Combinatorial Solving with Provably Correct Results

Bart Bogaerts

KU Leuven & Vrije Universiteit Brussel

26/02/2025; TCS Seminar @ U. Antwerp

DTAI
Declarative Languages & Artificial Intelligence

KU LEUVEN

VUB

ARTIFICIAL INTELLIGENCE RESEARCH GROUP

erc

ABOUT THIS TALK

- Based on (small port of) tutorial co-developed with Ciaran McCreesh and Jakob Nordström
- Useful material: `https://www.bartbogaerts.eu/talks/veripb-tutorial-series`

OUTLINE

1. Introduction
   1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
   2. Ensuring Correctness with the Help of Proof Logging
   3. This Seminar
2. Proof Logging for SAT
   1. SAT Basics
   2. DPLL and CDCL
   3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
   1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
   2. Pseudo-Boolean Proof Logging for SAT Solving
   3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
   1. Future Work
   2. Concluding Remarks

OUTLINE

1. Introduction
   1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
   2. Ensuring Correctness with the Help of Proof Logging
   3. This Seminar
2. Proof Logging for SAT
   1. SAT Basics
   2. DPLL and CDCL
   3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
   1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
   2. Pseudo-Boolean Proof Logging for SAT Solving
   3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
   1. Future Work
   2. Concluding Remarks

COMBINATORIAL SOLVING AND OPTIMISATION

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

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

WHAT CAN BE DONE ABOUT SOLVER BUGS?

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

WHAT CAN BE DONE ABOUT SOLVER BUGS?

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

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

WHAT CAN BE DONE ABOUT SOLVER BUGS?

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

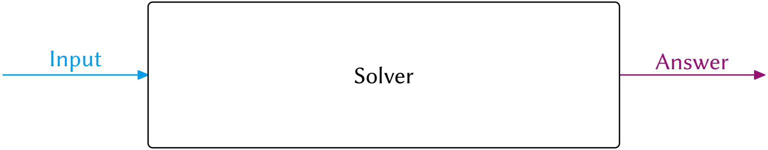
- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to this level of complexity

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

OUTLINE

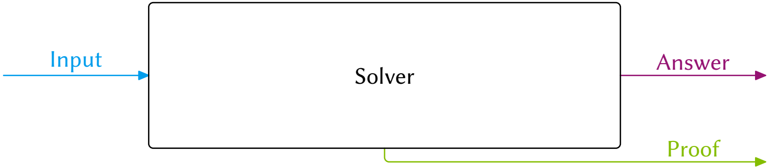
1. Introduction
    1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
    2. Ensuring Correctness with the Help of Proof Logging
    3. This Seminar
2. Proof Logging for SAT
    1. SAT Basics
    2. DPLL and CDCL
    3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
    1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
    2. Pseudo-Boolean Proof Logging for SAT Solving
    3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
    1. Future Work
    2. Concluding Remarks

PROOF LOGGING WITH CERTIFYING SOLVERS: WORKFLOW



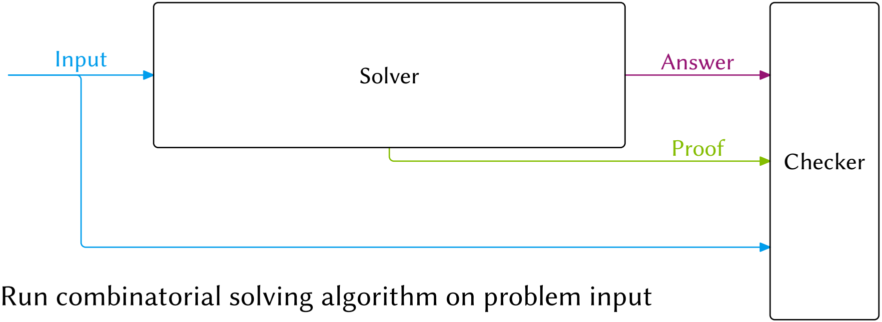
1. Run combinatorial solving algorithm on problem input

PROOF LOGGING WITH CERTIFYING SOLVERS: WORKFLOW



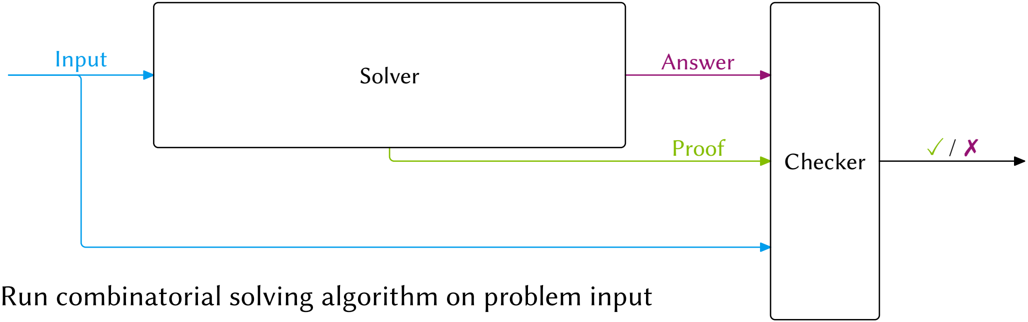
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

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

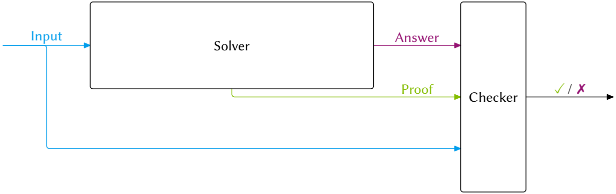
PROOF LOGGING WITH CERTIFYING SOLVERS: WORKFLOW



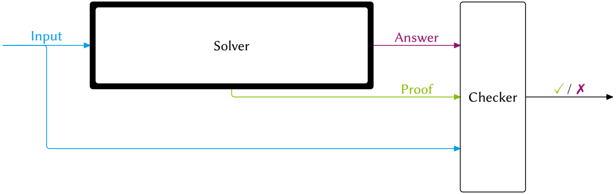
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

PROOF LOGGING DESIDERATA

Proof format for certifying solver should be



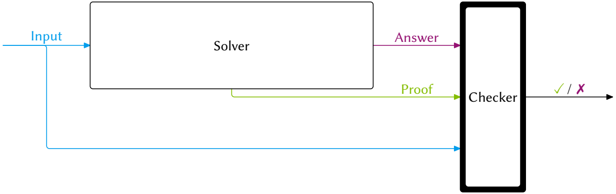
PROOF LOGGING DESIDERATA



Proof format for certifying solver should be

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

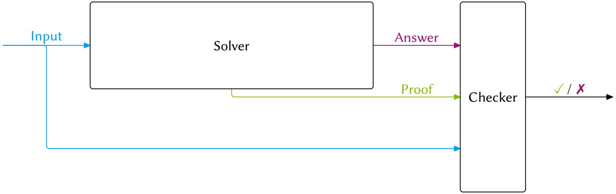
PROOF LOGGING DESIDERATA



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

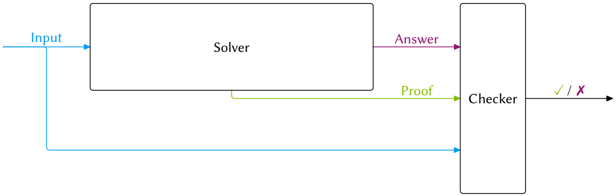


Proof format for certifying solver should be

- **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?

OUTLINE

1. Introduction
    1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
    2. Ensuring Correctness with the Help of Proof Logging
    3. This Seminar
2. Proof Logging for SAT
    1. SAT Basics
    2. DPLL and CDCL
    3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
    1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
    2. Pseudo-Boolean Proof Logging for SAT Solving
    3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
    1. Future Work
    2. Concluding Remarks

TAKE-AWAY MESSAGE

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

TAKE-AWAY MESSAGE

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

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

THE SALES PITCH FOR PROOF LOGGING

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

PROOF LOGGING WITH VERIPB

In extended version of this tutorial :

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

in a unified way

PROOF LOGGING WITH VERIPB

In extended version of this tutorial :

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

in a unified way

This seminar:

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

OUTLINE

1. Introduction
    1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
    2. Ensuring Correctness with the Help of Proof Logging
    3. This Seminar
2. Proof Logging for SAT
    1. SAT Basics
    2. DPLL and CDCL
    3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
    1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
    2. Pseudo-Boolean Proof Logging for SAT Solving
    3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
    1. Future Work
    2. Concluding Remarks

OUTLINE

1. Introduction
    1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
    2. Ensuring Correctness with the Help of Proof Logging
    3. This Seminar
2. Proof Logging for SAT
    1. SAT Basics
    2. DPLL and CDCL
    3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
    1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
    2. Pseudo-Boolean Proof Logging for SAT Solving
    3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
    1. Future Work
    2. Concluding Remarks

THE SAT PROBLEM

- Variable x: takes value **true** $(=1)$ or **false** $(=0)$
- Literal ℓ: variable x or its negation $\overline{x}$
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: 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 $\perp$)
- Means original formula must be inconsistent

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

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

Proof checker should know how to unit propagate until saturation

OUTLINE

DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

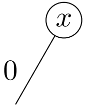
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

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

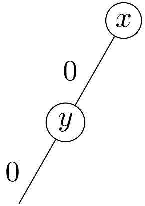


DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

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

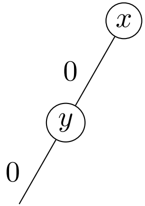


DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

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



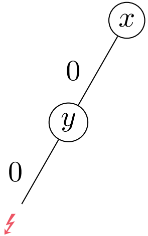
DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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. $x \vee y$



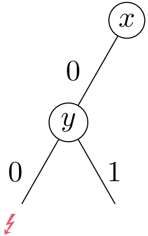
DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

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

1. $x \lor y$



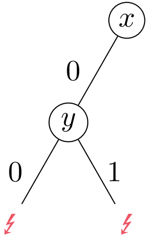
DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

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

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



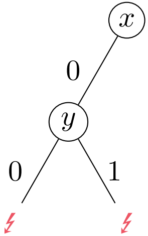
DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

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

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



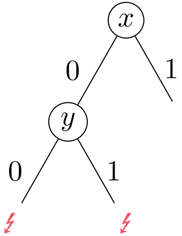
DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

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

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



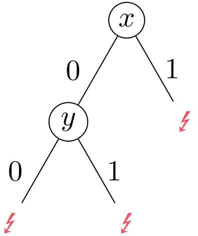
DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

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

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



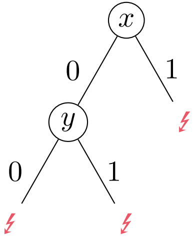
DAVIS-PUTMAN-LOGEMANN-LOVELAND (DPLL)

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

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

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

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



REVERSE UNIT PROPAGATION (RUP)

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

REVERSE UNIT PROPAGATION (RUP)

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

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

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

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

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

REVERSE UNIT PROPAGATION (RUP)

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

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

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

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

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

Fact

Backtrack clauses from DPLL solver generate a RUP proof

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

Decision

Free choice to assign value to variable

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

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

Decision

Free choice to assign value to variable

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

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \stackrel{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ 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:

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

$\boxed{p \stackrel{\mathrm{d}}{=} 0}$
$u \stackrel{p \lor \overline{u}}{=} 0$
decision level 1

$\boxed{q \stackrel{\mathrm{d}}{=} 0}$
$r \stackrel{q \lor r}{=} 1$
$w \stackrel{\overline{r} \lor w}{=} 1$
decision level 2

$\boxed{x \stackrel{\mathrm{d}}{=} 0}$
$y \stackrel{u \lor x \lor y}{=} 1$
$z \stackrel{x \lor \overline{y} \lor z}{=} 1$
$\stackrel{\overline{y} \lor \overline{z}}{\bot}$
decision level 3

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Add to assignment trail

Continue until satisfying assignment or conflict

CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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

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

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

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

CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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

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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

decision level 3

Could backtrack by erasing conflict level & flipping last decision

CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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

$\boxed{p \stackrel{\mathrm{d}}{=} 0}$
$u \stackrel{p \lor \overline{u}}{=} 0$
} decision level 1

$\boxed{q \stackrel{\mathrm{d}}{=} 0}$
$r \stackrel{q \lor r}{=} 1$
$w \stackrel{\overline{r} \lor w}{=} 1$
} decision level 2

$\boxed{x \stackrel{\mathrm{d}}{=} 0}$
$y \stackrel{u \lor x \lor y}{=} 1$
$z \stackrel{x \lor \overline{y} \lor z}{=} 1$
$\stackrel{\overline{y} \lor \overline{z}}{\bot}$
} decision level 3

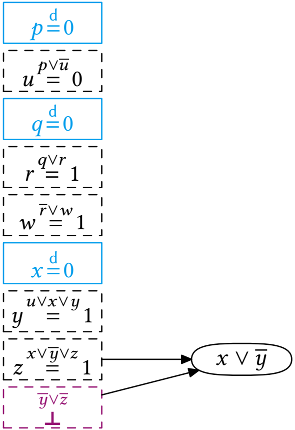
Could backtrack by erasing conflict level & flipping last decision

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

CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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



Could backtrack by erasing conflict level & flipping last decision

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

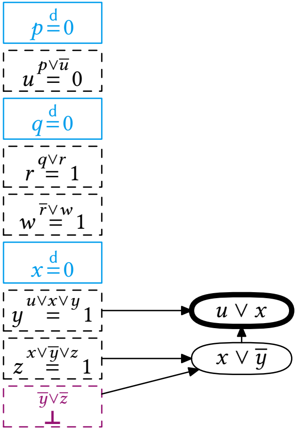
Case analysis over z for last two clauses:

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

CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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



Could backtrack by erasing conflict level & flipping last decision

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

Case analysis over z for last two clauses:

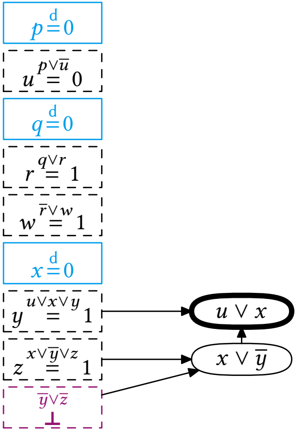
- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Resolve clauses by merging them & removing z — must satisfy $x \lor \overline{y}$

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

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

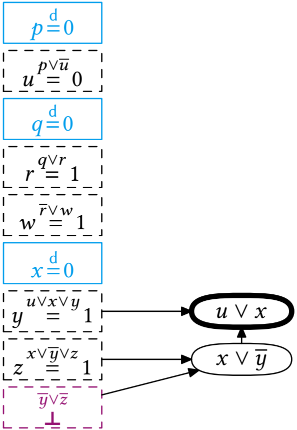
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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

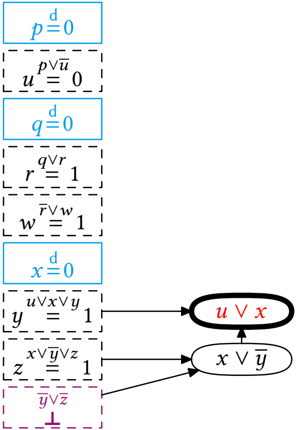


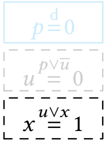
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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





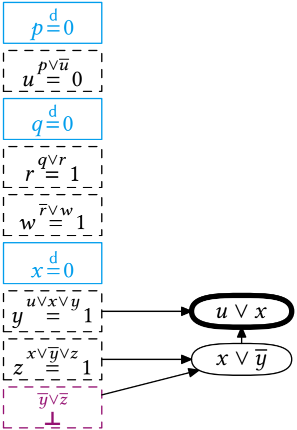
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

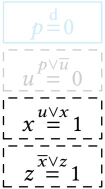
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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





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

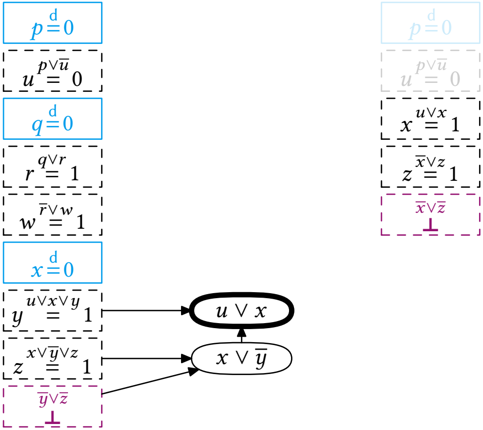
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Then continue as before...

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

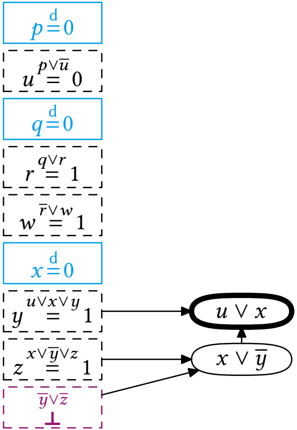
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

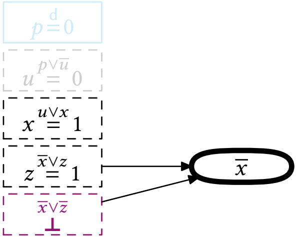


COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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

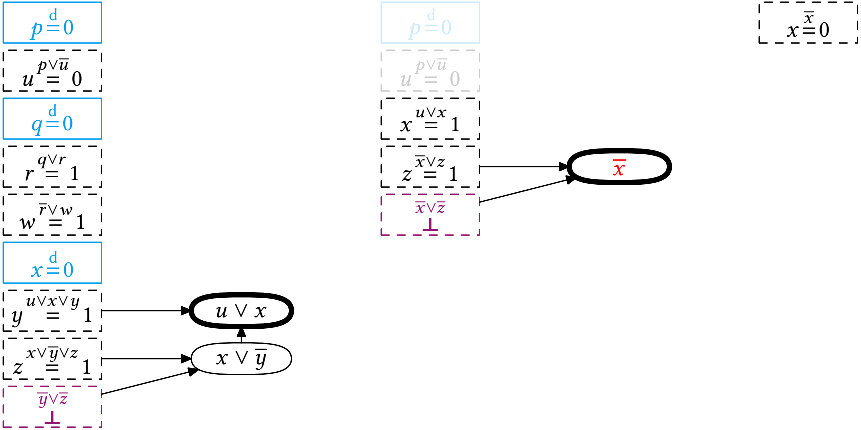




COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

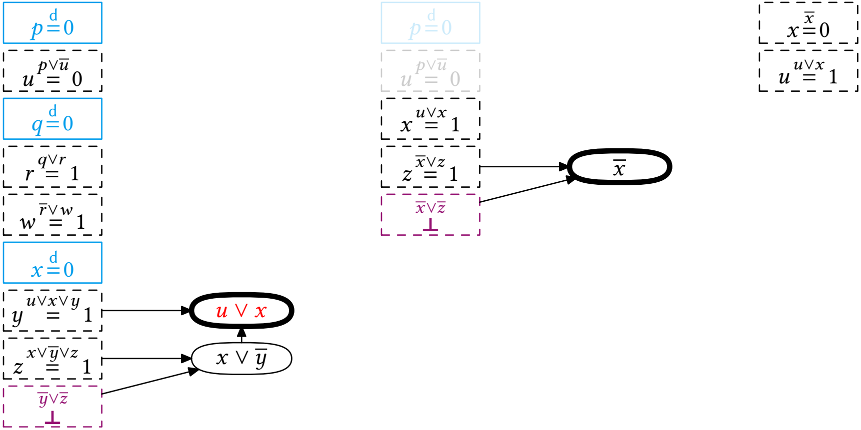
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

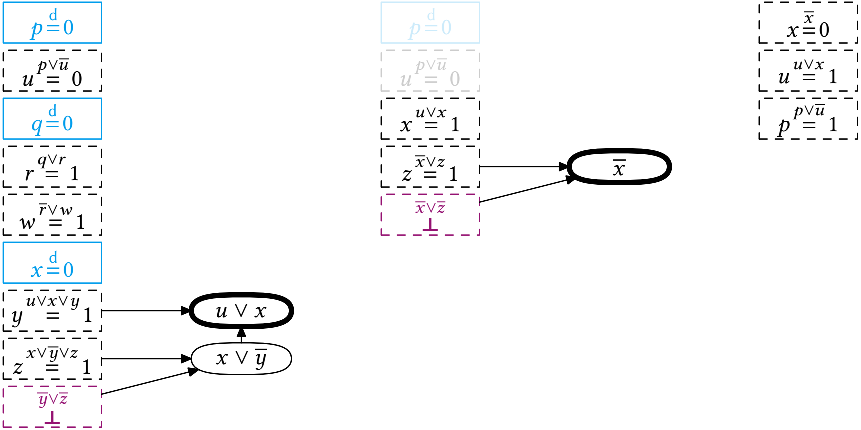
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

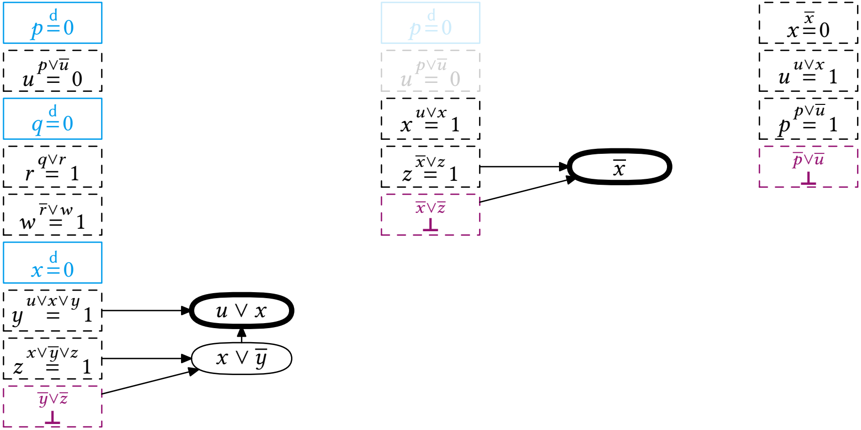
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

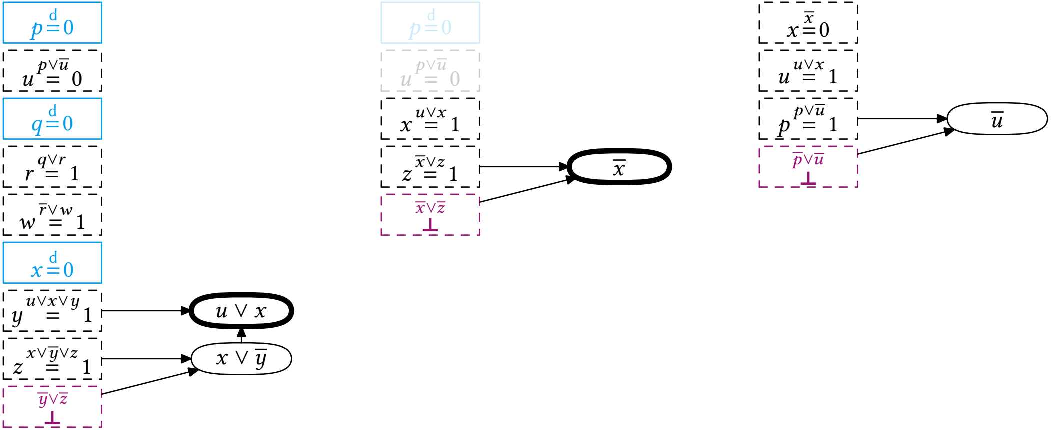
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

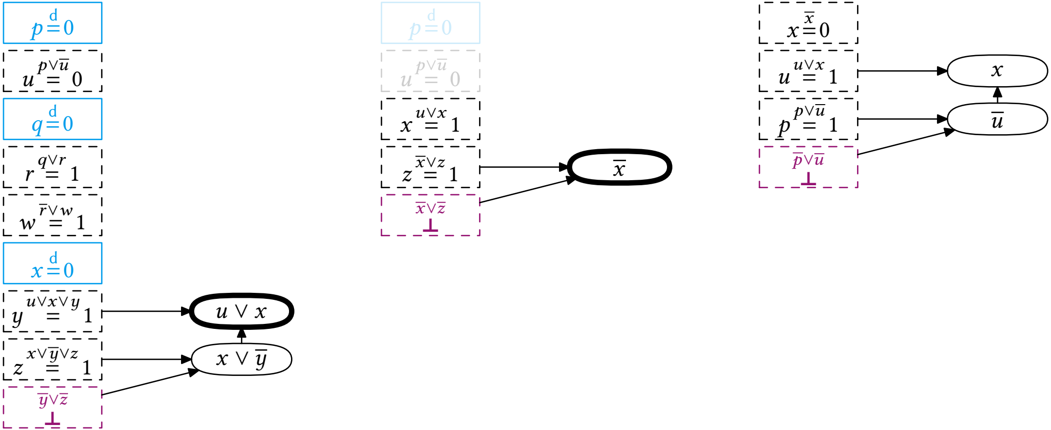
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

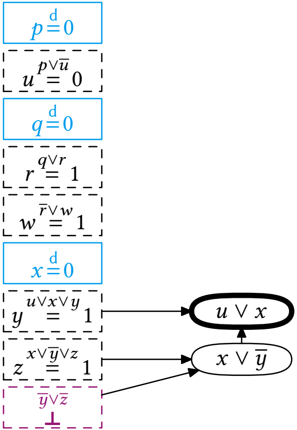
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

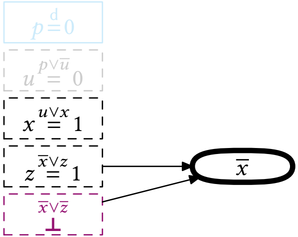


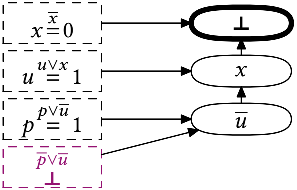
COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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







OUTLINE

1. Introduction
   1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
   2. Ensuring Correctness with the Help of Proof Logging
   3. This Seminar
2. Proof Logging for SAT
   1. SAT Basics
   2. DPLL and CDCL
   3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
   1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
   2. Pseudo-Boolean Proof Logging for SAT Solving
   3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
   1. Future Work
   2. Concluding Remarks

CDCL REASONING AND THE RESOLUTION PROOF SYSTEM

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

CDCL REASONING AND THE RESOLUTION PROOF SYSTEM

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

Resolution proof system [Bla37, Rob65]

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

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

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

CDCL REASONING AND THE RESOLUTION PROOF SYSTEM

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

Resolution proof system [Bla37, Rob65]

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

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

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

When run on unsatisfiable formula, CDCL generates resolution proof*

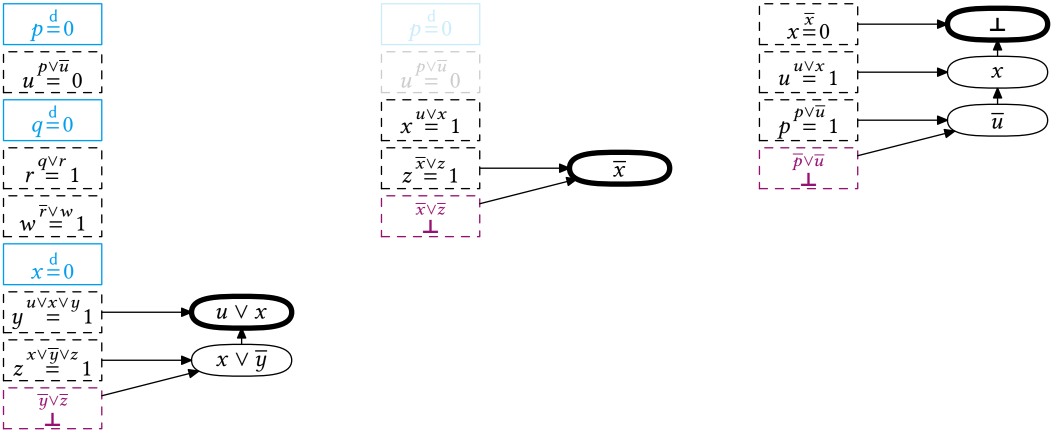
(*) Ignores pre- and inprocessing, but we will get there...

RESOLUTION PROOFS FROM CDCL EXECUTIONS

Obtain resolution proof…

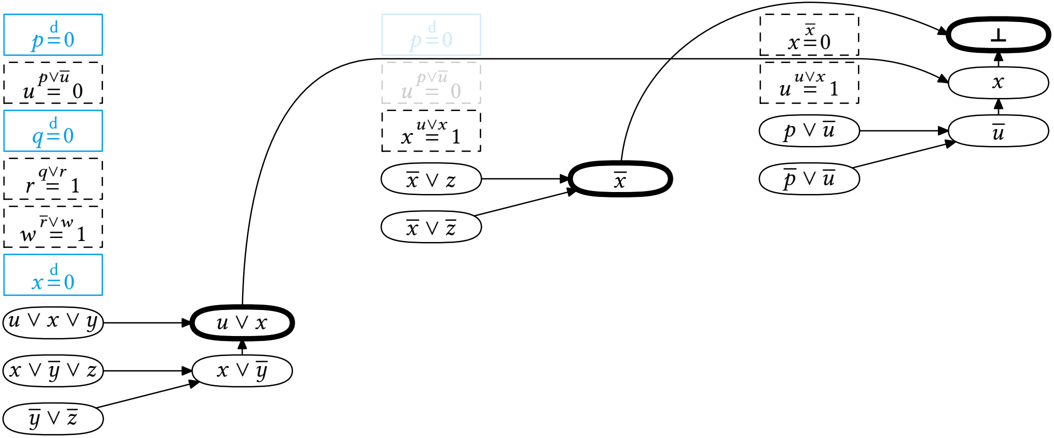
RESOLUTION PROOFS FROM CDCL EXECUTIONS

Obtain resolution proof from our example CDCL execution…



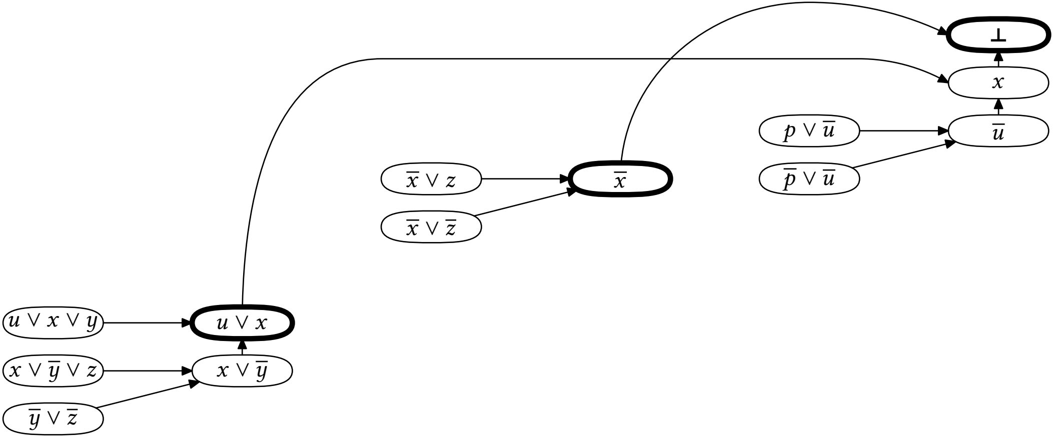
RESOLUTION PROOFS FROM CDCL EXECUTIONS

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



RESOLUTION PROOFS FROM CDCL EXECUTIONS

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



RUP PROOFS AND CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP PROOFS AND CDCL

But it turns out we can be lazier...

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP PROOFS AND CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP PROOFS AND CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP PROOFS AND CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

But it turns out we can be lazier...

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP PROOFS AND CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP PROOFS AND CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP PROOFS AND CDCL

But it turns out we can be lazier...

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP PROOFS AND CDCL

But it turns out we can be lazier…

Fact

All learned clauses generated by CDCL solver are RUP clauses

So shorter short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

MORE INGREDIENTS IN PROOF LOGGING FOR SAT

Fact

RUP proofs can be viewed as shorthand for resolution proofs

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

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

EXTENSION VARIABLES, PART 1

Suppose we want a variable a encoding

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

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

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

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

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

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

EXTENSION VARIABLES, PART 1

Suppose we want a variable a encoding

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

Extended resolution [Tse68]

Resolution rule plus extension rule introducing clauses

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

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

Fact

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

OUTLINE

WHY AREN'T WE DONE?

Practical limitations of current SAT proof logging technology:

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

WHY AREN'T WE DONE?

Practical limitations of current SAT proof logging technology:

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

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

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

OUTLINE

PSEUDO-BOOLEAN CONSTRAINTS

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

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

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

PSEUDO-BOOLEAN CONSTRAINTS

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

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

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

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

SOME TYPES OF PSEUDO-BOOLEAN CONSTRAINTS

1. Clauses

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

2. Cardinality constraints

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

3. General pseudo-Boolean constraints

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

Input/model axioms

From the input

PSEUDO-BOOLEAN REASONING: CUTTING PLANES

Input/model axioms From the input

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

PSEUDO-BOOLEAN REASONING: CUTTING PLANES

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

Input/model axioms From the input

Literal axioms

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

Addition

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

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

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

PSEUDO-BOOLEAN REASONING: CUTTING PLANES

Input/model axioms — From the input

Literal axioms

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

Addition

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

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

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

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

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

CUTTING PLANES TOY EXAMPLE

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

CUTTING PLANES TOY EXAMPLE

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

CUTTING PLANES TOY EXAMPLE

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

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

CUTTING PLANES TOY EXAMPLE

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

$$\text{Add}\quad \overline{\qquad 3w + 6x + 6y + 2z \geq 9 \qquad}$$

CUTTING PLANES TOY EXAMPLE

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

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

CUTTING PLANES TOY EXAMPLE

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

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

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

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

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

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

CUTTING PLANES TOY EXAMPLE

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

CUTTING PLANES TOY EXAMPLE

Multiply by 2

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

Add

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

Multiply by 2

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

Add

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

Divide by 3

$$\frac{3w + 6x + 6y \geq 7}{w + 2x + 2y \geq 2\tfrac{1}{3}}$$

CUTTING PLANES TOY EXAMPLE

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

CUTTING PLANES TOY EXAMPLE

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

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

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

CUTTING PLANES TOY EXAMPLE

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

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

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

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

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

OUTLINE

1. Introduction
   1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
   2. Ensuring Correctness with the Help of Proof Logging
   3. This Seminar
2. Proof Logging for SAT
   1. SAT Basics
   2. DPLL and CDCL
   3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
   1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
   2. Pseudo-Boolean Proof Logging for SAT Solving
   3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
   1. Future Work
   2. Concluding Remarks

RESOLUTION AND CUTTING PLANES

To simulate resolution step such as

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

we can perform the cutting planes steps

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

RESOLUTION AND CUTTING PLANES

To simulate resolution step such as

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

we can perform the cutting planes steps

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

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

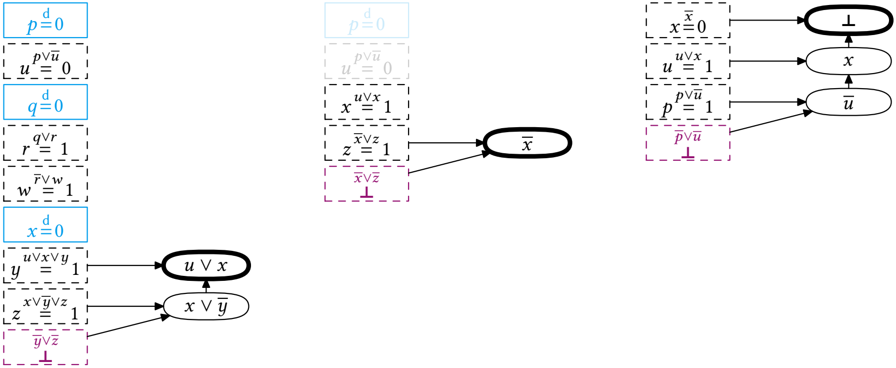
$$\text{Constraint 7} \doteq \overline{y} + \overline{z} \geq 1$$

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

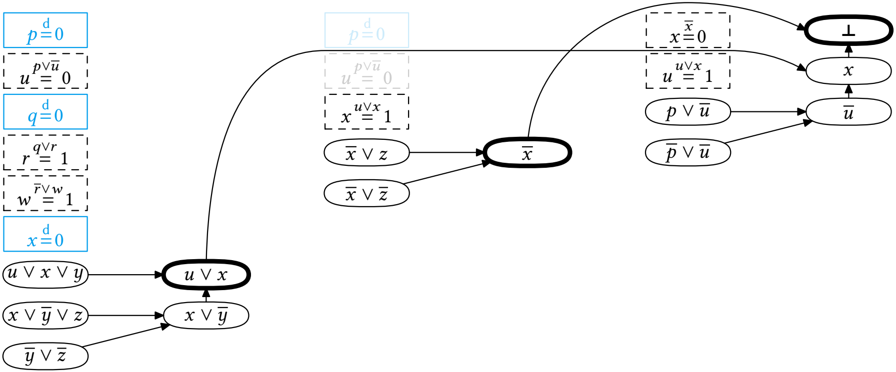
we can write this in the proof log as

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

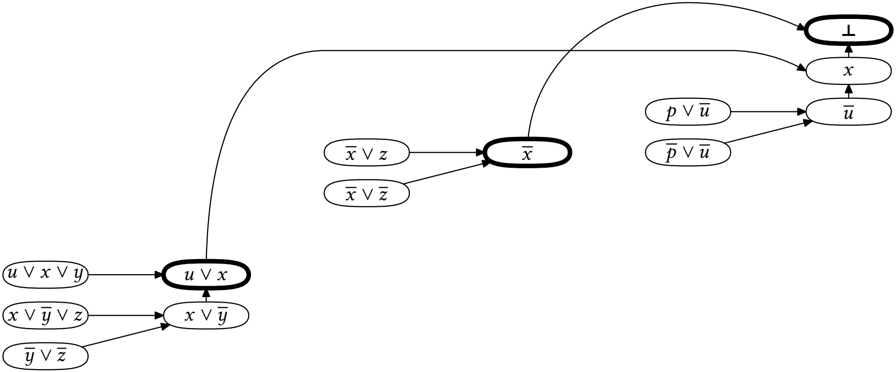
PSEUDO-BOOLEAN PROOF LOGGING FOR CDCL EXAMPLE



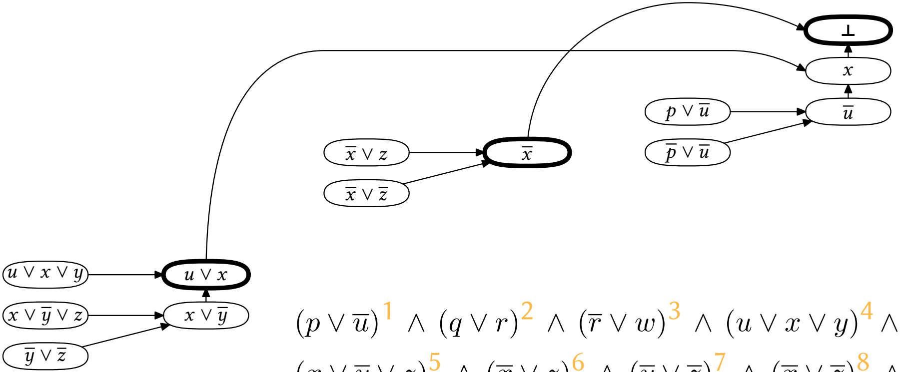
PSEUDO-BOOLEAN PROOF LOGGING FOR CDCL EXAMPLE



PSEUDO-BOOLEAN PROOF LOGGING FOR CDCL EXAMPLE

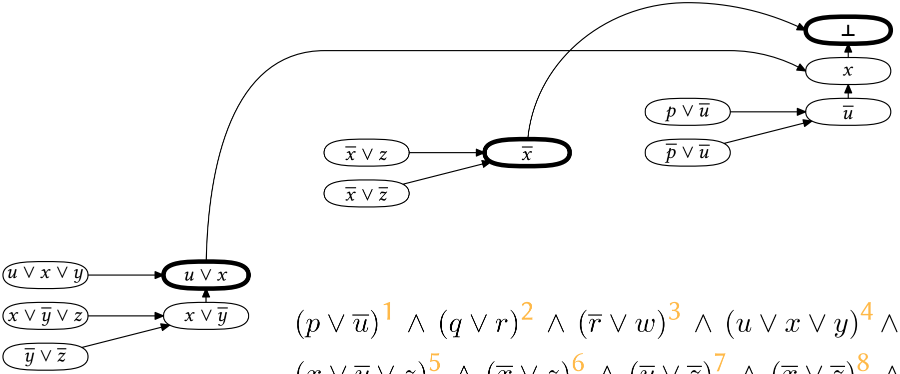


PSEUDO-BOOLEAN PROOF LOGGING FOR CDCL EXAMPLE



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

PSEUDO-BOOLEAN PROOF LOGGING FOR CDCL EXAMPLE



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

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

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

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

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

RUP REVISITED

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

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

E.g., if x_5 is false,

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

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

RUP REVISITED

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

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

E.g., if x_5 is false,

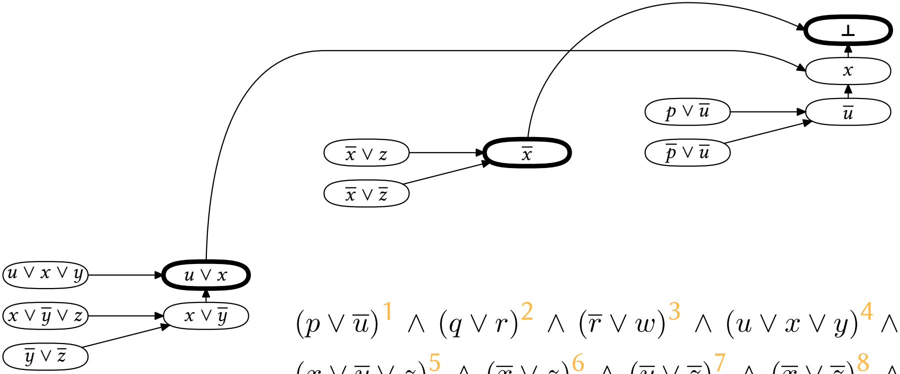
$$x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

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

Risk for confusion:

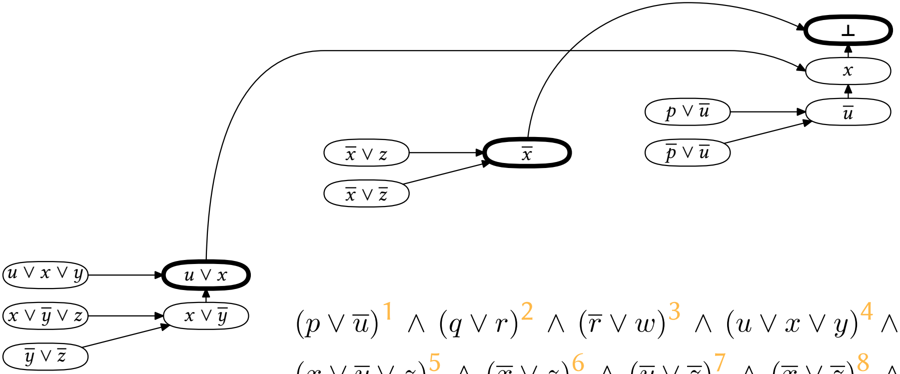
- Constraint programming people might call this (reverse) integer bounds consistency
  - Does the same thing if we're working with clauses
  - More interesting for general pseudo-Boolean constraints
- SAT people beware: constraints can propagate multiple times and multiple variables

PB PROOF LOGGING FOR EXAMPLE CDCL EXECUTION WITH RUP



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

PB PROOF LOGGING FOR EXAMPLE CDCL EXECUTION WITH RUP



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

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

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

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

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

OUTLINE

1. Introduction
    1. The Success of Combinatorial Solving (and the Dirty Little Secret…)
    2. Ensuring Correctness with the Help of Proof Logging
    3. This Seminar
2. Proof Logging for SAT
    1. SAT Basics
    2. DPLL and CDCL
    3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
    1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
    2. Pseudo-Boolean Proof Logging for SAT Solving
    3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
    1. Future Work
    2. Concluding Remarks

EXTENSION VARIABLES, PART 2

Suppose we want new, fresh variable a encoding

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

This time, introduce constraints

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

Again, needs support from the proof system

PROOF LOGS FOR “EXTENDED CUTTING PLANES”

For satisfiable instances: just specify a satisfying assignment.

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

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

PROOF LOGS FOR "EXTENDED CUTTING PLANES"

For satisfiable instances: just specify a satisfying assignment.

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

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

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

DELETING CONSTRAINTS

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

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

So ignored in this tutorial for simplicity and clarity

ENUMERATION AND OPTIMISATION PROBLEMS

Enumeration:

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

ENUMERATION AND OPTIMISATION PROBLEMS

Enumeration:

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

For optimisation:

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

PSEUDO-BOOLEAN PROOF LOGGING — HOW AND WHY?

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

- just do proof logging

PSEUDO-BOOLEAN PROOF LOGGING — HOW AND WHY?

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

- just do proof logging

Otherwise

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

PSEUDO-BOOLEAN PROOF LOGGING — HOW AND WHY?

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

- just do proof logging

Otherwise

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

Proof logging philosophy:

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

PSEUDO-BOOLEAN PROOF LOGGING — HOW AND WHY?

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

- just do proof logging

Otherwise

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

Proof logging philosophy:

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

Goldilocks compromise between expressivity and simplicity:

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

PSEUDO-BOOLEAN PROOF LOGGING — HOW AND WHY?

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

- just do proof logging

Otherwise

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

Proof logging philosophy:

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

Goldilocks compromise between expressivity and simplicity:

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

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

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

PSEUDO-BOOLEAN PROOF LOGGING — HOW AND WHY?

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

- just do proof logging

Otherwise

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

Proof logging philosophy:

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

Goldilocks compromise between expressivity and simplicity:

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

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

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

THE VERIPB FORMAT AND TOOL

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

Released under MIT Licence

Various features to help development:

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

Documentation:

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

OUTLINE

1. Introduction
   1. The Success of Combinatorial Solving (and the Dirty Little Secret…)
   2. Ensuring Correctness with the Help of Proof Logging
   3. This Seminar
2. Proof Logging for SAT
   1. SAT Basics
   2. DPLL and CDCL
   3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
   1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
   2. Pseudo-Boolean Proof Logging for SAT Solving
   3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
   1. Future Work
   2. Concluding Remarks

OUTLINE

1. Introduction
   1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
   2. Ensuring Correctness with the Help of Proof Logging
   3. This Seminar
2. Proof Logging for SAT
   1. SAT Basics
   2. DPLL and CDCL
   3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
   1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
   2. Pseudo-Boolean Proof Logging for SAT Solving
   3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
   1. Future Work
   2. Concluding Remarks

FUTURE RESEARCH DIRECTIONS

Performance of pseudo-Boolean proof logging

- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format

FUTURE RESEARCH DIRECTIONS

Performance of pseudo-Boolean proof logging

- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format

Proof logging for other combinatorial problems and techniques

- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(some work on SCIP in [CGS17, EG21])*
- Satisfiability modulo theories (SMT) solving *(some work by Bjørner and others)*
- High-level modelling languages

FUTURE RESEARCH DIRECTIONS

Performance of pseudo-Boolean proof logging

- Trim proof while verifying (as in *DRAT-Trim* [HHW13a])
- Compress proof file using binary format

Proof logging for other combinatorial problems and techniques

- Symmetric learning and recycling (substitution) of subproofs
- Mixed integer linear programming *(some work on SCIP in [CGS17, EG21])*
- Satisfiability modulo theories (SMT) solving *(some work by Bjørner and others)*
- High-level modelling languages

And more…

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

OUTLINE

1. Introduction
   1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
   2. Ensuring Correctness with the Help of Proof Logging
   3. This Seminar
2. Proof Logging for SAT
   1. SAT Basics
   2. DPLL and CDCL
   3. Proof System for SAT Proof Logging
3. Pseudo-Boolean Proof Logging
   1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
   2. Pseudo-Boolean Proof Logging for SAT Solving
   3. More Pseudo-Boolean Proof Logging Rules
4. Conclusions
   1. Future Work
   2. Concluding Remarks

SUMMING UP

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity

SUMMING UP

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity

Thanks for your attention!

SUMMING UP

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like most promising approach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity

Thanks for your attention!

Interested? I'm hiring: looking for PhD students (vacancy deadline March 6th)
`https://www.kuleuven.be/personeel/jobsite/jobs/60425822`

REFERENCES

[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.

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

[BGMN23] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI '22*.

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

REFERENCES

[Bla37] Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

[BLB10] Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, July 2010.

[BMM+23] Bart Bogaerts, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. Documentation of VeriPB and CakePB for the SAT competition 2023. Available at `https://satcompetition.github.io/2023/checkers.html`, March 2023.

[BMN22] Bart Bogaerts, Ciaran McCreesh, and Jakob Nordström. Solving with provably correct results: Beyond satisfiability, and towards constraint programming. Tutorial at the *28th International Conference on Principles and Practice of Constraint Programming*. Slides available at `http://www.jakobnordstrom.se/presentations/`, August 2022.

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

[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.

REFERENCES

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

[CGS17] Kevin K. H. Cheung, Ambros M. Gleixner, and Daniel E. Steffy. Verifying integer programming results. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO '17)*, volume 10328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, June 2017.

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

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

[CMS17] Luís Cruz-Filipe, João P. Marques-Silva, and Peter Schneider-Kamp. Efficient certified resolution proof checking. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 118–135. Springer, April 2017.

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

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

REFERENCES

[EG21] Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. In *Proceedings of the 22nd International Conference on Integer Programming and Combinatorial Optimization (IPCO '21)*, volume 12707 of *Lecture Notes in Computer Science*, pages 163–177. Springer, May 2021.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, February 2020.

[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.

[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

[GN03] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, pages 886–891, March 2003.

[GN21] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.

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

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

[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.

REFERENCES

[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.

[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.

[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.

[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*.

[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.

REFERENCES

[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.

ACKNOWLEDGEMENTS

Co-funded by the FWO Flanders (project G070521N) and by the European Union (ERC, CertiFOX, 101122653). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.