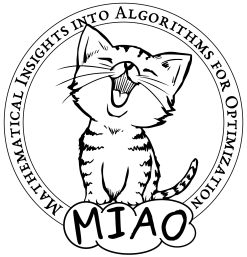
An Introduction to Pseudo-Boolean Proof Logging

Jakob Nordström

University of Copenhagen and Lund University



2nd International Workshop on Highlights in Organizing and Optimizing Proof-logging Systems
Paris, France
September 13–14, 2025

Based on Joint Work With…

- 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
- 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
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- Yong Kiam Tan
- James Trimble
- Dieter Vandesande
- Marc Vinyals

Based on Joint Work With…

- 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
- 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
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- Yong Kiam Tan
- James Trimble
- Dieter Vandesande
- Marc Vinyals

Based on Joint Work With…

- 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
- **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
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- Yong Kiam Tan
- James Trimble
- Dieter Vandesande
- Marc Vinyals

Based on Joint Work With…

- 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
- **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
- Gabriele Röger
- Tanja Schindler
- Konstantin Sidorov
- **Yong Kiam Tan**
- James Trimble
- Dieter Vandesande
- Marc Vinyals

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

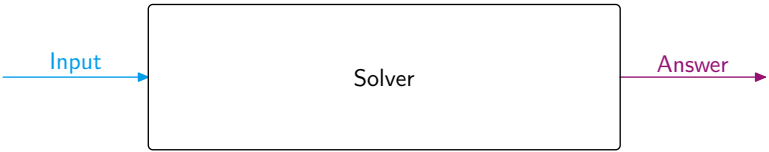
What Can Be Done About Solver Bugs?

- **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 verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to level of complexity in modern solvers
  (Despite valiant efforts in, e.g., [Fle20])

What Can Be Done About Solver Bugs?

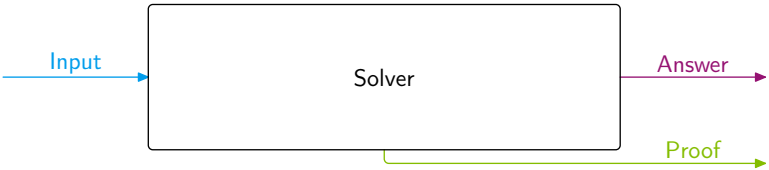
- **Software testing**
  Very useful, but bugs slip through even with careful domain-specific testing
  Progress using fuzzing and delta debugging [BB09, BLB10, KB22, NPB22, PB23]
  But testing inherently can only detect presence of bugs, not absence
- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current 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



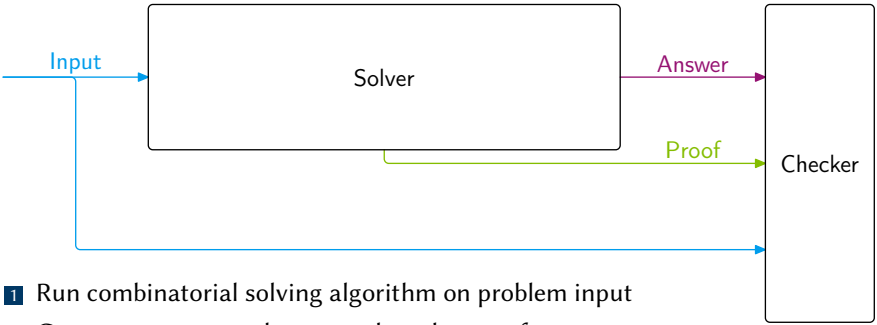
1. Run combinatorial solving algorithm on problem input

Proof Logging with Certifying Solvers: Workflow



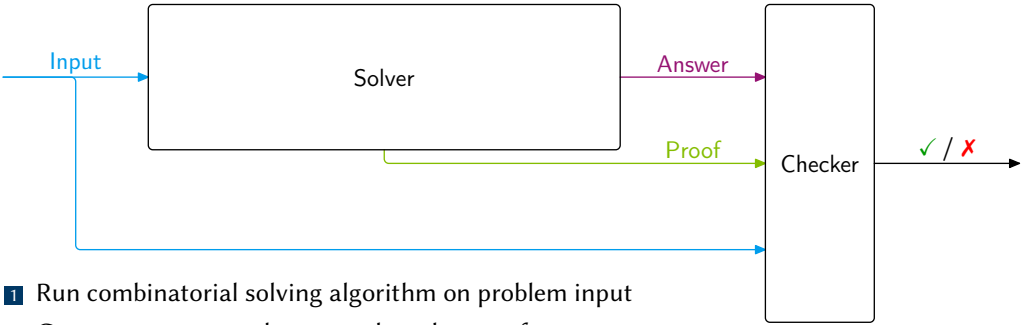
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

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

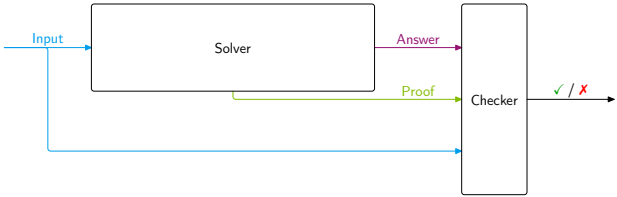
Proof Logging with Certifying Solvers: Workflow



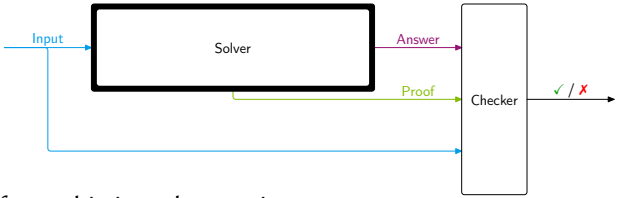
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

Proof Logging Desiderata

Proof format for certifying solver should be



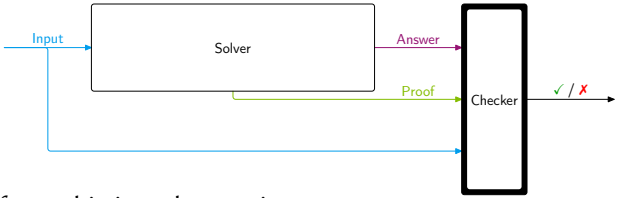
Proof Logging Desiderata



Proof format for certifying solver should be

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

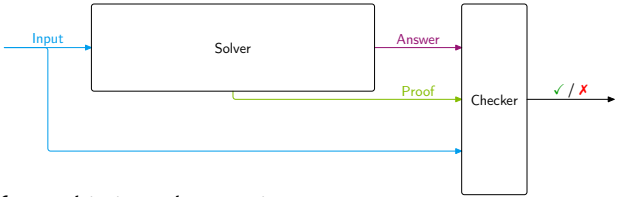
Proof Logging Desiderata



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

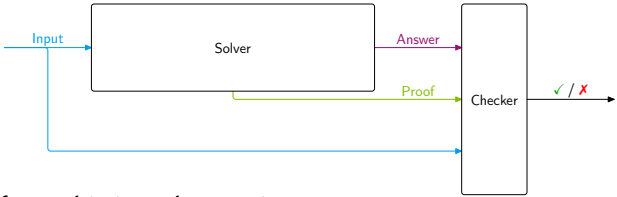


Proof format for certifying solver should be

- **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?

Take-Away Message from This Tutorial Day

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

Take-Away Message from This Tutorial Day

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

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

The Sales Pitch For Proof Logging

1. Certifies correctness of computed results
2. Detects errors even if due to compiler bugs, hardware failures, or cosmic rays
3. Provides debugging support during software development [GMM+20, KM21, BBN+23, EG23, KLM+25]
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, symmetries, ...
- But reason efficiently about such notions using power of proof system
- Combine proof logging with formally verified proof checker

Program for This Tutorial Day

Explain how to use VeriPB as a unified proof logging method for

- **09:00** SAT solving (Jakob Nordström)
- **10:00** Subgraph solving (Ciaran McCreesh)
- **11:30** Constraint programming (Matthew McIlree)
- **14:00** Pseudo-Boolean optimisation (Wietze Koops)
- **15:30** Preprocessing/presolving in MaxSAT and 0–1 linear programming (Andy Oertel)
- **16:30** Symmetry breaking (Markus Anders)

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

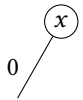
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

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

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

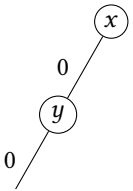


Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

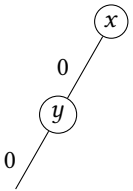


Davis-Putman-Logemann-Loveland (DPLL)

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

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

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



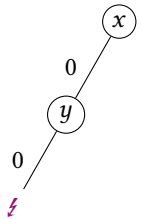
Davis-Putman-Logemann-Loveland (DPLL)

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 $x \vee y$



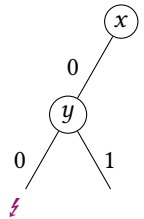
Davis-Putman-Logemann-Loveland (DPLL)

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 $x \vee y$



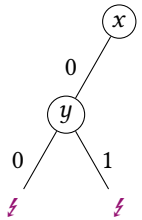
Davis-Putman-Logemann-Loveland (DPLL)

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. $x \vee \overline{y}$



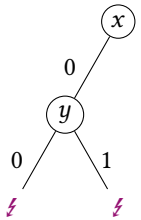
Davis-Putman-Logemann-Loveland (DPLL)

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



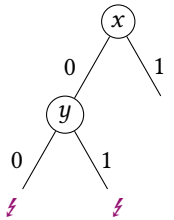
Davis-Putman-Logemann-Loveland (DPLL)

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



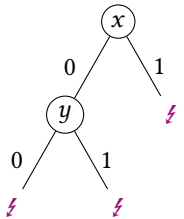
Davis-Putman-Logemann-Loveland (DPLL)

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. $\overline{x}$



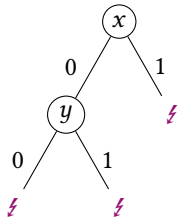
Davis-Putman-Logemann-Loveland (DPLL)

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 a proof, need backtrack clauses to be easily verifiable

Reverse Unit Propagation (RUP)

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

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

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

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

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

Reverse Unit Propagation (RUP)

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

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

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

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

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

Fact

Backtrack clauses from DPLL solver generate a RUP proof

What About Conflict-Driven Clause Learning (CDCL)?

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

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

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:

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

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

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:

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

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

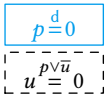
Decision
Free choice to assign value to variable
Notation $p \overset{\mathrm{d}}{=} 0$

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:

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



Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

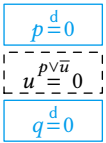
Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

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:

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



Decision
Free choice to assign value to variable
Notation $p \overset{\mathrm{d}}{=} 0$

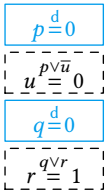
Unit propagation
Forced choice to avoid falsifying clause
Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$
Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

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:

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



Decision
Free choice to assign value to variable
Notation $p \overset{d}{=} 0$

Unit propagation
Forced choice to avoid falsifying clause
Given $p = 0$, clause $p \vee \overline{u}$ 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:

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

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

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

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

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

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

Decision
Free choice to assign value to variable
Notation $p \overset{\mathrm{d}}{=} 0$

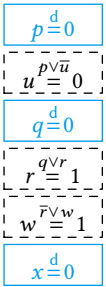
Unit propagation
Forced choice to avoid falsifying clause
Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$
Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

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:

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



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

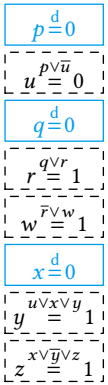
Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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



Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

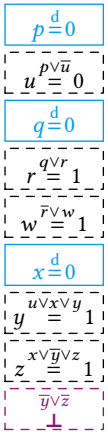
Add to assignment trail

Continue until satisfying assignment or conflict

What About Conflict-Driven Clause Learning (CDCL)?

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

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



Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

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

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

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

$\bot$

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!

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

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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

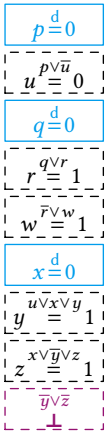
$\bot$

decision level 3

Conflict Analysis

Time to analyse this conflict and learn from it!

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



decision level 1

decision level 2

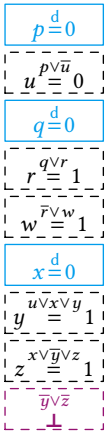
decision level 3

Could backtrack by erasing conflict level & flipping last decision

Conflict Analysis

Time to analyse this conflict and learn from it!

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



decision level 1

decision level 2

decision level 3

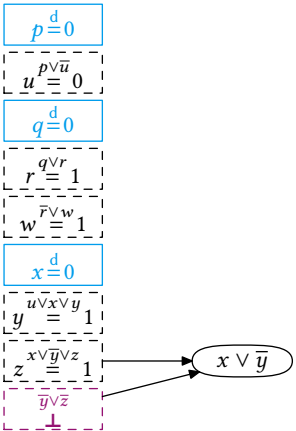
Could backtrack by erasing conflict level & flipping last decision

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

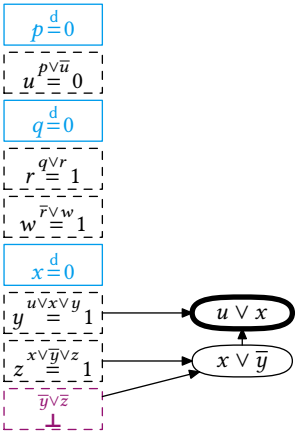
Case analysis over z for last two clauses:

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

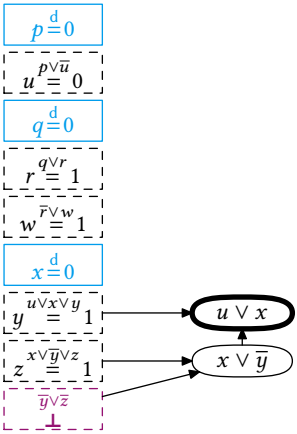
- $x \vee \overline{y} \vee z$ wants $z = 1$
- $\overline{y} \vee \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \vee \overline{y}$

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

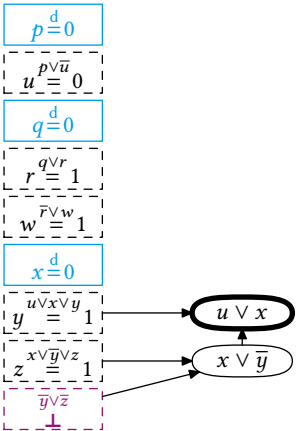
$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

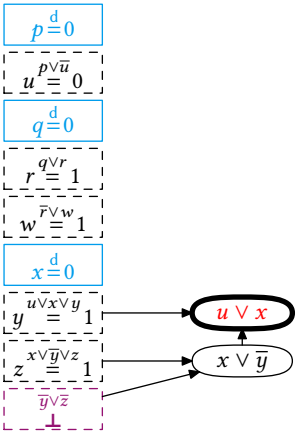


Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



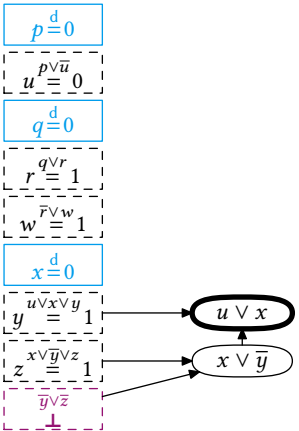
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

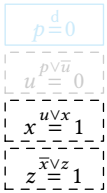
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

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

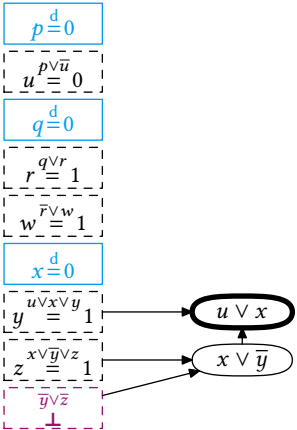
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

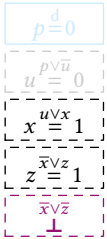
Then continue as before…

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

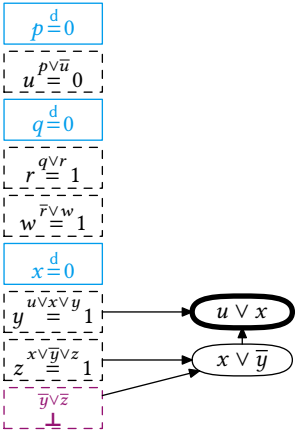


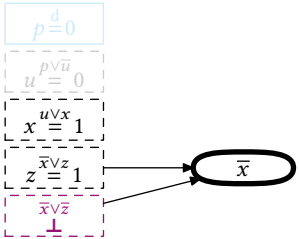


Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

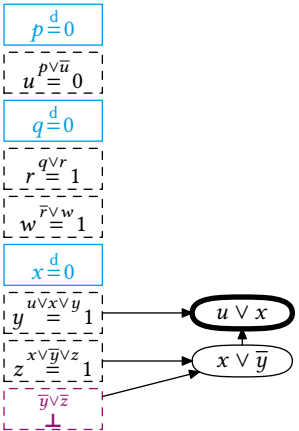


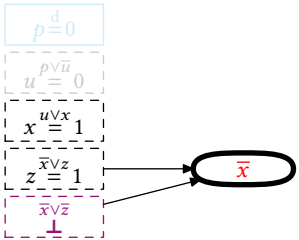


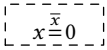
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



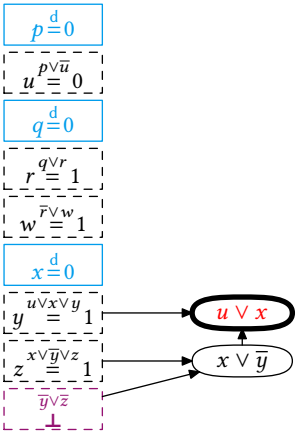


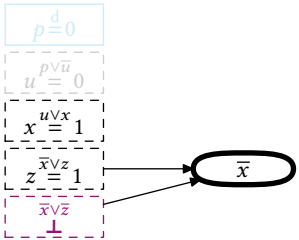


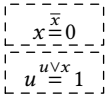
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



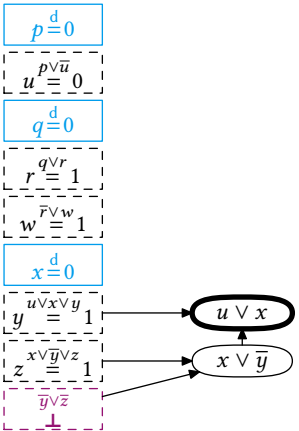


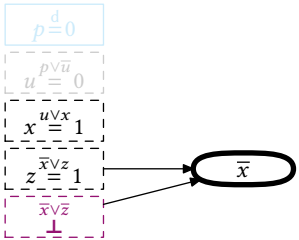


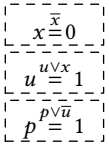
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



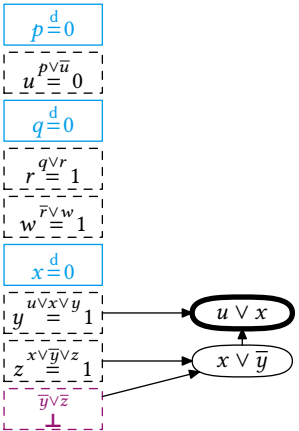


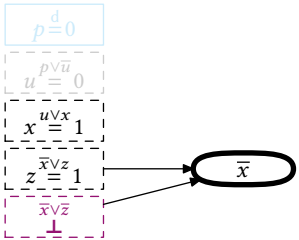


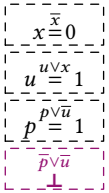
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



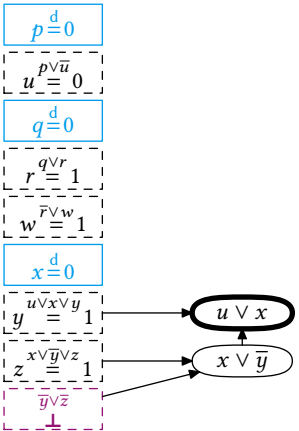


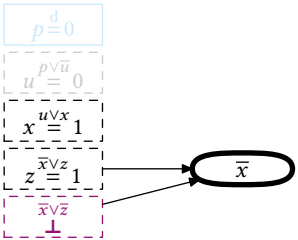


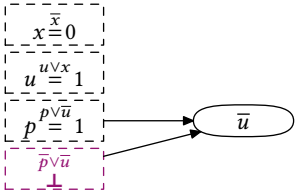
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



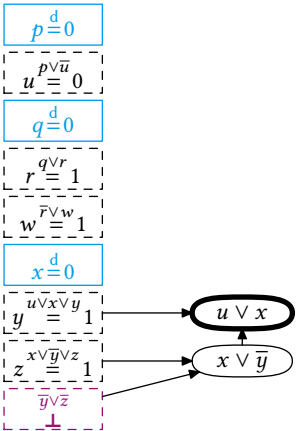


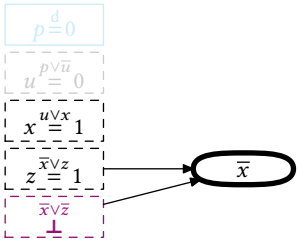


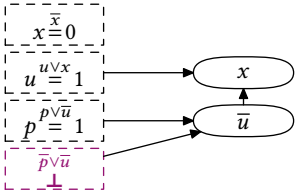
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



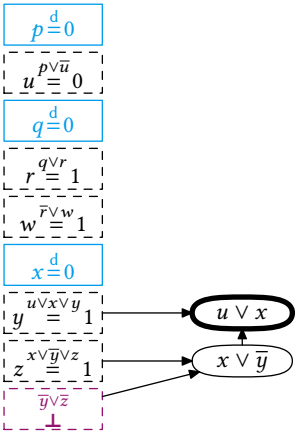


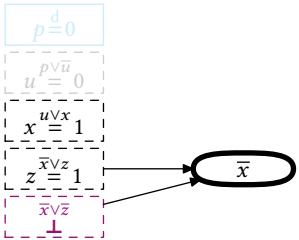


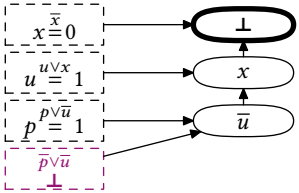
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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







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*

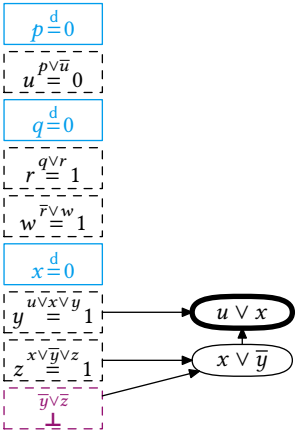
(*) Ignores pre- and inprocessing, but we will get there…

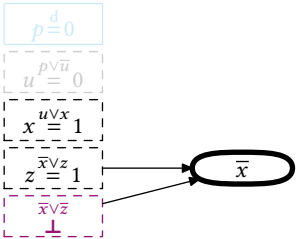
Resolution Proofs from CDCL Executions

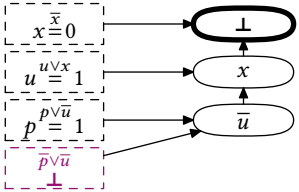
Obtain resolution proof…

Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution…

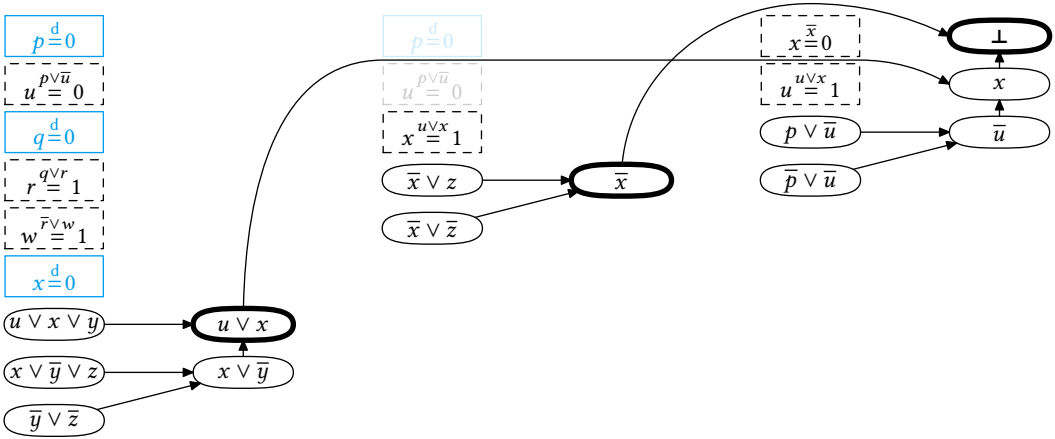






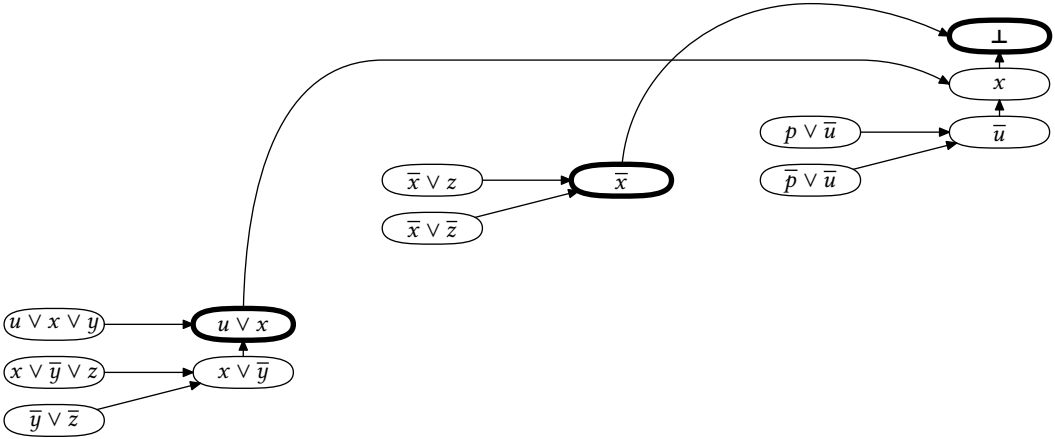
Resolution Proofs from CDCL Executions

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 variable a encoding

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

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

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

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

Extension Variables, Part 1

Suppose we want a variable a encoding

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

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

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

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

Fact

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

Why Aren't We Done?

Practical limitations of current SAT proof logging technology:

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

Why Aren't We Done?

Practical limitations of current SAT proof logging technology:

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

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

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

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

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

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

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

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

Cutting Planes Toy Example

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

$$w + 2x + 2y \geq 3 \quad \text{Divide by 3}$$

Cutting Planes Toy Example

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

$$w + 2x + 2y \geq 3 \quad \text{Divide by 3}$$

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

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

Cutting Planes Toy Example

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

Naming constraints by integers and literal axioms by the literal involved (with $\sim$ 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

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

$$w + 2x + 2y \geq 3 \quad \text{Divide by 3}$$

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

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

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

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

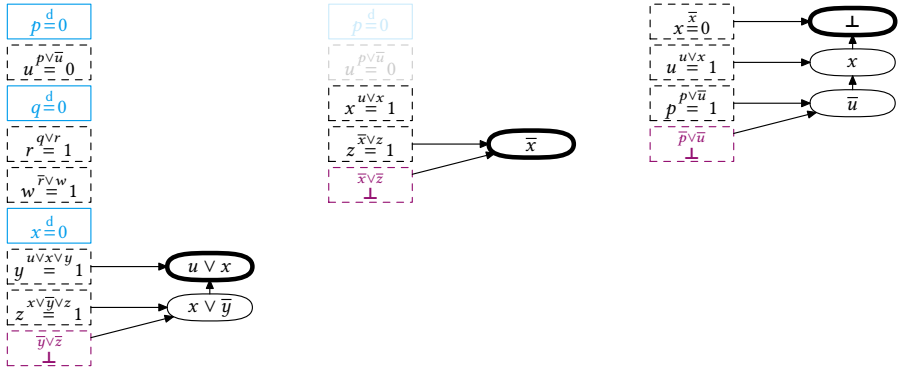
Given that the premises are clauses 7 and 5 in our example CNF formula, using references

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

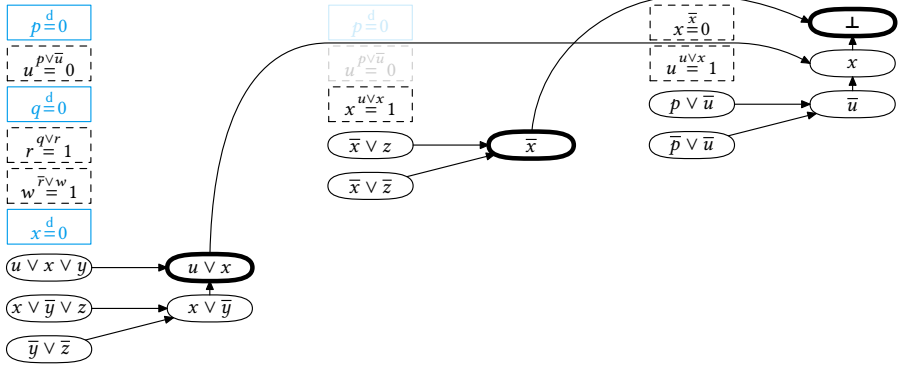
we can write this in the proof log as

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

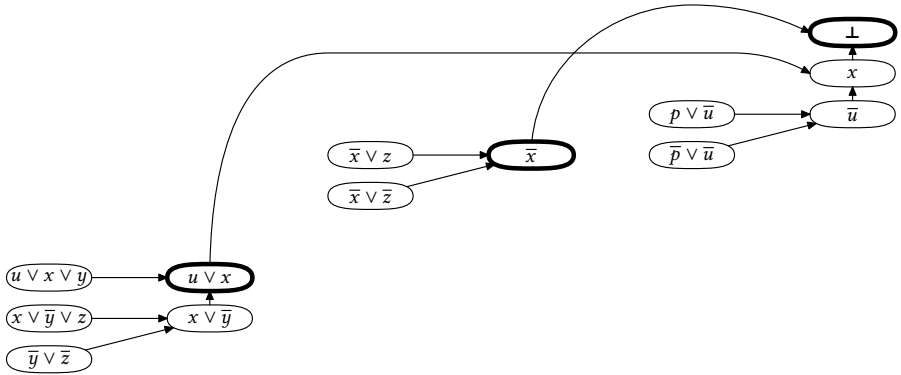
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



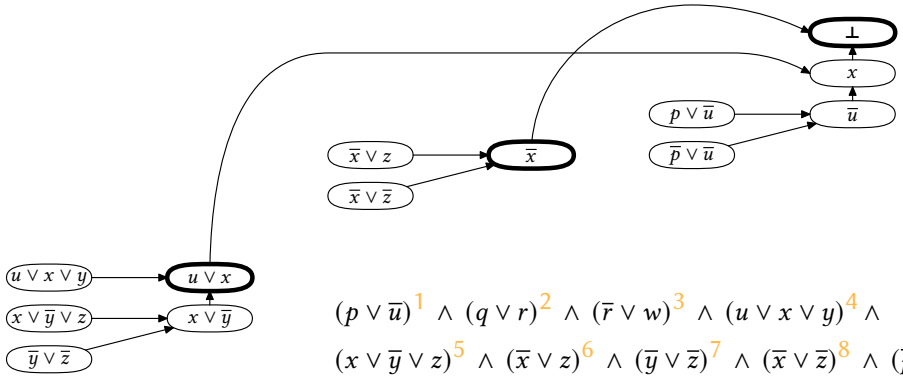
Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses

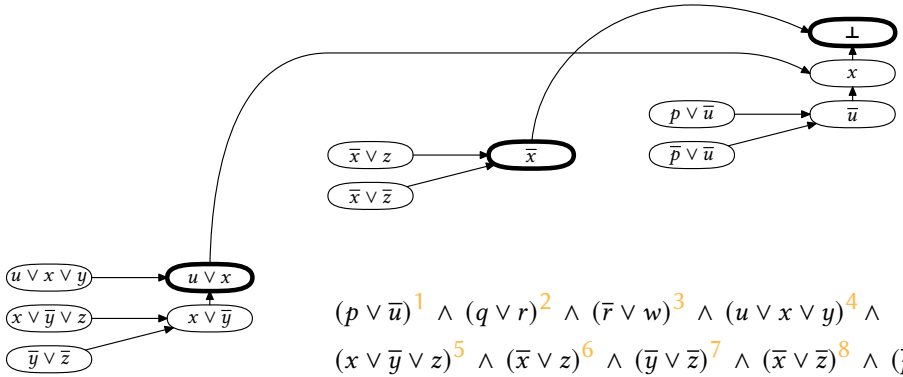


Pseudo-Boolean Proof Logging for Example CDCL Conflict Analyses



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

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,

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

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

RUP Revisited

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

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

E.g., if x_5 is false,

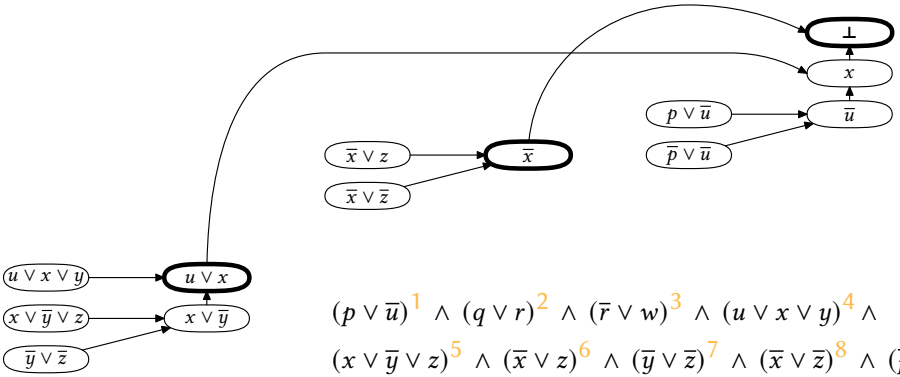
$$x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

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

Risk for confusion!

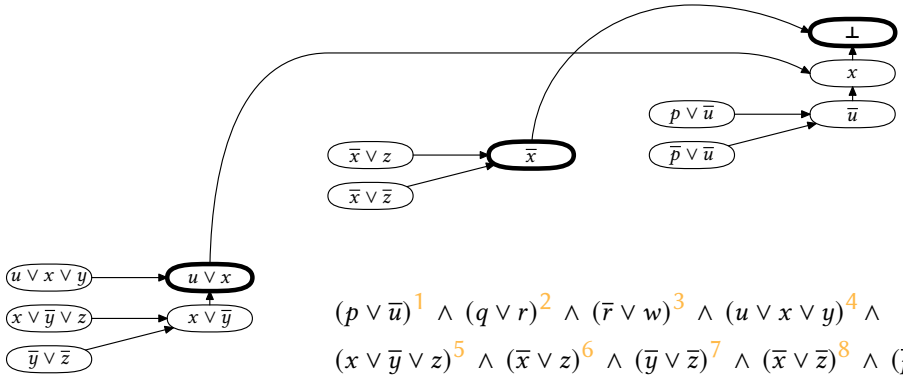
- Constraint programming people might call this (reverse) integer bounds consistency
  - Does the same thing if we're working with clauses
  - More interesting for general pseudo-Boolean constraints
- SAT people beware: constraints can propagate multiple times and multiple variables

Pseudo-Boolean Proof Logging for Example CDCL Execution with RUP



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

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

```
rup 1 ~x >= 1 ;
```
$\rightsquigarrow$ Constraint 11 $\doteq \overline{x} \geq 1$

```
rup >= 1 ;
```
$\rightsquigarrow$ Constraint 12 $\doteq 0 \geq 1$ ↯

Extension Variables, Part 2

Suppose we want new, fresh variable a encoding

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

This time, introduce constraints

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

Again, needs support from the proof system 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 …

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 proofs: significantly more delicate

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

Enumeration and Optimisation Problems

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”

Enumeration and Optimisation Problems

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"

Optimisation:

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

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 — example:

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

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

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]

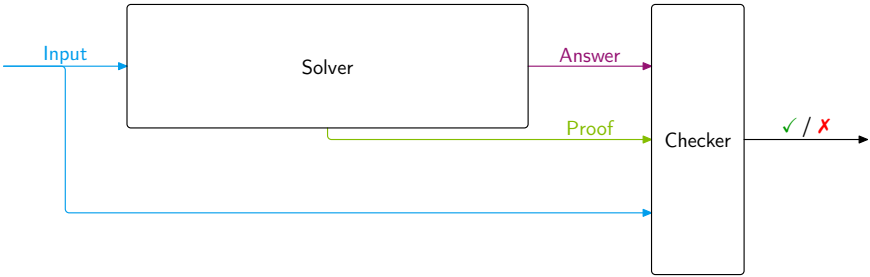
Goldilocks compromise between expressivity and simplicity:

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 — example:

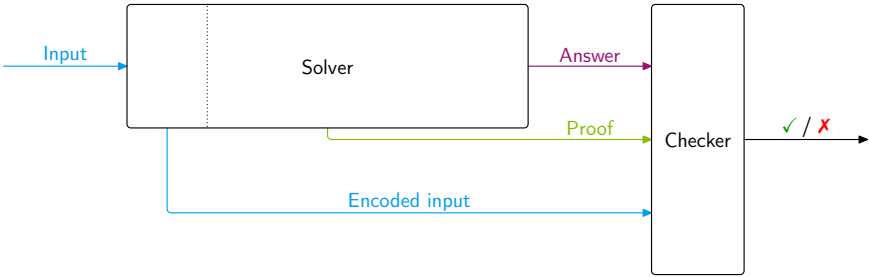
$r \Rightarrow x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$ $\quad$ $7\overline{r} + x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$

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

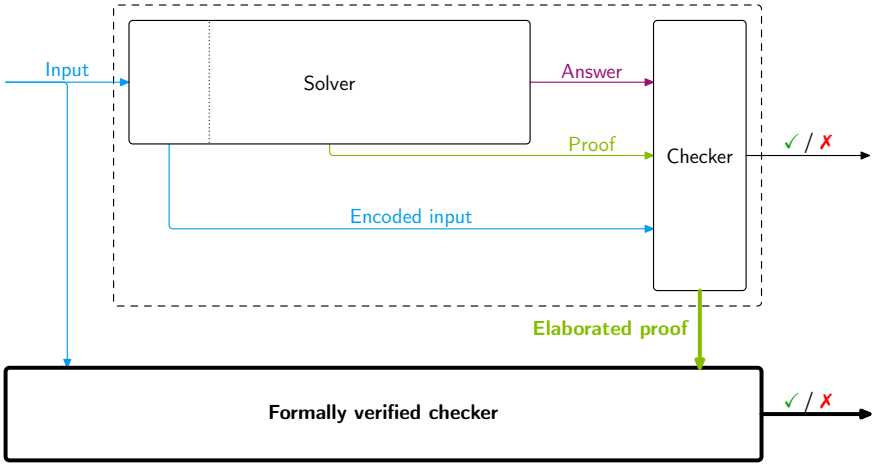
Proof Logging with Formally Verified Checking: Full Workflow



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 a encoding

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

we introduced pseudo-Boolean constraints

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

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

Strengthening Rules (And 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 a encoding

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

we introduced pseudo-Boolean constraints

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

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

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

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 $(F \cup \{C\})\restriction_\omega$, 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 $(F \cup \{C\})\restriction_\omega$, 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

- "obviously" (e.g., by reverse unit propagation) or
- by explicitly presented derivation

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

Want to derive

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

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

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

Want to derive

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

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

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

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

Want to derive

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

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

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

2. $F \cup \{3\overline{a} + 3x + 2y + z + w \geq 3\} \cup \{\neg(5a + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5)\} \models$
   $\big(F \cup \{3\overline{a} + 3x + 2y + z + w \geq 3,\ 5a + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5\}\big)\restriction_\omega$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new constraint satisfied
   $\neg(5a + 3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \geq 5)$ forces $3\overline{x} + 2\overline{y} + \overline{z} + \overline{w} \leq 4$
   This is the same constraint as $3x + 2y + z + w \geq 3$
   And VeriPB can automatically detect this

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 $f\big((\alpha \circ \omega) \circ \omega\big) < f\big(\alpha \circ \omega\big)$

Soundness of Dominance Rule

Dominance-based strengthening (simplified)

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

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

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 $F \wedge C$

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] for details

Strengthening Rules: Proof Format

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: Proof Format

```
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
  - since the proof goal is not affected by the witness substitution ω

Successful Applications of VeriPB Proof Logging

Pseudo-Boolean reasoning with strengthening rules sufficient to certify surprisingly wide range of combinatorial solving paradigms:

Successful Applications of VeriPB Proof Logging

Pseudo-Boolean reasoning with strengthening rules sufficient to certify surprisingly wide range of combinatorial solving paradigms:

1. Boolean satisfiability (SAT) solving including advanced techniques such as
   - Gaussian elimination [GN21]
   - symmetry breaking [BGMN23]
2. SAT-based optimization (MaxSAT) [VDB22, BBN+23, BBN+24, IOT+24]
3. (Linear) Pseudo-Boolean solving [GMNO22, KLM+25]
4. Subgraph solving (max clique, subgraph isomorphism, max common connected subgraph) [GMN20, GMM+20, GMM+24]
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]

VeriPB Resources

VeriPB tutorials

- Slides from tutorials at *CP '22* [BMN22] and *IJCAI '23* [BMN23]
- Video tutorial at `https://youtu.be/s_5BIi4I22w`
- Videos and slides from *WHOOPS '24* at `https://jakobnordstrom.se/WHOOPS24/`
- Videos and slides from *WHOOPS '25* with latest VeriPB updates will hopefully be online soon at `https://jakobnordstrom.se/WHOOPS25/`

VeriPB Resources

VeriPB tutorials

- Slides from tutorials at *CP '22* [BMN22] and *IJCAI '23* [BMN23]
- Video tutorial at `https://youtu.be/s_5BIi4I22w`
- Videos and slides from *WHOOPS '24* at `https://jakobnordstrom.se/WHOOPS24/`
- Videos and slides from *WHOOPS '25* with latest VeriPB updates will hopefully be online soon at `https://jakobnordstrom.se/WHOOPS25/`

Technical documentation [ABB+25] for SAT 2025 competition

- Available at `https://satcompetition.github.io/2025/output.html`

VeriPB Resources

VeriPB tutorials

- Slides from tutorials at *CP '22* [BMN22] and *IJCAI '23* [BMN23]
- Video tutorial at `https://youtu.be/s_5BIi4I22w`
- Videos and slides from *WHOOPS '24* at `https://jakobnordstrom.se/WHOOPS24/`
- Videos and slides from *WHOOPS '25* with latest VeriPB updates will hopefully be online soon at `https://jakobnordstrom.se/WHOOPS25/`

Technical documentation [ABB+25] for SAT 2025 competition

- Available at `https://satcompetition.github.io/2025/output.html`

Specific 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]

VeriPB Resources

VeriPB tutorials

- Slides from tutorials at *CP '22* [BMN22] and *IJCAI '23* [BMN23]
- Video tutorial at `https://youtu.be/s_5BIi4I22w`
- Videos and slides from *WHOOPS '24* at `https://jakobnordstrom.se/WHOOPS24/`
- Videos and slides from *WHOOPS '25* with latest VeriPB updates will hopefully be online soon at `https://jakobnordstrom.se/WHOOPS25/`

Technical documentation [ABB+25] for SAT 2025 competition

- Available at `https://satcompetition.github.io/2025/output.html`

Specific 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]

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

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging and checking

- Trim proof while checking (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Formally verifed end-to-end checking with CakePB (as in [GMM+24, IOT+24, KLM+25])
- Faster proof logging and checking!

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging and checking

- Trim proof while checking (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Formally verifed end-to-end checking with CakePB (as in [GMM+24, IOT+24, KLM+25])
- Faster proof logging and checking!

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 *(work on solvers cvc5, SMTInterpol, Z3, … [BBC+23, HS22])*

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging and checking

- Trim proof while checking (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Formally verifed end-to-end checking with CakePB (as in [GMM+24, IOT+24, KLM+25])
- Faster proof logging and checking!

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 *(work on solvers cvc5, SMTInterpol, Z3, … [BBC+23, HS22])*

And more…

- Use proof logs for algorithm analysis or explainability purposes
- Lots of other challenging problems and interesting ideas

Future Research Directions

Performance and reliability of pseudo-Boolean proof logging and checking

- Trim proof while checking (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format
- Formally verifed end-to-end checking with CakePB (as in [GMM+24, IOT+24, KLM+25])
- Faster proof logging and checking!

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 *(work on solvers cvc5, SMTInterpol, Z3, … [BBC+23, HS22])*

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

Summing up

- 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

- 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
- **Action point:** What problems can VeriPB solve for you?

  Come talk to us. We're hiring and open to collaboration!

Summing up

- Combinatorial solving and 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
- **Action point:** What problems can VeriPB solve for you?

  Come talk to us. We're hiring and open to collaboration!

Thank you for your attention!

References I

[ABB+25] 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. Documentation of VeriPB and CakePB for the SAT competition 2025. Available at `https://satcompetition.github.io/2025/output.html`, April 2025.

[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.

[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.

References II

[BBN+23] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, July 2023.

[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.

[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.

References III

[BMN23] Bart Bogaerts, Ciaran McCreesh, and Jakob Nordström. Combinatorial solving with provably correct results. Tutorial at the *32nd International Joint Conference on Artificial Intelligence*. Slides available at `https://jakobnordstrom.se/presentations/`, August 2023.

[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.

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

[CKSW13] William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, September 2013.

References IV

[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.

[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)*, November 2025. To appear.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[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.

[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.

References V

[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.

[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.

References VI

[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`.

[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.

References VII

[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.

[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.

References VIII

[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.

[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.

[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*.

[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.

References IX

[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.

[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`.

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

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