -1 + \sum_i w_i \cdot \alpha(\ell_i) + k$ added to core set (forces search for better solutions)

Note that

- α need not be solution for original formula
- but such solution can be reconstructed from the proof

Optimisation Problems

Any solution α found is logged with `soli` "log solution and improve" command

- Provided solution α checked against current core set $\mathscr{C}$
- Objective-improving constraint $f \leq -1 + \sum_i w_i \cdot \alpha(\ell_i) + k$ added to core set (forces search for better solutions)

Note that

- α need not be solution for original formula
- but such solution can be reconstructed from the proof

Proof of optimality: Contradiction derived from objective-improving constraint

Proof format supports not just optimality, but also non-tight upper and lower bounds

Conclusions for Optimisation Problems

`NONE`

No solution or lower bound found

`BOUNDS` ⟨`LB`⟩ `[ :` ⟨`constraint ID`⟩ `]` ⟨`UB`⟩ `[ :` ⟨`assignment`⟩ `]`

$\langle LB \rangle$ and $\langle UB \rangle$ are integers or `inf`; optimality if $\langle LB \rangle = \langle UB \rangle$

Conclusions for Optimisation Problems

`NONE`

No solution or lower bound found

`BOUNDS` ⟨`LB`⟩ `[ :` ⟨`constraint ID`⟩ `]` ⟨`UB`⟩ `[ :` ⟨`assignment`⟩ `]`

⟨LB⟩ and ⟨UB⟩ are integers or `inf`; optimality if ⟨LB⟩ = ⟨UB⟩

Lower bound

Constraint ⟨*constraint ID*⟩, if specified, should imply lower bound

Otherwise, $f \geq$ ⟨LB⟩ should be "obvious" to proof checker from current database

Conclusions for Optimisation Problems

`NONE`

No solution or lower bound found

`BOUNDS` ⟨`LB`⟩ `[ :` ⟨`constraint ID`⟩ `]` ⟨`UB`⟩ `[ :` ⟨`assignment`⟩ `]`

$\langle LB \rangle$ and $\langle UB \rangle$ are integers or `inf`; optimality if $\langle LB \rangle = \langle UB \rangle$

Lower bound

Constraint ⟨*constraint ID*⟩, if specified, should imply lower bound

Otherwise, $f \geq \langle LB \rangle$ should be "obvious" to proof checker from current database

Upper bound

Check provided assignment against original formula

If no assignment specified, then

- solution with value $\langle UB \rangle$ should have been logged
- no unchecked deletion must have occurred before this

More About VeriPB Proof Syntax

Variables

- start with a letter in `A-Z` or `a-z`
- continue with characters in `A-Z`, `a-z`, `0-9`, or `[]{}-_^`
  (square and curly brackets, hyphen, underscore, and caret)
- contain at least two characters

Constraints

Are referred to by positive integers (constraint IDs)

Derivation rules and requirements

Come in two flavours

1. kernel format for formally verified proof checker
2. augmented format with convenience rules such as reverse unit propagation (RUP)

See `https://VeriPB.org` for more details

Syntactic Sugar: Labels for Constraints

Constraint identifiers are consecutive numbers 1, 2, 3, …

- Can be annoying to keep track of the exact numbers when writing proofs
- Or plain difficult if several (sub)solvers collaborate

Syntactic Sugar: Labels for Constraints

Constraint identifiers are consecutive numbers 1, 2, 3, ...

- Can be annoying to keep track of the exact numbers when writing proofs
- Or plain difficult if several (sub)solvers collaborate

Labels simplify things — just aliases @⟨label⟩ for constraint IDs (in input file or proof file)

```
* input file (.opb)
@foo 1 x1 1 x2 >= 1 ;
@bar 1 x1 1 x3 >= 1 ;
@baz 1 x1 1 x3 >= 1 ;
```

```
% proof file (.pbp)
@sum pol @foo @bar + @baz + ;
pol @sum 2 d ;
```

Syntactic Sugar: Labels for Constraints

Constraint identifiers are consecutive numbers 1, 2, 3, ...

- Can be annoying to keep track of the exact numbers when writing proofs
- Or plain difficult if several (sub)solvers collaborate

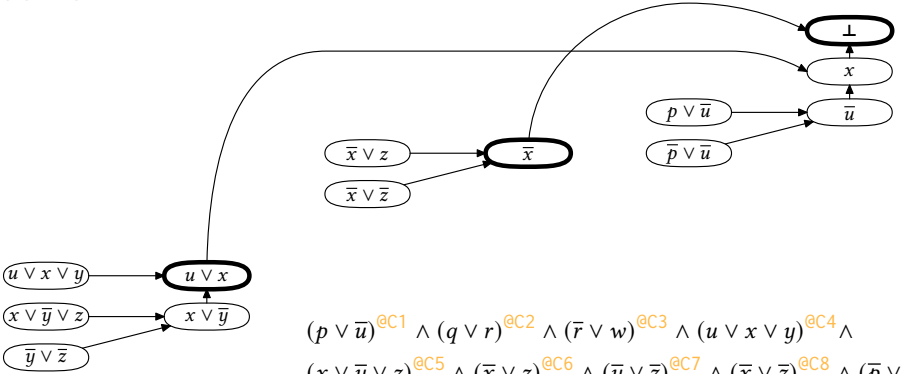
Labels simplify things — just aliases @⟨label⟩ for constraint IDs (in input file or proof file)

```
* input file (.opb)
@foo 1 x1 1 x2 >= 1 ;
@bar 1 x1 1 x3 >= 1 ;
@baz 1 x1 1 x3 >= 1 ;
```

```
% proof file (.pbp)
@sum pol @foo @bar + @baz + ;
pol @sum 2 d ;
```

- Can be used anywhere a constraint ID is expected
- Prefixing of proof rule assigns label to derived constraint
- Defining a label again just overwrites it — can be convenient when referring to simplified form of "the same constraint"

Proof Logging with Labels for CDCL Example



$$(p \vee \overline{u})^{@C1} \wedge (q \vee r)^{@C2} \wedge (\overline{r} \vee w)^{@C3} \wedge (u \vee x \vee y)^{@C4} \wedge$$
$$(x \vee \overline{y} \vee z)^{@C5} \wedge (\overline{x} \vee z)^{@C6} \wedge (\overline{y} \vee \overline{z})^{@C7} \wedge (\overline{x} \vee \overline{z})^{@C8} \wedge (\overline{p} \vee \overline{u})^{@C9}$$

```
@C10 pol @C7 @C5 + 2 d @C4 + 2 d ;
@C11 pol @C8 @C6 + 2 d ;
@C12 pol @C9 @C1 + 2 d @C10 + 2 d @C11 + 2 d ;
```

$\rightsquigarrow u + x \geq 1$

$\rightsquigarrow \overline{x} \geq 1$

$\rightsquigarrow 0 \geq 1$ ↯

Linear Pseudo-Boolean Constraints and Normalised Form

Usually, pseudo-Boolean constraints are any 0–1 integer linear constraints on the form

$$\sum_i a_i \ell_i \bowtie A$$

- $\bowtie \in \{\geq, \leq, =, >, <\}$
- $a_i, A \in \mathbb{Z}$
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)
- variables x_i take values $0 = \mathit{false}$ or $1 = \mathit{true}$

Linear Pseudo-Boolean Constraints and Normalised Form

Usually, pseudo-Boolean constraints are any 0–1 integer linear constraints on the form

$$\sum_i a_i \ell_i \bowtie A$$

- $\bowtie \in \{\geq, \leq, =, >, <\}$
- $a_i, A \in \mathbb{Z}$
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)
- variables x_i take values $0 = \mathit{false}$ or $1 = \mathit{true}$

For us, convenient to use normalised form [Bar95] (without loss of generality)

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

- constraint always greater-than-or-equal
- $a_i, A \in \mathbb{N}$ non-negative
- $A = \mathit{deg}(\sum_i a_i \ell_i \geq A)$ referred to as degree (of falsity)

Conversion to Normalised Form: Example

Normalised form is used for convenience and without loss of generality

$$-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0$$

Conversion to Normalised Form: Example

Normalised form is used for convenience and without loss of generality

$$-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0$$

1. Make inequality non-strict

$$-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 \leq -1$$

Conversion to Normalised Form: Example

Normalised form is used for convenience and without loss of generality

$$-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0$$

1. Make inequality non-strict

$$-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 \leq -1$$

2. Multiply by -1 to get greater-than-or-equal

$$x_1 - 2x_2 + 3x_3 - 4x_4 + 5x_5 \geq 1$$

Conversion to Normalised Form: Example

Normalised form is used for convenience and without loss of generality

$$-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0$$

1. Make inequality non-strict

$$-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 \leq -1$$

2. Multiply by -1 to get greater-than-or-equal

$$x_1 - 2x_2 + 3x_3 - 4x_4 + 5x_5 \geq 1$$

3. Replace $-\ell$ by $-(1 - \overline{\ell})$ [where we define $\overline{\overline{x}} \doteq x$]

$$x_1 - 2(1 - \overline{x}_2) + 3x_3 - 4(1 - \overline{x}_4) + 5x_5 \geq 1$$
$$x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

Conversion to Normalised Form: Example

Normalised form is used for convenience and without loss of generality

$$-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0$$

1. Make inequality non-strict

$$-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 \leq -1$$

2. Multiply by -1 to get greater-than-or-equal

$$x_1 - 2x_2 + 3x_3 - 4x_4 + 5x_5 \geq 1$$

3. Replace $-\ell$ by $-(1 - \overline{\ell})$ [where we define $\overline{\overline{x}} \doteq x$]

$$x_1 - 2(1 - \overline{x}_2) + 3x_3 - 4(1 - \overline{x}_4) + 5x_5 \geq 1$$
$$x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

4. Replace "=" by two inequalities "≥" and "≤"

Conversion to Normalised Form: Formal Details

Given (normalised) linear form $\sum_i a_i \ell_i$ with coefficient sum $\sum_i a_i = W$

Syntactic sugar	**Meaning**
$\sum_i a_i \ell_i > A$	$\sum_i a_i \ell_i \geq A + 1$
$\sum_i a_i \ell_i \leq A$	$\sum_i a_i \overline{\ell}_i \geq W - A$
$\sum_i a_i \ell_i < A$	$\sum_i a_i \overline{\ell}_i \geq W - A + 1$
$\sum_i a_i \ell_i = A$	$\sum_i a_i \ell_i \geq A$ and $\sum_i a_i \overline{\ell}_i \geq W - A$

Conversion to Normalised Form: Formal Details

Given (normalised) linear form $\sum_i a_i \ell_i$ with coefficient sum $\sum_i a_i = W$

Syntactic sugar	**Meaning**
$\sum_i a_i \ell_i > A$	$\sum_i a_i \ell_i \geq A + 1$
$\sum_i a_i \ell_i \leq A$	$\sum_i a_i \overline{\ell}_i \geq W - A$
$\sum_i a_i \ell_i < A$	$\sum_i a_i \overline{\ell}_i \geq W - A + 1$
$\sum_i a_i \ell_i = A$	$\sum_i a_i \ell_i \geq A$ and $\sum_i a_i \overline{\ell}_i \geq W - A$

In what follows:

- Use syntactic sugar when convenient
- Assume (implicit) normalisation whenever it matters
- Write $\doteq$ for syntactic equality

Some Convenient Notation for Constraints

Given constraint $C \doteq \sum_i a_i \ell_i \geq A$ with $\sum_i a_i = W$

Negation

$$\neg C \doteq \sum_i a_i \overline{\ell}_i \geq W - A + 1$$

Some Convenient Notation for Constraints

Given constraint $C \doteq \sum_i a_i \ell_i \geq A$ with $\sum_i a_i = W$

Negation

$$\neg C \doteq \sum_i a_i \overline{\ell}_i \geq W - A + 1$$

Reification

$$r \Rightarrow C \doteq A \cdot \overline{r} + \sum_i a_i \ell_i \geq A$$

$$r \Leftarrow C \doteq (W - A + 1) \cdot r + \sum_i a_i \overline{\ell}_i \geq W - A + 1$$

$$r \Leftrightarrow C \doteq r \Rightarrow C \text{ and } r \Leftarrow C$$

Some Convenient Notation for Constraints

Given constraint $C \doteq \sum_i a_i \ell_i \geq A$ with $\sum_i a_i = W$

Negation

$$\neg C \doteq \sum_i a_i \overline{\ell}_i \geq W - A + 1$$

Reification

$$r \Rightarrow C \doteq A \cdot \overline{r} + \sum_i a_i \ell_i \geq A$$

$$r \Leftarrow C \doteq (W - A + 1) \cdot r + \sum_i a_i \overline{\ell}_i \geq W - A + 1$$

$$r \Leftrightarrow C \doteq r \Rightarrow C \text{ and } r \Leftarrow C$$

Some calculations

$$r \Leftarrow C \doteq \overline{r} \Rightarrow \neg C$$

$$C + \neg C \doteq 0 \geq 1$$

$$deg(C) \cdot (r \geq 1) + (r \Rightarrow C) \doteq C$$

$$C + (r \Leftarrow C) \doteq deg(\neg C) \cdot r \geq 1$$

More Syntactic Sugar: Reification in VeriPB Syntax

For big-M reification constraints, can be annoying to compute the right constant
So VeriPB allows to write reified constraints with arrows

```
r ==> ⟨constraint⟩ ;
r <== ⟨constraint⟩ ;
r <==> ⟨constraint⟩ ;
```

More Syntactic Sugar: Reification in VeriPB Syntax

For big-M reification constraints, can be annoying to compute the right constant
So VeriPB allows to write reified constraints with arrows

```
r ==> ⟨constraint⟩ ;
r <== ⟨constraint⟩ ;
r <==> ⟨constraint⟩ ;
```

- <==> stands for two constraints and lives in the input file only
- <== and <==> take exactly one literal on the left
- ==> allows one or more literals, read as their conjunction

More Syntactic Sugar: Reification in VeriPB Syntax

For big-M reification constraints, can be annoying to compute the right constant
So VeriPB allows to write reified constraints with arrows

```
r ==> ⟨constraint⟩ ;
r <== ⟨constraint⟩ ;
r <==> ⟨constraint⟩ ;
```

- <==> stands for two constraints and lives in the input file only
- <== and <==> take exactly one literal on the left
- ==> allows one or more literals, read as their conjunction

For $C \doteq \sum_i a_i \ell_i \geq A$ and literals $r_1, r_2, \ldots, r_n$, we have

$$r_1 \wedge r_2 \wedge \cdots \wedge r_n \Rightarrow C \doteq A\overline{r}_1 + A\overline{r}_2 + \cdots + A\overline{r}_n + \sum_i a_i \ell_i \geq A$$

Reification in VeriPB Syntax: Concrete Examples

For $C \doteq x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5$ we have

- $deg(C) = 5$
- $W = \sum_i a_i = 10$
- $\neg C \doteq \overline{x}_1 + 2\overline{x}_2 + 3\overline{x}_3 + 4\overline{x}_4 \geq 10 - 5 + 1$

Reification in VeriPB Syntax: Concrete Examples

For $C \doteq x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5$ we have

- $deg(C) = 5$
- $W = \sum_i a_i = 10$
- $\neg C \doteq \overline{x}_1 + 2\overline{x}_2 + 3\overline{x}_3 + 4\overline{x}_4 \geq 6$

Reification in VeriPB Syntax: Concrete Examples

For $C \doteq x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5$ we have

- $deg(C) = 5$
- $W = \sum_i a_i = 10$
- $\neg C \doteq \overline{x}_1 + 2\overline{x}_2 + 3\overline{x}_3 + 4\overline{x}_4 \geq 6$

Syntactic sugar	**Meaning**
`r ==> 1 x1 2 x2 3 x3 4 x4 >= 5 ;`	$5\overline{r} + x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5$
`r <== 1 x1 2 x2 3 x3 4 x4 >= 5 ;`	$6r + \overline{x}_1 + 2\overline{x}_2 + 3\overline{x}_3 + 4\overline{x}_4 \geq 6$
`r <==> 1 x1 2 x2 3 x3 4 x4 >= 5 ;`	1st and 2nd constraint, in that order
`r1 ~r2 r3 ==> 1 x1 2 x2 3 x3 4 x4 >= 5 ;`	$5\overline{r}_1 + 5r_2 + 5\overline{r}_3 + x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5$

Reification in VeriPB Syntax: Concrete Examples

For $C \doteq x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5$ we have

- $deg(C) = 5$
- $W = \sum_i a_i = 10$
- $\neg C \doteq \overline{x}_1 + 2\overline{x}_2 + 3\overline{x}_3 + 4\overline{x}_4 \geq 6$

Syntactic sugar	**Meaning**
`r ==> 1 x1 2 x2 3 x3 4 x4 >= 5 ;`	$5\overline{r} + x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5$
`r <== 1 x1 2 x2 3 x3 4 x4 >= 5 ;`	$6r + \overline{x}_1 + 2\overline{x}_2 + 3\overline{x}_3 + 4\overline{x}_4 \geq 6$
`r <==> 1 x1 2 x2 3 x3 4 x4 >= 5 ;`	1st and 2nd constraint, in that order
`r1 ~r2 r3 ==> 1 x1 2 x2 3 x3 4 x4 >= 5 ;`	$5\overline{r}_1 + 5r_2 + 5\overline{r}_3 + x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5$

- Right reification: negate reification literal(s) and add with coefficient $deg(C) = 5$
- Left reification: negate constraint and add reification literal with coefficient $deg(\neg C) = 6$

Calculations with Reified Constraints: Concrete Examples

Still with $C \doteq x_1 + 2x_2 + 3x_3 + 4x_4 \geq 5$, so that $\neg C \doteq \overline{x}_1 + 2\overline{x}_2 + 3\overline{x}_3 + 4\overline{x}_4 \geq 6$
(and recall that $\ell + \overline{\ell} = 1$, so that opposing literals cancel and leave 1)

$$\begin{aligned} C + \neg C &\doteq (x_1 + \overline{x}_1) + 2(x_2 + \overline{x}_2) + 3(x_3 + \overline{x}_3) + 4(x_4 + \overline{x}_4) \geq 11 \\ &\doteq 0 \geq 11 - 10 \\ &\doteq 0 \geq 1 \end{aligned}$$

$$\begin{aligned} deg(C) \cdot (r \geq 1) + (r \Rightarrow C) &\doteq 5(r + \overline{r}) + x_1 + 2x_2 + 3x_3 + 4x_4 \geq 10 \\ &\doteq x_1 + 2x_2 + 3x_3 + 4x_4 \geq 10 - 5 \\ &\doteq C \end{aligned}$$

$$\begin{aligned} C + (r \Leftarrow C) &\doteq 6r + (x_1 + \overline{x}_1) + 2(x_2 + \overline{x}_2) + 3(x_3 + \overline{x}_3) + 4(x_4 + \overline{x}_4) \geq 11 \\ &\doteq 6r \geq 11 - 10 \\ &\doteq deg(\neg C) \cdot r \geq 1 \end{aligned}$$

Cutting Planes Derivations Under Assumptions

Suppose we have the constraints

$$r_1 + x_1 + x_2 \geq 1$$
$$r_1 + x_1 + x_3 \geq 1$$
$$x_2 + x_3 \geq 1$$
$$\overline{r}_2 + \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2$$

Cutting Planes Derivations Under Assumptions

Suppose we have the constraints

$$\cancel{r_1 +} x_1 + x_2 \geq 1 \quad \rightsquigarrow \quad x_1 + x_2 \geq 1$$

$$\cancel{r_1 +} x_1 + x_3 \geq 1 \quad \rightsquigarrow \quad x_1 + x_3 \geq 1$$

$$x_2 + x_3 \geq 1 \quad \rightsquigarrow \quad x_2 + x_3 \geq 1$$

$$\cancel{\overline{r}_2 +} \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2 \quad \rightsquigarrow \quad \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2$$

Assume $r_1 = 0$ and $r_2 = 1$ and simplify

Cutting Planes Derivations Under Assumptions

Suppose we have the constraints

$$\begin{aligned}
\cancel{r_1 +}\ x_1 + x_2 &\geq 1 &\rightsquigarrow\quad & x_1 + x_2 \geq 1\\
\cancel{r_1 +}\ x_1 + x_3 &\geq 1 &\rightsquigarrow\quad & x_1 + x_3 \geq 1\\
x_2 + x_3 &\geq 1 &\rightsquigarrow\quad & x_2 + x_3 \geq 1\\
\cancel{\overline{r}_2 +}\ \overline{x}_1 + \overline{x}_2 + \overline{x}_3 &\geq 2 &\rightsquigarrow\quad & \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2
\end{aligned}$$

Assume $r_1 = 0$ and $r_2 = 1$ and simplify — then cutting planes can derive contradiction

$$\dfrac{\dfrac{\dfrac{\dfrac{x_1 + x_2 \geq 1 \qquad x_1 + x_3 \geq 1}{2x_1 + x_2 + x_3 \geq 2}\ \text{Add} \qquad x_2 + x_3 \geq 1}{2x_1 + 2x_2 + 2x_3 \geq 3}\ \text{Add}}{x_1 + x_2 + x_3 \geq 2}\ \text{Divide by 2} \qquad \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{0 \geq 1}\ \text{Add}$$

Cutting Planes Derivations Under Assumptions

Suppose we have the constraints

$$\cancel{r_1 +} x_1 + x_2 \geq 1 \quad \rightsquigarrow \quad x_1 + x_2 \geq 1$$

$$\cancel{r_1 +} x_1 + x_3 \geq 1 \quad \rightsquigarrow \quad x_1 + x_3 \geq 1$$

$$x_2 + x_3 \geq 1 \quad \rightsquigarrow \quad x_2 + x_3 \geq 1$$

$$\cancel{\overline{r}_2 +} \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2 \quad \rightsquigarrow \quad \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2$$

Assume $r_1 = 0$ and $r_2 = 1$ and simplify — then cutting planes can derive contradiction

$$\dfrac{\dfrac{\dfrac{\dfrac{x_1 + x_2 \geq 1 \qquad x_1 + x_3 \geq 1}{2x_1 + x_2 + x_3 \geq 2}\,\text{Add} \qquad x_2 + x_3 \geq 1}{2x_1 + 2x_2 + 2x_3 \geq 3}\,\text{Add}}{x_1 + x_2 + x_3 \geq 2}\,\text{Divide by 2} \qquad \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{0 \geq 1}\,\text{Add}$$

This means that the original constraints imply $r_1 + \overline{r}_2 \geq 1$

The Same Derivation Without Assumptions

Now forget about the assumptions and run exactly the same derivation on the original constraints:

$$\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Divide by 2}\ \dfrac{\text{Add}\ \dfrac{r_1 + x_1 + x_2 \geq 1 \qquad r_1 + x_1 + x_3 \geq 1}{2r_1 + 2x_1 + x_2 + x_3 \geq 2} \qquad x_2 + x_3 \geq 1}{2r_1 + 2x_1 + 2x_2 + 2x_3 \geq 3}}{r_1 + x_1 + x_2 + x_3 \geq 2} \qquad \overline{r}_2 + \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{r_1 + \overline{r}_2 \geq 1}$$

The Same Derivation Without Assumptions

Now forget about the assumptions and run exactly the same derivation on the original constraints:

$$\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Divide by 2}\ \dfrac{\text{Add}\ \dfrac{r_1 + x_1 + x_2 \geq 1 \qquad r_1 + x_1 + x_3 \geq 1}{2r_1 + 2x_1 + x_2 + x_3 \geq 2} \qquad x_2 + x_3 \geq 1}{2r_1 + 2x_1 + 2x_2 + 2x_3 \geq 3}}{r_1 + x_1 + x_2 + x_3 \geq 2} \qquad \overline{r}_2 + \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{r_1 + \overline{r}_2 \geq 1}$$

This yields

$$r_1 + \overline{r}_2 \geq 1$$

[VeriPB syntax: `1 r1 1 ~r2 >= 1 ;`]

or equivalently

$$\overline{r}_1 \wedge r_2 \Rightarrow (0 \geq 1)$$

[VeriPB syntax: `~r1 r2 ==> >= 1 ;`]

The Same Derivation Without Assumptions

Now forget about the assumptions and run exactly the same derivation on the original constraints:

$$\text{Add}\ \frac{\text{Add}\ \frac{\text{Divide by 2}\ \frac{\text{Add}\ \frac{r_1 + x_1 + x_2 \geq 1 \qquad r_1 + x_1 + x_3 \geq 1}{2r_1 + 2x_1 + x_2 + x_3 \geq 2} \qquad x_2 + x_3 \geq 1}{2r_1 + 2x_1 + 2x_2 + 2x_3 \geq 3}}{r_1 + x_1 + x_2 + x_3 \geq 2} \qquad \overline{r}_2 + \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{r_1 + \overline{r}_2 \geq 1}$$

This yields

$$r_1 + \overline{r}_2 \geq 1$$

[VeriPB syntax: `1 r1 1 ~r2 >= 1 ;`]

or equivalently

$$\overline{r}_1 \wedge r_2 \Rightarrow (0 \geq 1)$$

[VeriPB syntax: `~r1 r2 ==> >= 1 ;`]

That is, our assumptions imply contradiction

Derivations Under Assumptions: The General Claim

This works in general!

Derivations Under Assumptions: The General Claim

This works in general!

- Suppose that we have
  1. constraints $C_1, C_2, \ldots, C_m$
  2. assumptions $\rho = \{\ell_1 = 1, \ell_2 = 1, \ldots, \ell_n = 1\}$
  3. cutting planes derivation of D from the restricted constraints $C_1\restriction_\rho, C_2\restriction_\rho, \ldots, C_m\restriction_\rho$ (i.e., derivation without strengthening)

Derivations Under Assumptions: The General Claim

This works in general!

- Suppose that we have
  1. constraints $C_1, C_2, \ldots, C_m$
  2. assumptions $\rho = \{\ell_1 = 1, \ell_2 = 1, \ldots, \ell_n = 1\}$
  3. cutting planes derivation of D from the restricted constraints $C_1\restriction_\rho, C_2\restriction_\rho, \ldots, C_m\restriction_\rho$ (i.e., derivation without strengthening)
- Then the same derivation applied to $C_1, C_2, \ldots, C_m$ yields a constraint at least as strong as

$$\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n \Rightarrow D \;\doteq\; deg(D) \cdot \textstyle\sum_{i=1}^{n} \overline{\ell}_i \;+\; D$$

Derivations Under Assumptions: The General Claim

This works in general!

- Suppose that we have
  1. constraints $C_1, C_2, \ldots, C_m$
  2. assumptions $\rho = \{\ell_1 = 1, \ell_2 = 1, \ldots, \ell_n = 1\}$
  3. cutting planes derivation of D from the restricted constraints $C_1\restriction_\rho, C_2\restriction_\rho, \ldots, C_m\restriction_\rho$ (i.e., derivation without strengthening)
- Then the same derivation applied to $C_1, C_2, \ldots, C_m$ yields a constraint at least as strong as

$$\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n \Rightarrow D \;\doteq\; deg(D) \cdot \textstyle\sum_{i=1}^{n} \overline{\ell}_i \;+\; D$$

Give a convenient way to construct derivations:

- Apply assumptions in ρ to constraints C_j and simplify to get $C_j\restriction_\rho$

Derivations Under Assumptions: The General Claim

This works in general!

- Suppose that we have
  1. constraints $C_1, C_2, \ldots, C_m$
  2. assumptions $\rho = \{\ell_1 = 1, \ell_2 = 1, \ldots, \ell_n = 1\}$
  3. cutting planes derivation of D from the restricted constraints $C_1\restriction_\rho, C_2\restriction_\rho, \ldots, C_m\restriction_\rho$ (i.e., derivation without strengthening)
- Then the same derivation applied to $C_1, C_2, \ldots, C_m$ yields a constraint at least as strong as

$$\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n \Rightarrow D \;\doteq\; deg(D) \cdot \textstyle\sum_{i=1}^{n} \overline{\ell}_i \;+\; D$$

Give a convenient way to construct derivations:

- Apply assumptions in ρ to constraints C_j and simplify to get $C_j\restriction_\rho$
- Derive desired conclusion D

Derivations Under Assumptions: The General Claim

This works in general!

- Suppose that we have
  1. constraints $C_1, C_2, \ldots, C_m$
  2. assumptions $\rho = \{\ell_1 = 1, \ell_2 = 1, \ldots, \ell_n = 1\}$
  3. cutting planes derivation of D from the restricted constraints $C_1\restriction_\rho, C_2\restriction_\rho, \ldots, C_m\restriction_\rho$ (i.e., derivation without strengthening)
- Then the same derivation applied to $C_1, C_2, \ldots, C_m$ yields a constraint at least as strong as

$$\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n \Rightarrow D \;\doteq\; deg(D) \cdot \textstyle\sum_{i=1}^{n} \overline{\ell}_i \;+\; D$$

Give a convenient way to construct derivations:

- Apply assumptions in ρ to constraints C_j and simplify to get $C_j\restriction_\rho$
- Derive desired conclusion D
- Remove assumptions to get reified version $\rho \Rightarrow D$ of constraint

Derivations Under Assumptions: The General Claim

This works in general!

- Suppose that we have
  1. constraints $C_1, C_2, \ldots, C_m$
  2. assumptions $\rho = \{\ell_1 = 1, \ell_2 = 1, \ldots, \ell_n = 1\}$
  3. cutting planes derivation of D from the restricted constraints $C_1\restriction_\rho, C_2\restriction_\rho, \ldots, C_m\restriction_\rho$ (i.e., derivation without strengthening)
- Then the same derivation applied to $C_1, C_2, \ldots, C_m$ yields a constraint at least as strong as

$$\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n \Rightarrow D \;\doteq\; deg(D) \cdot \textstyle\sum_{i=1}^{n} \overline{\ell}_i \;+\; D$$

Give a convenient way to construct derivations:

- Apply assumptions in ρ to constraints C_j and simplify to get $C_j\restriction_\rho$
- Derive desired conclusion D
- Remove assumptions to get reified version $\rho \Rightarrow D$ of constraint
- Important special case: Backtracking clause from decisions/guesses leading to contradiction

Progress So Far

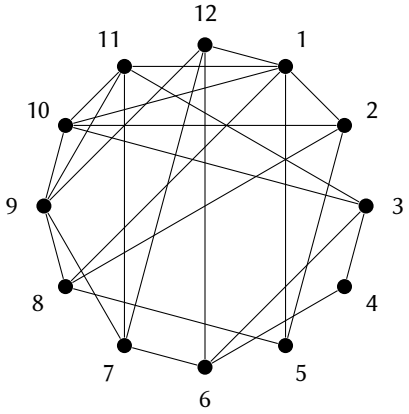
We've seen proof logging, and how it works for SAT

We've learned about

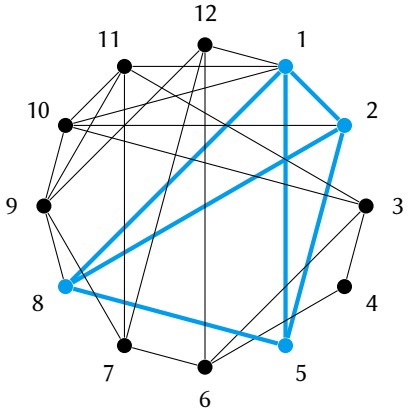
- pseudo-Boolean constraints (0–1 linear inequalities)
- cutting planes reasoning
- strengthening rules

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 away v (and all incident edges) and pick another vertex

A Brief and Incomplete Guide to Clique Solving (2/4)

Key data structures:

- Growing clique C
- Shrinking 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)

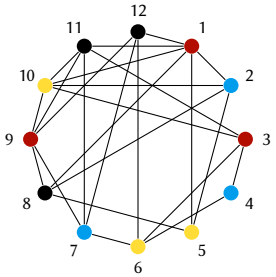
Key data structures:

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

Branch and bound:

- Remember the biggest clique $C^\star$ found so far
- If $|C| + |P| \leq |C^\star|$, no need to keep going

A Brief and Incomplete Guide to Clique Solving (3/4)



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

We can use $|C| + \#colours(P)$ as a bound, for any colouring

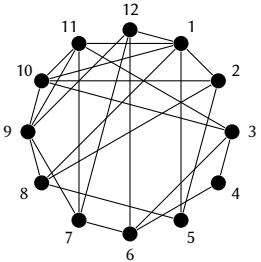
A Brief and Incomplete Guide to Clique Solving (4/4)

- This brings us to 1997
- Many improvements since then
  - better bound functions
  - clever vertex selection heuristics
  - efficient data structures
  - local search
  - ...
- But key ideas for proof logging can be explained without worrying about such things

Making a Certifying Clique Solver

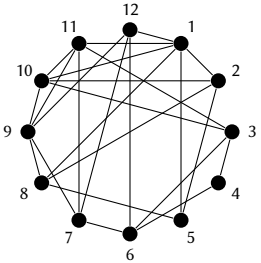
1. Output a pseudo-Boolean encoding of the problem
   - Clique problems have several standard file formats
   - Give the constraints labels, so the proof can refer to them by name
2. Make solver generate proof log for search tree
   - Output a small header
   - Output something on every backtrack
   - Output something every time a solution is found
   - Output a small footer
3. Figure out how to certify the bound function

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 ;
@noedge1_3 -1 x3 -1 x1 >= -1 ;
@noedge2_3 -1 x3 -1 x2 >= -1 ;
@noedge1_4 -1 x4 -1 x1 >= -1 ;
* . . . and a further 38 similar lines for the remaining non-edges
```

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



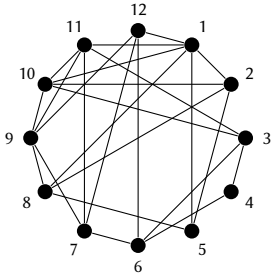

```
* #variable= 12 #constraint= 41
min: 1 ~x1 1 ~x2 1 ~x3 1 ~x4 . . . and so on. . .  1 ~x11 1 ~x12 ;
@noedge1_3 -1 x3 -1 x1 >= -1 ;
@noedge2_3 -1 x3 -1 x2 >= -1 ;
@noedge1_4 -1 x4 -1 x1 >= -1 ;
* . . . and a further 38 similar lines for the remaining non-edges
```

$x_i = 1$ means "vertex i is in the clique"

VeriPB proof minimises number of vertices not selected, so 4-clique yields objective value $12 - 4 = 8$

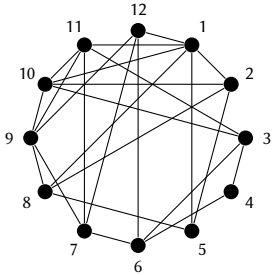
First Attempt at a Proof

```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



First Attempt at a Proof

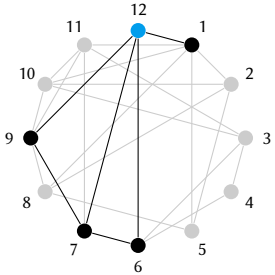
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Start with a header
The problem constraints are loaded for us

First Attempt at a Proof

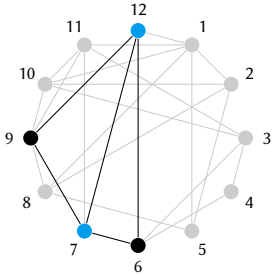
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Branch accepting 12
Throw away non-adjacent vertices

First Attempt at a Proof

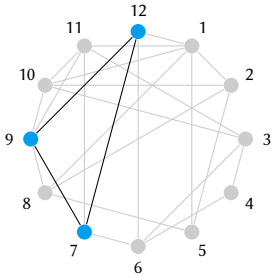
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Branch also accepting 7
Throw away non-adjacent vertices

First Attempt at a Proof

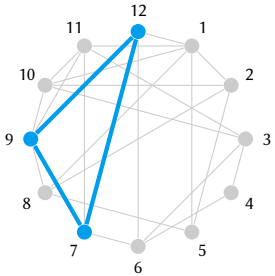
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Branch also accepting 9
Throw away non-adjacent vertices

First Attempt at a Proof

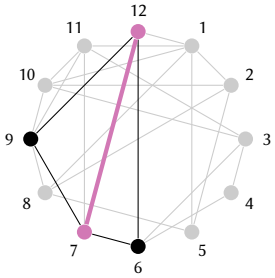
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



We branched on 12, 7, 9
Found a new incumbent
Now looking for a ≥ 4 vertex clique
"@obj" names the constraint this creates

First Attempt at a Proof

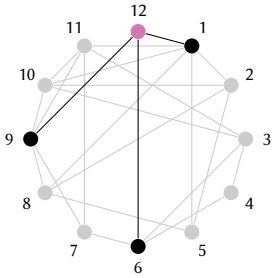
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Backtrack from 12, 7
9 explored already, only 6 feasible
No $\geq$ 4 vertex clique possible
Effectively this deletes the 7–12 edge

First Attempt at a Proof

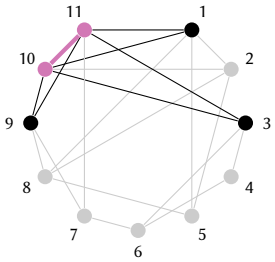
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Backtrack from 12
Only 1, 6 and 9 feasible (1-colourable)
No $\geq$ 4 vertex clique possible
Effectively this deletes vertex 12

First Attempt at a Proof

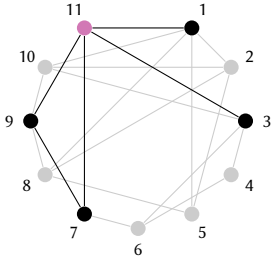
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Branch on 11 then 10
Only 1, 3 and 9 feasible (1-colourable)
No $\geq$ 4 vertex clique possible
Backtrack, deleting the edge

First Attempt at a Proof

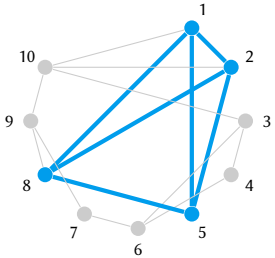
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Backtrack from 11
2-colourable, so no ≥ 4 clique
Delete the vertex

First Attempt at a Proof

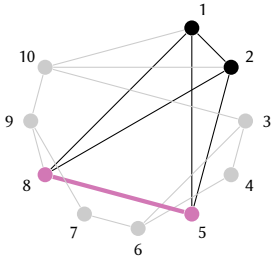
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



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

First Attempt at a Proof

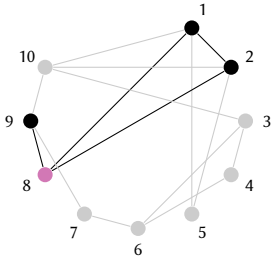
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



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

First Attempt at a Proof

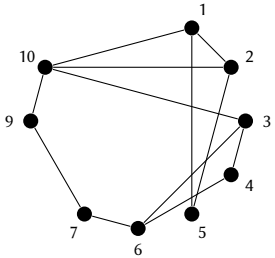
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Backtrack from 8
Still not enough vertices
Delete the vertex

First Attempt at a Proof

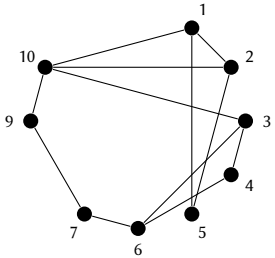
```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



Remaining graph is 3-colourable
Backtrack from root node

First Attempt at a Proof

```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
rup 1 ~x12 >= 1 ;
rup 1 ~x11 1 ~x10 >= 1 ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
rup 1 ~x8 >= 1 ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```



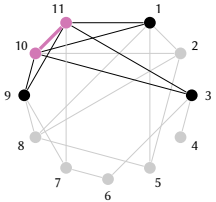
Finish with what we've concluded
We specify a lower and an upper bound
Remember we're minimising the number of vertices we *don't* select, so a 4-clique has an objective value of $12 - 4 = 8$
("output" is for features we're not using)

Verifying This Proof (Or Not...)

```
$ veripb clique.opb clique-attempt-one.pbp
Running VeriPB version 3.0.2
Error: Checking error at clique-attempt-one.pbp:5

Caused by:
        The constraint is not implied by reverse unit propagation
        (RUP) from core and derived database. Rerun with the option
        '--trace-failed' to see the propagations. . . .
```

Verifying This Proof (Or Not...)

```
$ veripb clique.opb clique-attempt-one.pbp
Running VeriPB version 3.0.2
Error: Checking error at clique-attempt-one.pbp:5

Caused by:
        The constraint is not implied by reverse unit propagation
        (RUP) from core and derived database. Rerun with the option
        '--trace-failed' to see the propagations. . . .
```



Verifying This Proof (Or Not...)

```
$ veripb --trace clique.opb clique-attempt-one.pbp
Running VeriPB version 3.0.2
  Objective: min 1 ~x1 1 ~x2 . . .  1 ~x11 1 ~x12 0
  ConstraintId 1: 1 ~x1 1 ~x3 >= 1
  ConstraintId 2: 1 ~x2 1 ~x3 >= 1
  . . .
  ConstraintId 41: 1 ~x11 1 ~x12 >= 1
line    1: pseudo-Boolean proof version 3.0
line    2: @obj soli x7 x9 x12 ;
  ConstraintId 42: 1 x1 1 x2 . . .  1 x11 1 x12 >= 4
line    3: rup 1 ~x12 1 ~x7 >= 1 ;
  ConstraintId 43: 1 ~x7 1 ~x12 >= 1
line    4: rup 1 ~x12 >= 1 ;
  ConstraintId 44: 1 ~x12 >= 1
line    5: rup 1 ~x11 1 ~x10 >= 1 ;
Error: Checking error at clique-attempt-one.pbp:5
```

Verifying This Proof (Or Not...)

```
$ veripb --trace-failed clique.opb clique-attempt-one.pbp
Running VeriPB version 3.0.2
Propagation check failed! The propagation had the following trail:
  propagations in format: <assignment> (<reason constraint>)
    ~x12 (1 ~x12 >= 1)
    x10 (1 x10 1 x11 >= 2)
    x11 (1 x10 1 x11 >= 2)
    ~x4 (1 ~x4 1 ~x10 >= 1)
    ~x5 (1 ~x5 1 ~x10 >= 1)
    ~x6 (1 ~x6 1 ~x10 >= 1)
    ~x7 (1 ~x7 1 ~x10 >= 1)
    ~x8 (1 ~x8 1 ~x10 >= 1)
    ~x2 (1 ~x2 1 ~x11 >= 1)
Error: Checking error at clique-attempt-one.pbp:5
```

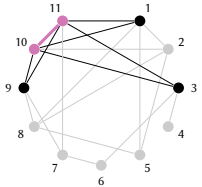
The Problem of Counting

The proof checker isn't smart enough to work out that

- vertices 1, 3 and 9 can all be given the same colour…
- and each colour class contributes at most one vertex to a clique…
- so "find me a clique with more than three vertices" can't be satisfied once we've accepted both 10 and 11

Unit propagation never sees this:

- The bound needs counting
- But the encoding only ever talks about pairs of vertices



Dealing With Colourings

So the colour bound doesn't follow by reverse unit propagation...

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

Dealing With Colourings

So the colour bound doesn't follow by reverse unit propagation…

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

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

Dealing With Colourings

So the colour bound doesn't follow by reverse unit propagation…

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

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

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 out this derivation

Details on At-Most-One Constraint Recovery (3 Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

Details on At-Most-One Constraint Recovery (3 Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

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

Details on At-Most-One Constraint Recovery (3 Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Add}\ \dfrac{\text{Add}\ \dfrac{\overline{x}_1 + \overline{x}_2 \geq 1 \qquad \overline{x}_1 + \overline{x}_3 \geq 1}{2\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2} \qquad \overline{x}_2 + \overline{x}_3 \geq 1}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}$$

Details on At-Most-One Constraint Recovery (3 Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Divide by 2}\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\overline{x}_1 + \overline{x}_2 \geq 1 \qquad \overline{x}_1 + \overline{x}_3 \geq 1}{2\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2} \qquad \overline{x}_2 + \overline{x}_3 \geq 1}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}}{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq \frac{3}{2}}$$

Details on At-Most-One Constraint Recovery (3 Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Divide by 2}\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\overline{x}_1 + \overline{x}_2 \geq 1 \qquad \overline{x}_1 + \overline{x}_3 \geq 1}{2\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2} \qquad \overline{x}_2 + \overline{x}_3 \geq 1}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}}{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq \left\lceil \frac{3}{2} \right\rceil}$$

Details on At-Most-One Constraint Recovery (3 Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Divide by 2}\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\overline{x}_1 + \overline{x}_2 \geq 1 \qquad \overline{x}_1 + \overline{x}_3 \geq 1}{2\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2} \qquad \overline{x}_2 + \overline{x}_3 \geq 1}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}}{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}$$

Details on At-Most-One Constraint Recovery (3 Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Divide by 2}\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\overline{x}_1 + \overline{x}_2 \geq 1 \qquad \overline{x}_1 + \overline{x}_3 \geq 1}{2\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2} \qquad \overline{x}_2 + \overline{x}_3 \geq 1}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}}{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}$$

Assuming constraint $\overline{x}_i + \overline{x}_j \geq 1$ has label `@Cij`, get VeriPB code

```
pol @C12 @C13 + @C23 + 2 d ;
```

Details on At-Most-One Constraint Recovery (4 and More Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

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

Details on At-Most-One Constraint Recovery (4 and More Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Multiply by 2} \quad \frac{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 4}$$

Details on At-Most-One Constraint Recovery (4 and More Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 4}\ \text{Multiply by 2} \qquad \overline{x}_1 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 + \overline{x}_4 \geq 5}\ \text{Add} \qquad \overline{x}_2 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_3 + 2\overline{x}_4 \geq 6}\ \text{Add} \qquad \overline{x}_3 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 3\overline{x}_2 + 3\overline{x}_3 + 3\overline{x}_4 \geq 7}\ \text{Add}$$

Details on At-Most-One Constraint Recovery (4 and More Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Multiply by 2}\ \overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2$$

$$\text{Add}\ 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 4 \qquad \overline{x}_1 + \overline{x}_4 \geq 1$$

$$\text{Add}\ 3\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 + \overline{x}_4 \geq 5 \qquad \overline{x}_2 + \overline{x}_4 \geq 1$$

$$\text{Add}\ 3\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_3 + 2\overline{x}_4 \geq 6 \qquad \overline{x}_3 + \overline{x}_4 \geq 1$$

$$\text{Divide by 3}\ 3\overline{x}_1 + 3\overline{x}_2 + 3\overline{x}_3 + 3\overline{x}_4 \geq 7$$

$$\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4 \geq \left\lceil \frac{7}{3} \right\rceil$$

Details on At-Most-One Constraint Recovery (4 and More Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Multiply by 2}\ \frac{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 4}$$

$$\text{Add}\ \frac{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 4 \qquad \overline{x}_1 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 + \overline{x}_4 \geq 5}$$

$$\text{Add}\ \frac{3\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 + \overline{x}_4 \geq 5 \qquad \overline{x}_2 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_3 + 2\overline{x}_4 \geq 6}$$

$$\text{Add}\ \frac{3\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_3 + 2\overline{x}_4 \geq 6 \qquad \overline{x}_3 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 3\overline{x}_2 + 3\overline{x}_3 + 3\overline{x}_4 \geq 7}$$

$$\text{Divide by 3}\ \frac{3\overline{x}_1 + 3\overline{x}_2 + 3\overline{x}_3 + 3\overline{x}_4 \geq 7}{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4 \geq 3}$$

Details on At-Most-One Constraint Recovery (4 and More Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Divide by 3}\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Multiply by 2}\ \dfrac{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 4} \qquad \overline{x}_1 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 + \overline{x}_4 \geq 5} \qquad \overline{x}_2 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_3 + 2\overline{x}_4 \geq 6} \qquad \overline{x}_3 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 3\overline{x}_2 + 3\overline{x}_3 + 3\overline{x}_4 \geq 7}}{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4 \geq 3}$$

Assuming constraint $\overline{x}_i + \overline{x}_j \geq 1$ has label `@Cij`, VeriPB code for full derivation is

```
pol @C12 @C13 + @C23 + 2 d 2 * @C14 + @C24 + @C34 + 3 d ;
```

Details on At-Most-One Constraint Recovery (4 and More Variables)

- Derive constraint $x_1 + \ldots + x_n \leq 1$ from $\binom{n}{2}$ constraints $x_i + x_j \leq 1$
- I.e., $\overline{x}_1 + \ldots + \overline{x}_n \geq n - 1$ from constraints $\overline{x}_i + \overline{x}_j \geq 1$ in normalised form
- Need this to prove that at most one vertex per colour class can be a member of the clique

$$\text{Divide by 3}\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Multiply by 2}\ \dfrac{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 2}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 4} \qquad \overline{x}_1 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 + \overline{x}_4 \geq 5} \qquad \overline{x}_2 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 3\overline{x}_2 + 2\overline{x}_3 + 2\overline{x}_4 \geq 6} \qquad \overline{x}_3 + \overline{x}_4 \geq 1}{3\overline{x}_1 + 3\overline{x}_2 + 3\overline{x}_3 + 3\overline{x}_4 \geq 7}}{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4 \geq 3}$$

Assuming constraint $\overline{x}_i + \overline{x}_j \geq 1$ has label `@Cij`, VeriPB code for full derivation is

```
pol @C12 @C13 + @C23 + 2 d 2 * @C14 + @C24 + @C34 + 3 d ;
```

(and this pattern generalizes to any number of variables)

What The Revised Proof with Explicit Counting Looks Like

```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
pol @noedge1_6 @noedge1_9 + @noedge6_9 + 2 d @obj + ;
rup 1 ~x12 >= 1 ;
pol @noedge1_3 @noedge1_9 + @noedge3_9 + 2 d @obj + ;
rup 1 ~x11 1 ~x10 >= 1 ;
pol @noedge1_3 @noedge1_7 + @noedge3_7 + 2 d @obj + ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
pol @obj @noedge1_9 + ;
rup 1 ~x8 >= 1 ;
pol @noedge1_3 @noedge1_7 + @noedge3_7 + 2 d @noedge2_4 @noedge2_9 + @noedge4_9 + 2 d +
    @noedge5_6 @noedge5_10 + @noedge6_10 + 2 d + @obj + ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```

What The Revised Proof with Explicit Counting Looks Like

```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
pol @noedge1_6 @noedge1_9 + @noedge6_9 + 2 d @obj + ;
rup 1 ~x12 >= 1 ;
pol @noedge1_3 @noedge1_9 + @noedge3_9 + 2 d @obj + ;
rup 1 ~x11 1 ~x10 >= 1 ;
pol @noedge1_3 @noedge1_7 + @noedge3_7 + 2 d @obj + ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
pol @obj @noedge1_9 + ;
rup 1 ~x8 >= 1 ;
pol @noedge1_3 @noedge1_7 + @noedge3_7 + 2 d @noedge2_4 @noedge2_9 + @noedge4_9 + 2 d +
    @noedge5_6 @noedge5_10 + @noedge6_10 + 2 d + @obj + ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```

Failing RUP steps have become pol derivations: add three non-adjacency constraints together, divide by two, then add the result to the solution-improving constraint

The labels come straight out of the OPB file

What The Revised Proof with Explicit Counting Looks Like

```
pseudo-Boolean proof version 3.0
@obj soli x7 x9 x12 ;
rup 1 ~x12 1 ~x7 >= 1 ;
pol @noedge1_6 @noedge1_9 + @noedge6_9 + 2 d @obj + ;
rup 1 ~x12 >= 1 ;
pol @noedge1_3 @noedge1_9 + @noedge3_9 + 2 d @obj + ;
rup 1 ~x11 1 ~x10 >= 1 ;
pol @noedge1_3 @noedge1_7 + @noedge3_7 + 2 d @obj + ;
rup 1 ~x11 >= 1 ;
@obj soli x1 x2 x5 x8 ;
rup 1 ~x8 1 ~x5 >= 1 ;
pol @obj @noedge1_9 + ;
rup 1 ~x8 >= 1 ;
pol @noedge1_3 @noedge1_7 + @noedge3_7 + 2 d @noedge2_4 @noedge2_9 + @noedge4_9 + 2 d +
    @noedge5_6 @noedge5_10 + @noedge6_10 + 2 d + @obj + ;
rup >= 1 ;
output NONE ;
conclusion BOUNDS 8 8 ;
end pseudo-Boolean proof ;
```

Three colour classes here, so three at-most-one constraints, all added to the solution-improving constraint

Verifying This Proof (For Real, This Time)

```
$ veripb clique.opb clique-new-attempt.pbp
Running VeriPB version 3.0.2
s VERIFIED BOUNDS 8 <= obj <= 8
```

No errors this time — every step is justified

Note that the checker knows nothing about

- cliques,
- colourings, or
- bound functions

Verifying This Proof (For Real, This Time)

```
$ veripb --trace clique.opb clique-new-attempt.pbp
Running VeriPB version 3.0.2
Objective: min 1 ~x1 1 ~x2 . . .  1 ~x11 1 ~x12 0
ConstraintId 1: 1 ~x1 1 ~x3 >= 1
. . .
ConstraintId 41: 1 ~x11 1 ~x12 >= 1
line    2: @obj soli x7 x9 x12 ;
ConstraintId 42: 1 x1 1 x2 . . .  1 x11 1 x12 >= 4
line    3: rup 1 ~x12 1 ~x7 >= 1 ;
ConstraintId 43: 1 ~x7 1 ~x12 >= 1
line    4: pol @noedge1_6 @noedge1_9 + @noedge6_9 + 2 d @obj + ;
ConstraintId 44: 1 x2 1 x3 1 x4 1 x5 1 x7 1 x8 1 x10 1 x11 1 x12 >= 3
  . . .
line   16: rup >= 1 ;
ConstraintId 55: >= 1
s VERIFIED BOUNDS 8 <= obj <= 8
```

Why a General-Purpose Proof System?

Why not just put a “colouring bound” rule into the proof format?

Why a General-Purpose Proof System?

Why not just put a "colouring bound" rule into the proof format?

- There are dozens of different colouring algorithms

Why a General-Purpose Proof System?

Why not just put a "colouring bound" rule into the proof format?

- There are dozens of different colouring algorithms
- And extensions of these algorithms using MaxSAT reasoning

Why a General-Purpose Proof System?

Why not just put a "colouring bound" rule into the proof format?

- There are dozens of different colouring algorithms
- And extensions of these algorithms using MaxSAT reasoning
- Even more for maximum weight clique, where weighted colour class rules get extremely clever (vertices can split their weight between several colours)

Why a General-Purpose Proof System?

Why not just put a "colouring bound" rule into the proof format?

- There are dozens of different colouring algorithms
- And extensions of these algorithms using MaxSAT reasoning
- Even more for maximum weight clique, where weighted colour class rules get extremely clever (vertices can split their weight between several colours)
- And we would need to do the same again for every other kind of solver

Why a General-Purpose Proof System?

Why not just put a "colouring bound" rule into the proof format?

- There are dozens of different colouring algorithms
- And extensions of these algorithms using MaxSAT reasoning
- Even more for maximum weight clique, where weighted colour class rules get extremely clever (vertices can split their weight between several colours)
- And we would need to do the same again for every other kind of solver

We would rather not have a proof format and a proof checker per solver, per paper…

Why a General-Purpose Proof System?

Why not just put a "colouring bound" rule into the proof format?

- There are dozens of different colouring algorithms
- And extensions of these algorithms using MaxSAT reasoning
- Even more for maximum weight clique, where weighted colour class rules get extremely clever (vertices can split their weight between several colours)
- And we would need to do the same again for every other kind of solver

We would rather not have a proof format and a proof checker per solver, per paper…

Cutting planes proof system to the rescue!

- Turns out to be strong enough for all of these cases
- Proof length scales linearly with solver reasoning time (asymptotically)

Different Clique Algorithms

Different search orders?

- ✓ Irrelevant for proof logging

Using local search to initialise?

- ✓ Just log the incumbent

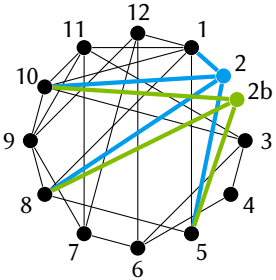
Different bound functions?

- 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"

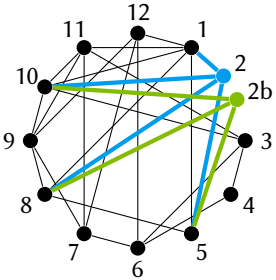
Lazy Global Domination for Maximum Clique [MP16]



Can ignore vertex 2b

- Every neighbour of 2b is also a neighbour of 2
- This is an example of dominance reasoning

Lazy Global Domination for Maximum Clique [MP16]



Can ignore vertex 2b

- Every neighbour of 2b is also a neighbour of 2
- This is an example of dominance reasoning

Dominance-based strengthening rule can justify this

- Even when detected dynamically during search

Reducing the Trust Base

We are still trusting rather a lot

Reducing the Trust Base

We are still trusting rather a lot

- That the proof checker is correct

Reducing the Trust Base

We are still trusting rather a lot

- That the proof checker is correct
- That the pseudo-Boolean encoding really does describe the clique problem we were given

Reducing the Trust Base

We are still trusting rather a lot

- That the proof checker is correct
- That the pseudo-Boolean encoding really does describe the clique problem we were given

Both can be dealt with:

Reducing the Trust Base

We are still trusting rather a lot

- That the proof checker is correct
- That the pseudo-Boolean encoding really does describe the clique problem we were given

Both can be dealt with:

- VeriPB can elaborate a proof into a simpler kernel format

Reducing the Trust Base

We are still trusting rather a lot

- That the proof checker is correct
- That the pseudo-Boolean encoding really does describe the clique problem we were given

Both can be dealt with:

- VeriPB can elaborate a proof into a simpler kernel format
- Uses only rules that the formally verified checker CakePB knows about
- The proof checker CakePB (which is written in CakeML and proved correct in HOL4)
  - reads the original graph file
  - does a formally verified translation to 0–1 ILP
  - performs formally verified checking of the kernel proof against this translation

Reducing the Trust Base

We are still trusting rather a lot

- That the proof checker is correct
- That the pseudo-Boolean encoding really does describe the clique problem we were given

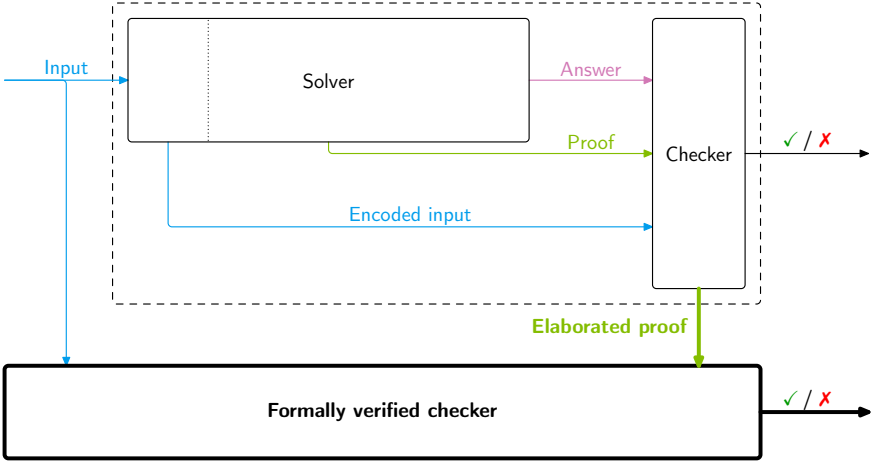
Both can be dealt with:

- VeriPB can elaborate a proof into a simpler kernel format
- Uses only rules that the formally verified checker CakePB knows about
- The proof checker CakePB (which is written in CakeML and proved correct in HOL4)
  - reads the original graph file
  - does a formally verified translation to 0–1 ILP
  - performs formally verified checking of the kernel proof against this translation
- So the answer is formally verified in terms of the original input

End-to-End Verification for Maximum Clique

```
$ glasgow_clique_solver brock200_4.clq
recursions = 56613
clique = 17 (12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192)
runtime = 0.02153s

$ glasgow_clique_solver brock200_4.clq --prove brock200_4
runtime = 0.266475s

$ cake_pb_clique brock200_4.clq > brock200_4.verifiedopb

$ veripb --elaborate brock200_4.corepb brock200_4.verifiedopb brock200_4.pbp
s VERIFIED BOUNDS 183 <= obj <= 183

$ cake_pb_clique brock200_4.clq brock200_4.corepb
s VERIFIED MAX CLIQUE SIZE |CLIQUE| = 17
```

Workflow for Solving with End-to-End Certified Results



What's Left to Trust?

Still have to trust:

- The HOL4 theorem prover
- That the formal model of the CakeML environment matches the hardware it runs on
- The HOL4 definition of what it means to be a maximum clique
- Input parsing and output formatting

What's Left to Trust?

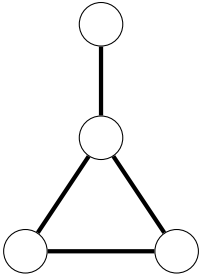
Still have to trust:

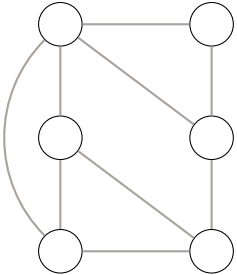
- The HOL4 theorem prover
- That the formal model of the CakeML environment matches the hardware it runs on
- The HOL4 definition of what it means to be a maximum clique
- Input parsing and output formatting

No need to trust, or even to know about:

- How the solver works
- That the VeriPB proof system is sound
- Or even what "pseudo-Boolean" means

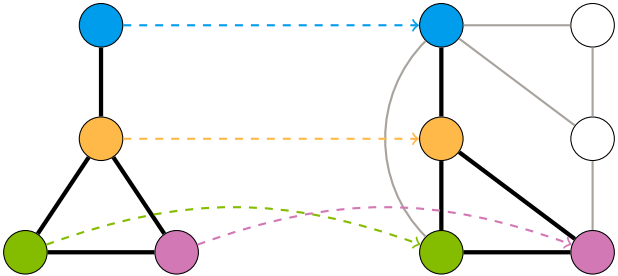
Subgraph Isomorphism





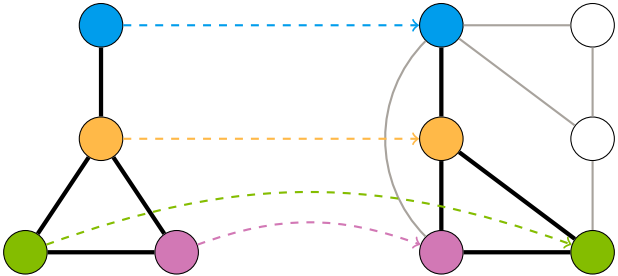
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

Subgraph Isomorphism



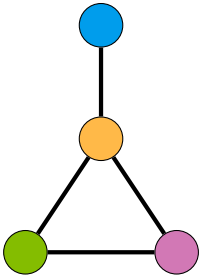
- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

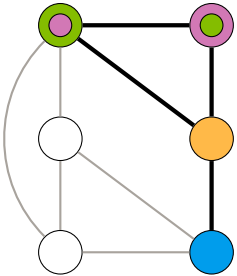
Subgraph Isomorphism



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

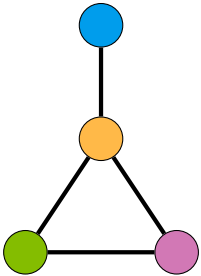
Subgraph Isomorphism

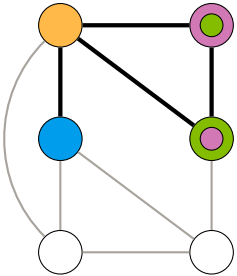




- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

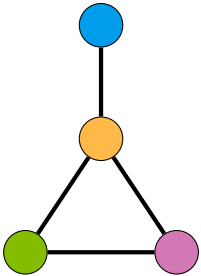
Subgraph Isomorphism

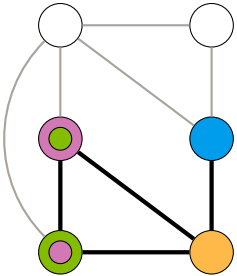




- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

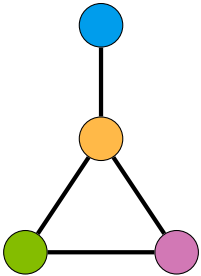
Subgraph Isomorphism

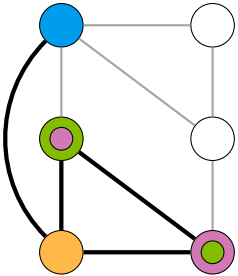




- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

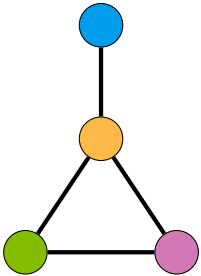
Subgraph Isomorphism

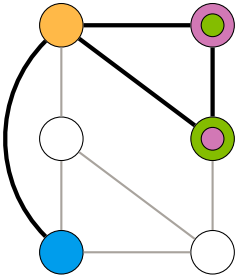




- Find the pattern inside the target
- Applications in compilers, biochemistry, model checking, pattern recognition, …
- Often want to find all matches

Subgraph Isomorphism





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

Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a target vertex:

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

Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a target vertex:

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

Each target vertex may be used at most once:

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

Subgraph Isomorphism in Pseudo-Boolean Form

Each pattern vertex gets a target vertex:

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

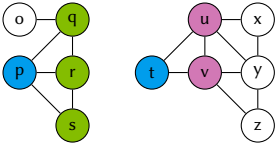
Each target vertex may be used at most once:

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

Adjacency constraints: if p maps to t, then neighbours of p map to neighbours of t:

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

Degree Reasoning in Cutting Planes

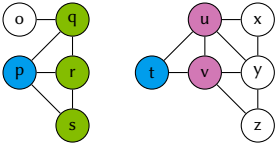


Pattern vertex p of degree $\deg(p)$ can never be mapped to target vertex t of degree $< \deg(p)$

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

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

Degree Reasoning in Cutting Planes



Adjacency:

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

Injectivity:

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

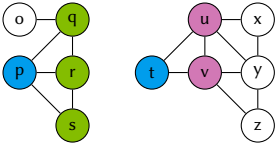
Literal axioms:

$$x_{o,u} \geq 0$$
$$x_{o,v} \geq 0$$
$$x_{p,u} \geq 0$$
$$x_{p,v} \geq 0$$

Add these together …

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

Degree Reasoning in Cutting Planes



Adjacency:

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

Injectivity:

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

Literal axioms:

$$x_{o,u} \geq 0$$
$$x_{o,v} \geq 0$$
$$x_{p,u} \geq 0$$
$$x_{p,v} \geq 0$$

Add these together and divide by 3 to get

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

Degree Reasoning in VeriPB

```
pol @adj_p_t_q @adj_p_t_r + @adj_p_t_s +   % sum adjacency constraints
    @inj_u + @inj_v +                      % sum injectivity constraints
    xo_u + xo_v +                          % cancel stray xo_* >= 0
    xp_u + xp_v +                          % cancel stray xp_* >= 0
    3 d ;                                  % divide, and we're done
```

Degree Reasoning in VeriPB

```
pol @adj_p_t_q @adj_p_t_r + @adj_p_t_s +   % sum adjacency constraints
    @inj_u + @inj_v +                      % sum injectivity constraints
    xo_u + xo_v +                          % cancel stray xo_* >= 0
    xp_u + xp_v +                          % cancel stray xp_* >= 0
    3 d ;                                  % divide, and we're done
```

The labels come from the OPB file, so solver doesn't need to keep track of constraint IDs

Degree Reasoning in VeriPB

```
pol @adj_p_t_q @adj_p_t_r + @adj_p_t_s +   % sum adjacency constraints
    @inj_u + @inj_v +                      % sum injectivity constraints
    xo_u + xo_v +                          % cancel stray xo_* >= 0
    xp_u + xp_v +                          % cancel stray xp_* >= 0
    3 d ;                                  % divide, and we're done
```

The labels come from the OPB file, so solver doesn't need to keep track of constraint IDs

Or we can let VeriPB do the last bit of tidying up for us:

```
pol @adj_p_t_q @adj_p_t_r + @adj_p_t_s +   % sum adjacency constraints
    @inj_u + @inj_v + ;                    % sum injectivity constraints
rup 1 ~xp_t >= 1 ;                         % implied by the above
```

Proof Logging for Other Forms of Reasoning

Possible to certify (almost) all of the other things state-of-the-art subgraph solvers do:

- Conditional injectivity reasoning and filtering
- Distance filtering
- Neighbourhood degree sequences
- Path filtering
- Supplemental graphs

Proof Logging for Other Forms of Reasoning

Possible to certify (almost) all of the other things state-of-the-art subgraph solvers do:

- Conditional injectivity reasoning and filtering
- Distance filtering
- Neighbourhood degree sequences
- Path filtering
- Supplemental graphs

Proof steps are "efficient" using cutting planes

- Length of proof scales linearly with time complexity of the reasoning algorithms
- Most proof steps require only trivial additional computations

Proof Logging for Other Forms of Reasoning

Possible to certify (almost) all of the other things state-of-the-art subgraph solvers do:

- Conditional injectivity reasoning and filtering
- Distance filtering
- Neighbourhood degree sequences
- Path filtering
- Supplemental graphs

Proof steps are "efficient" using cutting planes

- Length of proof scales linearly with time complexity of the reasoning algorithms
- Most proof steps require only trivial additional computations

Code for the certifying *Glasgow Subgraph Solver* at
https://github.com/ciaranm/glasgow-subgraph-solver

Limitations

Why trust the encoding?

- ✓ For clique and subgraph isomorphism this is now taken care of [GMM+24]
- Has to be done again for each new problem, but not too bad

Limitations

Why trust the encoding?

- ✓ For clique and subgraph isomorphism this is now taken care of [GMM+24]
- Has to be done again for each new problem, but not too bad

Proof logging can introduce large slowdowns

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

Limitations

Why trust the encoding?

- ✓ For clique and subgraph isomorphism this is now taken care of [GMM+24]
- Has to be done again for each new problem, but not too bad

Proof logging can introduce large slowdowns

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

Verification can be even slower

- Unit propagation also much slower than bit-parallel algorithms

Limitations

Why trust the encoding?

- ✓ For clique and subgraph isomorphism this is now taken care of [GMM+24]
- Has to be done again for each new problem, but not too bad

Proof logging can introduce large slowdowns

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

Verification can be even slower

- Unit propagation also 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"

What About Constraint Programming?

Non-Boolean variables?

Constraints?

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

Reformulations?

Integer Variables in Constraint Programming: (1/2)

How to deal with integer variables in constraint programming?

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

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

Integer Variables in Constraint Programming: (1/2)

How to deal with integer variables in constraint programming?

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…

Integer Variables in Constraint Programming: (1/2)

How to deal with integer variables in constraint programming?

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

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

This doesn't work for large domains…

We could use a binary encoding:

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

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

Bad properties for solver propagation, but that isn't a problem for proof logging

Integer Variables in Constraint Programming: (1/2)

How to deal with integer variables in constraint programming?

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

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

This doesn't work for large domains…

We could use a binary encoding:

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

Bad properties for solver propagation, but that isn't a problem for proof logging

Convention in what follows:

- Upper-case A, B, C are CP variables used by solver
- Lower-case a, b, c are corresponding Boolean variables used for PB encoding in proof log

Integer Variables in Constraint Programming: (2/2)

We can mix binary and order encodings! Define big-M linear inequalities encoding

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

$$a_{\geq 5} \Leftrightarrow -16a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} \geq 5$$

$$a_{=4} \Leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}$$

Integer Variables in Constraint Programming: (2/2)

We can mix binary and order encodings! Define big-M linear inequalities encoding

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

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

$$a_{\geq i} \Rightarrow a_{\geq j} \quad \text{and} \quad a_{\geq h} \Rightarrow a_{\geq i}$$

for the closest values $j < i < h$ that already exist

Integer Variables in Constraint Programming: (2/2)

We can mix binary and order encodings! Define big-M linear inequalities encoding

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

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

$$a_{\geq i} \Rightarrow a_{\geq j} \quad \text{and} \quad a_{\geq h} \Rightarrow a_{\geq i}$$

for the closest values $j < i < h$ that already exist

We can do this:

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

Compiling Constraints

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

Compiling Linear Inequalities

Given inequality

$$2A + 3B + 4C \geq 42$$

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

Compiling Linear Inequalities

Given inequality

$$2A + 3B + 4C \geq 42$$

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

Encode in pseudo-Boolean form as

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

Compiling Table Constraints

Constraints can be specified extensionally as list of feasible tuples, called a table

Variable assignments must match some row in table

Compiling Table Constraints

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

Given table constraint

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

define

$$3\bar{t}_1 + a_{=1} + b_{=2} + c_{=3} \geq 3 \qquad \text{i.e., } t_1 \Rightarrow (a_{=1} \wedge b_{=2} \wedge c_{=3})$$
$$3\bar{t}_2 + a_{=1} + b_{=3} + c_{=4} \geq 3 \qquad \text{i.e., } t_2 \Rightarrow (a_{=1} \wedge b_{=3} \wedge c_{=4})$$
$$3\bar{t}_3 + a_{=2} + b_{=2} + c_{=5} \geq 3 \qquad \text{i.e., } t_3 \Rightarrow (a_{=2} \wedge b_{=2} \wedge c_{=5})$$

using newly introduced tuple selector variables

$$t_1 + t_2 + t_3 = 1$$

Compiling Table Constraints

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

Given table constraint

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

define

$$3\bar{t}_1 + a_{=1} + b_{=2} + c_{=3} \geq 3 \qquad \text{i.e., } t_1 \Rightarrow (a_{=1} \wedge b_{=2} \wedge c_{=3})$$
$$3\bar{t}_2 + a_{=1} + b_{=3} + c_{=4} \geq 3 \qquad \text{i.e., } t_2 \Rightarrow (a_{=1} \wedge b_{=3} \wedge c_{=4})$$
$$3\bar{t}_3 + a_{=2} + b_{=2} + c_{=5} \geq 3 \qquad \text{i.e., } t_3 \Rightarrow (a_{=2} \wedge b_{=2} \wedge c_{=5})$$

using newly introduced tuple selector variables

$$t_1 + t_2 + t_3 = 1$$

Tuple variables used for proof logging — need not be known by solver (except for printing proofs)

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 out of the box

Restarts are easy

No need to prove anything about guesses/decisions — only justify backtracking

Justifying Inference

Key idea

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

Justifying Inference

Key idea

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

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

Some propagators and encodings need RUP steps for inferences

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

Justifying Inference

Key idea

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

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

Some propagators and encodings need RUP steps for inferences

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

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

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

Justifying All-Different Failures

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

Justifying All-Different Failures

$$
\begin{array}{rcl}
V & \in & \{\ 1 \quad\quad\ \ 4 \quad 5\ \} \\
W & \in & \{\ 1 \quad 2 \quad 3 \quad\quad\ \}\ \ \ \

Justifying All-Different Failures

$$
\begin{array}{rl}
V \in & \{\ 1 \quad\quad\ \ 4\ \ 5\ \} \\
W \in & \{\ 1\ \ 2\ \ 3\ \quad\ \ \}\quad w_{=1} + \ w_{=2} + \ w_{=3} \quad \geq \ 1 \quad [\ W \text{ takes some value}\ ] \\
X \in & \{\quad\ \ 2\ \ 3\ \quad\ \ \} \\
Y \in & \{\ 1\ \quad\ \ 3\ \quad\ \ \} \\
Z \in & \{\ 1\ \quad\ \ 3\ \quad\ \ \}
\end{array}
$$

Justifying All-Different Failures

$$
\begin{array}{rcl}
V & \in & \{\ 1 \quad\quad\ \ 4 \quad 5\ \} \\
W & \in & \{\ 1 \quad 2 \quad 3 \quad\quad\ \} \\
X & \in & \{\ \quad\ \ 2 \quad 3 \quad\quad\ \} \\
Y & \in & \{\ 1 \quad\quad 3 \quad\quad\ \} \\
Z & \in & \{\ 1 \quad\quad 3 \quad\quad\ \}
\end{array}
\qquad
\begin{array}{rcrcrcl}
w_{=1} & + & w_{=2} & + & w_{=3} & \geq & 1 \\
 & & x_{=2} & + & x_{=3} & \geq & 1 \\
y_{=1} & & & + & y_{=3} & \geq & 1 \\
z_{=1} & & & + & z_{=3} & \geq & 1
\end{array}
\qquad
\begin{array}{l}
[\ W \text{ takes some value}\ ] \\
[\ X \text{ takes some value}\ ] \\
[\ Y \text{ takes some value}\ ] \\
[\ Z \text{ takes some value}\ ]
\end{array}
$$

Justifying All-Different Failures

$$
\begin{array}{rlrl}
V \in \{\ 1 & \quad\ \ 4\ \ 5\ \} & & \\
W \in \{\ 1\ \ 2\ \ 3 & \quad\ \ \ \ \ \ \ \ \ \ \} & w_{=1} + w_{=2} + w_{=3} \geq 1 & [\ W \text{ takes some value}\ ] \\
X \in \{\ \ \ \ 2\ \ 3 & \quad\ \ \ \ \ \ \ \ \ \ \} & x_{=2} + x_{=3} \geq 1 & [\ X \text{ takes some value}\ ] \\
Y \in \{\ 1\ \ \ \ \ 3 & \quad\ \ \ \ \ \ \ \ \ \ \} & y_{=1} + y_{=3} \geq 1 & [\ Y \text{ takes some value}\ ] \\
Z \in \{\ 1\ \ \ \ \ 3 & \quad\ \ \ \ \ \ \ \ \ \ \} & z_{=1} + z_{=3} \geq 1 & [\ Z \text{ takes some value}\ ]
\end{array}
$$

$$
\begin{array}{rlrl}
\rightarrow & & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} \geq -1 & [\ \text{At most one variable} = 1\ ] \\
\rightarrow & & -w_{=2} + -x_{=2} \geq -1 & [\ \text{At most one variable} = 2\ ] \\
\rightarrow & & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1 & [\ \text{At most one variable} = 3\ ]
\end{array}
$$

Justifying All-Different Failures

$$
\begin{array}{rlcl}
V \in \{1 \quad\quad 4 \quad 5\} & & & \\
W \in \{1 \quad 2 \quad 3 \quad\quad\} & w_{=1} + w_{=2} + w_{=3} & \geq 1 & [\ W \text{ takes some value}\ ] \\
X \in \{\quad 2 \quad 3 \quad\quad\} & x_{=2} + x_{=3} & \geq 1 & [\ X \text{ takes some value}\ ] \\
Y \in \{1 \quad\quad 3 \quad\quad\} & y_{=1} + y_{=3} & \geq 1 & [\ Y \text{ takes some value}\ ] \\
Z \in \{1 \quad\quad 3 \quad\quad\} & z_{=1} + z_{=3} & \geq 1 & [\ Z \text{ takes some value}\ ] \\
\\
\rightarrow & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} & \geq -1 & [\ \text{At most one variable} = 1\ ] \\
\rightarrow & -w_{=2} + -x_{=2} & \geq -1 & [\ \text{At most one variable} = 2\ ] \\
\rightarrow & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} & \geq -1 & [\ \text{At most one variable} = 3\ ] \\
\\
 & -v_{=1} & \geq 1 & [\ \text{Sum all constraints so far}\ ]
\end{array}
$$

Justifying All-Different Failures

$$
\begin{array}{rlll}
V & \in \{\ 1 \quad\quad\ \ 4\ \ 5\ \} & & \\
W & \in \{\ 1\ \ 2\ \ 3\ \quad\ \ \} & w_{=1} + w_{=2} + w_{=3} \geq 1 & [\ W \text{ takes some value}\ ] \\
X & \in \{\quad 2\ \ 3\ \quad\ \ \} & x_{=2} + x_{=3} \geq 1 & [\ X \text{ takes some value}\ ] \\
Y & \in \{\ 1\ \quad 3\ \quad\ \ \} & y_{=1} + y_{=3} \geq 1 & [\ Y \text{ takes some value}\ ] \\
Z & \in \{\ 1\ \quad 3\ \quad\ \ \} & z_{=1} + z_{=3} \geq 1 & [\ Z \text{ takes some value}\ ]
\end{array}
$$

$$
\begin{array}{rlll}
\rightarrow & & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} \geq -1 & [\ \text{At most one variable} = 1\ ] \\
\rightarrow & & -w_{=2} + -x_{=2} \geq -1 & [\ \text{At most one variable} = 2\ ] \\
\rightarrow & & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1 & [\ \text{At most one variable} = 3\ ] \\
 & & -v_{=1} \geq 1 & [\ \text{Sum all constraints so far}\ ] \\
 & & v_{=1} \geq 0 & [\ \text{Variable } v_{=1} \text{ non-negative}\ ]
\end{array}
$$

Justifying All-Different Failures

$$
\begin{array}{rlrrrrrrrl}
V \in \{ & 1 & & & 4 & 5 \} & & & & \\
W \in \{ & 1 & 2 & 3 & & \} & w_{=1} + & w_{=2} + & w_{=3} & \geq 1 \quad [\ W \text{ takes some value}\ ] \\
X \in \{ & & 2 & 3 & & \} & & x_{=2} + & x_{=3} & \geq 1 \quad [\ X \text{ takes some value}\ ] \\
Y \in \{ & 1 & & 3 & & \} & y_{=1} & + & y_{=3} & \geq 1 \quad [\ Y \text{ takes some value}\ ] \\
Z \in \{ & 1 & & 3 & & \} & z_{=1} & + & z_{=3} & \geq 1 \quad [\ Z \text{ takes some value}\ ]
\end{array}
$$

$$
\begin{array}{rlrl}
\rightarrow & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} & \geq -1 & [\ \text{At most one variable} = 1\ ] \\
\rightarrow & -w_{=2} + -x_{=2} & \geq -1 & [\ \text{At most one variable} = 2\ ] \\
\rightarrow & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} & \geq -1 & [\ \text{At most one variable} = 3\ ]
\end{array}
$$

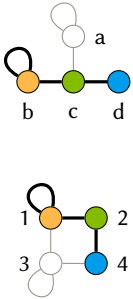
$$
\begin{array}{rlrl}
 & -v_{=1} & \geq 1 & [\ \text{Sum all constraints so far}\ ] \\
 & v_{=1} & \geq 0 & [\ \text{Variable } v_{=1} \text{ non-negative}\ ] \\
 & 0 & \geq 1 & [\ \text{Sum above two constraints}\ ]
\end{array}
$$

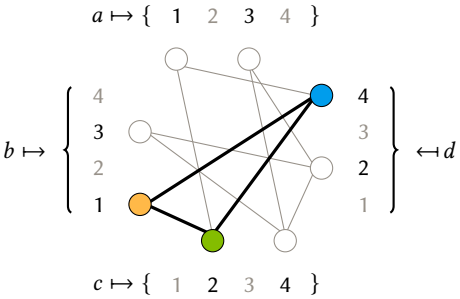
Reformulation

Auto-tabulation is possible

- Heavy use of extension variables

Can re-encode max common subgraph as clique problem without changing PB 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?

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?

We don't

Need modelling language with formally specified semantics…

The *Glasgow Constraint Solver*

Support for 37 (families of) constraints (and counting), with all inferences certified:

Abs, AllDifferent, AllEqual, Among, AtMostOne, BinPacking, Circuit, Comparison, Count, Cumulative, Diffn, Disjunctive, DivMod, Element, Equals, GCC, In, Increasing, Inverse, Knapsack, Lex, Linear, Logical, MDD, MinDistance, MinMax, Multiply, NValue, Parity, PlusMinus, Power, Regular, SeqPrecedeChain, SmartTable, Sort, Table, ValuePrecede

The *Glasgow Constraint Solver*

Support for 37 (families of) constraints (and counting), with all inferences certified:

Abs, AllDifferent, AllEqual, Among, AtMostOne, BinPacking, Circuit, Comparison, Count, Cumulative, Diffn, Disjunctive, DivMod, Element, Equals, GCC, In, Increasing, Inverse, Knapsack, Lex, Linear, Logical, MDD, MinDistance, MinMax, Multiply, NValue, Parity, PlusMinus, Power, Regular, SeqPrecedeChain, SmartTable, Sort, Table, ValuePrecede

Also certifies reformulations not written in but inferred from input:

The *Glasgow Constraint Solver*

Support for 37 (families of) constraints (and counting), with all inferences certified:

Abs, AllDifferent, AllEqual, Among, AtMostOne, BinPacking, Circuit, Comparison, Count, Cumulative, Diffn, Disjunctive, DivMod, Element, Equals, GCC, In, Increasing, Inverse, Knapsack, Lex, Linear, Logical, MDD, MinDistance, MinMax, Multiply, NValue, Parity, PlusMinus, Power, Regular, SeqPrecedeChain, SmartTable, Sort, Table, ValuePrecede

Also certifies reformulations not written in but inferred from input:

- Auto-tabulation: Identify small subproblem, solve it, and replace by table

The *Glasgow Constraint Solver*

Support for 37 (families of) constraints (and counting), with all inferences certified:

Abs, AllDifferent, AllEqual, Among, AtMostOne, BinPacking, Circuit, Comparison, Count, Cumulative, Diffn, Disjunctive, DivMod, Element, Equals, GCC, In, Increasing, Inverse, Knapsack, Lex, Linear, Logical, MDD, MinDistance, MinMax, Multiply, NValue, Parity, PlusMinus, Power, Regular, SeqPrecedeChain, SmartTable, Sort, Table, ValuePrecede

Also certifies reformulations not written in but inferred from input:

- Auto-tabulation: Identify small subproblem, solve it, and replace by table
- Capacity strengthening by integrality, conflict cliques across resources, lifted cover cuts: Extract information implied by scheduling problem constraints

The *Glasgow Constraint Solver*

Support for 37 (families of) constraints (and counting), with all inferences certified:

Abs, AllDifferent, AllEqual, Among, AtMostOne, BinPacking, Circuit, Comparison, Count, Cumulative, Diffn, Disjunctive, DivMod, Element, Equals, GCC, In, Increasing, Inverse, Knapsack, Lex, Linear, Logical, MDD, MinDistance, MinMax, Multiply, NValue, Parity, PlusMinus, Power, Regular, SeqPrecedeChain, SmartTable, Sort, Table, ValuePrecede

Also certifies reformulations not written in but inferred from input:

- Auto-tabulation: Identify small subproblem, solve it, and replace by table
- Capacity strengthening by integrality, conflict cliques across resources, lifted cover cuts: Extract information implied by scheduling problem constraints
- Difference logic: Spot that a pile of two-term inequalities is a precedence network, and hand to temporal propagator

The *Glasgow Constraint Solver*

Support for 37 (families of) constraints (and counting), with all inferences certified:

Abs, AllDifferent, AllEqual, Among, AtMostOne, BinPacking, Circuit, Comparison, Count, Cumulative, Diffn, Disjunctive, DivMod, Element, Equals, GCC, In, Increasing, Inverse, Knapsack, Lex, Linear, Logical, MDD, MinDistance, MinMax, Multiply, NValue, Parity, PlusMinus, Power, Regular, SeqPrecedeChain, SmartTable, Sort, Table, ValuePrecede

Also certifies reformulations not written in but inferred from input:

- Auto-tabulation: Identify small subproblem, solve it, and replace by table
- Capacity strengthening by integrality, conflict cliques across resources, lifted cover cuts: Extract information implied by scheduling problem constraints
- Difference logic: Spot that a pile of two-term inequalities is a precedence network, and hand to temporal propagator

Details in [EGMN20, GMN22, MM23, MMN24, MM25] *(or, better still, talk to Ciaran)*
Code at `https://github.com/ciaranm/glasgow-constraint-solver`

Future Research Directions

Performance of pseudo-Boolean proof logging and checking

- Faster proof logging and checking!
- Trim proofs while checking as in SAT solving [HHW13a] *(upcoming work [ABB+26a])*
- Compress proof files using binary format

Future Research Directions

Performance of pseudo-Boolean proof logging and checking

- Faster proof logging and checking!
- Trim proofs while checking as in SAT solving [HHW13a] *(upcoming work [ABB+26a])*
- Compress proof files using binary format

Proof logging for other combinatorial problems and techniques

- Model counting
- Mixed integer linear programming *(suggested extension of VeriPB in [DEGH23])*
- Satisfiability modulo theories (SMT) solving *(Lots of work in, e.g., [HS22, BBC+23, RSB+26, RSS+26])*

Future Research Directions

Performance of pseudo-Boolean proof logging and checking

- Faster proof logging and checking!
- Trim proofs while checking as in SAT solving [HHW13a] *(upcoming work [ABB+26a])*
- Compress proof files using binary format

Proof logging for other combinatorial problems and techniques

- Model counting
- Mixed integer linear programming *(suggested extension of VeriPB in [DEGH23])*
- Satisfiability modulo theories (SMT) solving *(Lots of work in, e.g., [HS22, BBC+23, RSB+26, RSS+26])*

And more…

- Use proof logs for algorithm analysis or explainability purposes
- Lots of other challenging problems and interesting ideas

Future Research Directions

Performance of pseudo-Boolean proof logging and checking

- Faster proof logging and checking!
- Trim proofs while checking as in SAT solving [HHW13a] *(upcoming work [ABB+26a])*
- Compress proof files using binary format

Proof logging for other combinatorial problems and techniques

- Model counting
- Mixed integer linear programming *(suggested extension of VeriPB in [DEGH23])*
- Satisfiability modulo theories (SMT) solving *(Lots of work in, e.g., [HS22, BBC+23, RSB+26, RSS+26])*

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 pseudo-Boolean proof logging revolution! ☺
  We're happy to collaborate, and we're hiring

Reminder about VeriPB Resources

VeriPB tutorials, technical documentation, and more information at
https://VeriPB.org

Tutorial series with videos and slides from *WHOOPS '25* workshop at
https://jakobnordstrom.se/WHOOPS25/

Technical details on different proof logging techniques covered in research papers [EGMN20, GMN20, GMM+20, GN21, GMN22, GMNO22, VDB22, BBN+23, BGMN23, MM23, BBN+24, DMM+24, GMM+24, HOGN24, IOT+24, MMN24, DHN+25, JBBJ25, KLM+25, MM25, ABB+26a, ABB+26b, DKK+26, MNOT26, VCB26] and PhD theses [Goc22, McI26, Oer26]

Lots of concrete example files at https://gitlab.com/MIAOresearch/software/VeriPB

Reminder About Software for Exercises Tomorrow Thursday

Kindling: See `README.md` and files in `slides-proofs/`

https://github.com/ciaranm/kindling

VeriPB (official version): See `README.md` and try to run `veripb justified.opb justified.pbp`

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

VeriPB (experimental): Interactive mode (Linux and Mac only?); see `README.md` and then use `cargo run -p veripb-repl -m path-to-veripb-repl/Cargo.toml`

https://github.com/ciaranm/veripb-repl

Summing up

Second half of lecture

- More technical details on VeriPB (mostly for reference)
- Some convenient syntactic sugar (labels and reification)
- Example applications
  1. Maximum clique
  2. Subgraph isomorphism
  3. Constraint programming *(much more about this tomorrow, including hands-on exercises)*

Summing up

Second half of lecture

- More technical details on VeriPB (mostly for reference)
- Some convenient syntactic sugar (labels and reification)
- Example applications
  1. Maximum clique
  2. Subgraph isomorphism
  3. Constraint programming *(much more about this tomorrow, including hands-on exercises)*

Main message

- Combinatorial solvers are amazing, but ensuring correctness is crucial
- Certifying solvers producing machine-verifiable proofs is the most promising approach
- Pseudo-Boolean proof logging has been surprisingly successful in moving the research frontier forward

Summing up

Second half of lecture

- More technical details on VeriPB (mostly for reference)
- Some convenient syntactic sugar (labels and reification)
- Example applications
  1. Maximum clique
  2. Subgraph isomorphism
  3. Constraint programming *(much more about this tomorrow, including hands-on exercises)*

Main message

- Combinatorial solvers are amazing, but ensuring correctness is crucial
- Certifying solvers producing machine-verifiable proofs is the most promising approach
- Pseudo-Boolean proof logging has been surprisingly successful in moving the research frontier forward
- ***Action point:*** What problems can VeriPB solve for you? ☺

Summing up

Second half of lecture

- More technical details on VeriPB (mostly for reference)
- Some convenient syntactic sugar (labels and reification)
- Example applications
  1. Maximum clique
  2. Subgraph isomorphism
  3. Constraint programming *(much more about this tomorrow, including hands-on exercises)*

Main message

- Combinatorial solvers are amazing, but ensuring correctness is crucial
- Certifying solvers producing machine-verifiable proofs is the most promising approach
- Pseudo-Boolean proof logging has been surprisingly successful in moving the research frontier forward
- ***Action point:*** What problems can VeriPB solve for you? ☺

Thank you for your attention!

References I

[ABB+26a] Berhan Oumer Adame, Bart Bogaerts, Benjamin Bogø, Simon Dold, Arthur Gontier, Wietze Koops, Ciaran McCreesh, Matthew McIlree, Jakob Nordström, Andy Oertel, Adrián Rebola-Pardo, and Mark Turnbull. Trimming pseudo-Boolean proofs. In *Proceedings of the 26th Conference on Formal Methods in Computer-Aided Design (FMCAD '26)*, September 2026. To appear.

[ABB+26b] Markus Anders, Bart Bogaerts, Benjamin Bogø, Arthur Gontier, Wietze Koops, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Adrián Rebola-Pardo, and Yong Kiam Tan. Faster certified symmetry breaking using orders with auxiliary variables. In *Proceedings of the 40th AAAI Conference on Artificial Intelligence (AAAI '26)*, pages 14140–14148, January 2026.

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

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

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

[Bar95] Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, January 1995.

[BB09] Robert Brummayer and Armin Biere. Fuzzing and delta-debugging SMT solvers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories (SMT '09)*, pages 1–5, August 2009.

References II

[BBC+23] Haniel Barbosa, Clark Barrett, Byron Cook, Bruno Dutertre, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Cesare Tinelli, and Yoni Zohar. Generating and exploiting automated reasoning proof certificates. *Communications of the ACM*, 66(10):86––95, October 2023.

[BBN+23] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, July 2023.

[BBN+24] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, Tobias Paxian, and Dieter Vandesande. Certifying without loss of generality reasoning in solution-improving maximum satisfiability. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:28, September 2024.

[BGMN23] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI '22*.

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

[Bla37] Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

[BLB10] Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, July 2010.

References III

[BMN22] Bart Bogaerts, Ciaran McCreesh, and Jakob Nordström. Solving with provably correct results: Beyond satisfiability, and towards constraint programming. Tutorial at the *28th International Conference on Principles and Practice of Constraint Programming*. Slides available at `https://jakobnordstrom.se/presentations/`, August 2022.

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

[BR07] Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming—A look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, February 2007.

[BS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

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

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

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

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

References IV

[CKSW13] William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, September 2013.

[CMS17] Luís Cruz-Filipe, João P. Marques-Silva, and Peter Schneider-Kamp. Efficient certified resolution proof checking. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 118–135. Springer, April 2017.

[DEGH23] Jasper van Doornmalen, Leon Eifler, Ambros Gleixner, and Christopher Hojny. A proof system for certifying symmetry and optimality reasoning in integer programming. Technical Report 2311.03877, arXiv.org, November 2023.

[DFS12] Nicholas Downing, Thibaut Feydy, and Peter J. Stuckey. Explaining alldifferent. In *Proceedings of the 35th Australasian Computer Science Conference (ACSC '12)*, pages 115–124, January 2012.

[DHN+25] Simon Dold, Malte Helmert, Jakob Nordström, Gabriele Röger, and Tanja Schindler. Pseudo-Boolean proof logging for optimal classical planning. In *Proceedings of the 35th International Conference on Automated Planning and Scheduling (ICAPS '25)*, pages 54–63, November 2025.

[DKK+26] Simon Dold, George Katsirelos, Wietze Koops, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. End-to-end certified graph colouring. In *Proceedings of the 32nd International Conference on Principles and Practice of Constraint Programming (CP '26)*, volume 379 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:27, July 2026.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

References V

[DMM+24] Emir Demirović, Ciaran McCreesh, Matthew McIlree, Jakob Nordström, Andy Oertel, and Konstantin Sidorov. Pseudo-Boolean reasoning about states and transitions to certify dynamic programming and decision diagram algorithms. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:21, September 2024.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[EG23] Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. *Mathematical Programming*, 197(2):793–812, February 2023. Preliminary version in *IPCO* '21.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, February 2020.

[Fle20] Mathias Fleury. *Formalization of Logical Calculi in Isabelle/HOL*. PhD thesis, Universität des Saarlandes, 2020. Available at `https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/28722`.

[GCS23] Graeme Gange, Geoffrey Chu, and Peter J. Stuckey. Certifying optimality in constraint programming. Manuscript. Available at `https://people.eng.unimelb.edu.au/pstuckey/papers/certified-cp.pdf`, 2023.

[GMM+20] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.

References VI

[GMM+24] Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. End-to-end verification for subgraph solving. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI '24)*, pages 8038–8047, February 2024.

[GMN20] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 1134–1140, July 2020.

[GMN22] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In *Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP '22)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, August 2022.

[GMNO22] Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:25, August 2022.

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

[GN21] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.

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

References VII

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

[HHW13a] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, pages 181–188, October 2013.

[HHW13b] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, June 2013.

[HOGN24] Alexander Hoen, Andy Oertel, Ambros Gleixner, and Jakob Nordström. Certifying MIP-based presolve reductions for 0–1 integer linear programs. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14742 of *Lecture Notes in Computer Science*, pages 310–328. Springer, May 2024.

[HS22] Jochen Hoenicke and Tanja Schindler. A simple proof format for SMT. In *Proceedings of the 20th Internal Workshop on Satisfiability Modulo Theories (SMT '22)*, volume 3185 of *CEUR Workshop Proceedings*, pages 54–70, August 2022.

[IOT+24] Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström. Certified MaxSAT preprocessing. In *Proceedings of the 12th International Joint Conference on Automated Reasoning (IJCAR '24)*, volume 14739 of *Lecture Notes in Computer Science*, pages 396–418. Springer, July 2024.

[JBBJ25] Christoph Jabs, Jeremias Berg, Bart Bogaerts, and Matti Järvisalo. Certifying Pareto-optimality in multi objective maximum satisfiability. In *Proceedings of the 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '25)*, volume 15697 of *Lecture Notes in Computer Science*, pages 108–129. Springer, May 2025.

References VIII

[JHB12] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR '12)*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, June 2012.

[KB22] Daniela Kaufmann and Armin Biere. Fuzzing and delta debugging and-inverter graph verification tools. In *Proceedings of the 16th International Conference on Tests and Proofs (TAP '22)*, volume 13361 of *Lecture Notes in Computer Science*, pages 69–88. Springer, July 2022.

[KLM+25] Wietze Koops, Daniel Le Berre, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Yong Kiam Tan, and Marc Vinyals. Practically feasible proof logging for pseudo-Boolean optimization. In *Proceedings of the 31st International Conference on Principles and Practice of Constraint Programming (CP '25)*, volume 340 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:27, August 2025.

[KM21] Sonja Kraiczy and Ciaran McCreesh. Solving graph homomorphism and subgraph isomorphism problems faster through clique neighbourhood constraints. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI '21)*, pages 1396–1402, August 2021.

[KRH18] Benjamin Kiesl, Adrián Rebola-Pardo, and Marijn J. H. Heule. Extended resolution simulates DRAT. In *Proceedings of the 9th International Joint Conference on Automated Reasoning (IJCAR '18)*, volume 10900 of *Lecture Notes in Computer Science*, pages 516–531. Springer, July 2018.

[McI26] Matthew McIlree. *Pseudo-Boolean Proof Logging for Constraint Propagation Algorithms*. PhD thesis, University of Glasgow, June 2026. Available at `https://theses.gla.ac.uk/86049/`.

[MM23] Matthew McIlree and Ciaran McCreesh. Proof logging for smart extensional constraints. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP '23)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:17, August 2023.

References IX

[MM25] Matthew McIlree and Ciaran McCreesh. Certifying bounds propagation for integer multiplication constraints. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI '25)*, pages 11309–11317, February-March 2025.

[MMN24] Matthew McIlree, Ciaran McCreesh, and Jakob Nordström. Proof logging for the circuit constraint. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14743 of *Lecture Notes in Computer Science*, pages 38–55. Springer, May 2024.

[MMNS11] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, May 2011.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[MNOT26] Ciaran McCreesh, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. Proof logging for projected enumeration (and counting?) problems in VeriPB. In *Proceedings of the 32nd International Conference on Principles and Practice of Constraint Programming (CP '26)*, volume 379 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:21, July 2026.

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

[MPP19] Ciaran McCreesh, William Pettersson, and Patrick Prosser. Understanding the empirical hardness of random optimisation problems. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 333–349. Springer, September 2019.

[MS99] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

References X

[NPB22] Aina Niemetz, Mathias Preiner, and Clark W. Barrett. Murxla: A modular and highly extensible API fuzzer for SMT solvers. In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV '22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 92–106. Springer, August 2022.

[Oer26] Andy Oertel. *Certifying Combinatorial Optimization: A Unified Approach Using Pseudo-Boolean Reasoning*. PhD thesis, Lund University, May 2026. Available at `https://portal.research.lu.se/sv/publications/certifying-combinatorial-optimization-a-unified-approach-using-ps`.

[OSC09] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, January 2009. Preliminary versions of these results appeared in *CP '07* and *CATS '08*.

[PB23] Tobias Paxian and Armin Biere. Uncovering and classifying bugs in MaxSAT solvers through fuzzing and delta debugging. In *Proceedings of the 14th International Workshop on Pragmatics of SAT*, volume 3545 of *CEUR Workshop Proceedings*, pages 59–71. CEUR-WS.org, July 2023.

[RM16] Olivier Roussel and Vasco M. Manquinho. Input/output format and solver requirements for the competitions of pseudo-Boolean solvers. Revision 2324. Available at `http://www.cril.univ-artois.fr/PB16/format.pdf`, January 2016.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[RSB+26] Andrew Reynolds, Hans-Jörg Schurr, Haniel Barbosa, Ofec Israel, Jibiana Jakpor, Hanna Lachnitt, Abdalrhman Mohamed, Aina Niemetz, Mathias Preiner, Yoni Zohar, Robert B. Jones, Clark W. Barrett, and Cesare Tinelli. The cooperating proof calculus: Comprehensive proofs for an SMT solver. In *Proceedings of the 38th International Conference on Computer Aided Verification (CAV '26), Part II*, volume 16683 of *Lecture Notes in Computer Science*, pages 188–212. Springer, July 2026.

References XI

[RSS+26] Andrew Reynolds, Hans-Jörg Schurr, Mallku Soldevila, Haniel Barbosa, Clark W. Barrett, and Cesare Tinelli. Ethos: A fast proof checker for the Eunoia logical framework. In *Proceedings of the 13th International Joint Conference on Automated Reasoning (IJCAR '26), Part I*, volume 16688 of *Lecture Notes in Computer Science*, pages 305–314. Springer, July 2026.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[Tse68] Grigori Tseitin. On the complexity of derivation in propositional calculus. In A. O. Silenko, editor, *Structures in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Consultants Bureau, New York-London, 1968.

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

[VCB26] Dieter Vandesande, Jordi Coll, and Bart Bogaerts. Certified branch-and-bound MaxSAT solving. pages 14342–14351, January 2026.

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

[VS10] Michael Veksler and Ofer Strichman. A proof-producing CSP solver. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI '10)*, pages 204–209, July 2010.

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