

An Introduction to Proof Logging

(Certifying Algorithms for Combinatorial Search & Optimization)

Bart Bogaerts

KU Leuven

May 28, 2026; CODES seminar



KU LEUVEN



ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP



ACKNOWLEDGEMENT

Partly based on a tutorial co-developed with Ciaran McCreesh and Jakob Nordström, and on work with many others



1. Introduction

1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
2. Ensuring Correctness with the Help of Proof Logging
3. Advantages of Proof Logging

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. Taking a step back

1. Recent Work
2. Outlook



1. Introduction

1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
2. Ensuring Correctness with the Help of Proof Logging
3. Advantages of Proof Logging

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. Taking a step back

1. Recent Work
2. Outlook



COMBINATORIAL SOLVING AND OPTIMISATION

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

¹See end of slides for all references with bibliographic details

WHAT CAN BE DONE ABOUT SOLVER BUGS?

- ▶ **Software testing**

Hard to get good test coverage for sophisticated solvers

Inherently can only detect presence of bugs, not absence

WHAT CAN BE DONE ABOUT SOLVER BUGS?

- ▶ **Software testing**

 - Hard to get good test coverage for sophisticated solvers

 - Inherently can only detect presence of bugs, not absence

- ▶ **Formal verification**

 - Prove that solver implementation adheres to formal specification

 - Current techniques cannot scale to this level of complexity

WHAT CAN BE DONE ABOUT SOLVER BUGS?

▶ **Software testing**

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

▶ **Formal verification**

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

▶ **Proof logging**

Make solver **certifying** [ABM⁺11, MMNS11] by outputting

1. not only **answer** but also
2. simple, machine-verifiable **proof** that answer is correct

1. Introduction

1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
2. Ensuring Correctness with the Help of Proof Logging
3. Advantages of Proof Logging

2. Proof Logging for SAT

1. SAT Basics
2. DPLL and CDCL
3. Proof System for SAT Proof Logging

3. Pseudo-Boolean Proof Logging

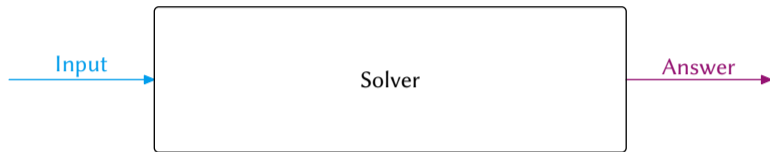
1. Pseudo-Boolean Constraints and Cutting Planes Reasoning
2. Pseudo-Boolean Proof Logging for SAT Solving
3. More Pseudo-Boolean Proof Logging Rules

4. Taking a step back

1. Recent Work
2. Outlook

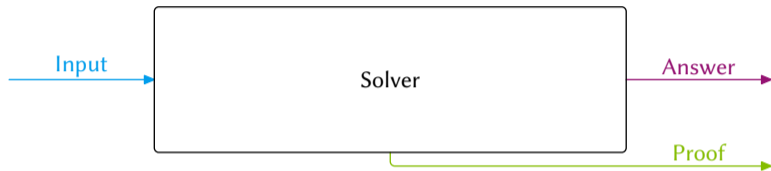


PROOF LOGGING WITH CERTIFYING SOLVERS: WORKFLOW



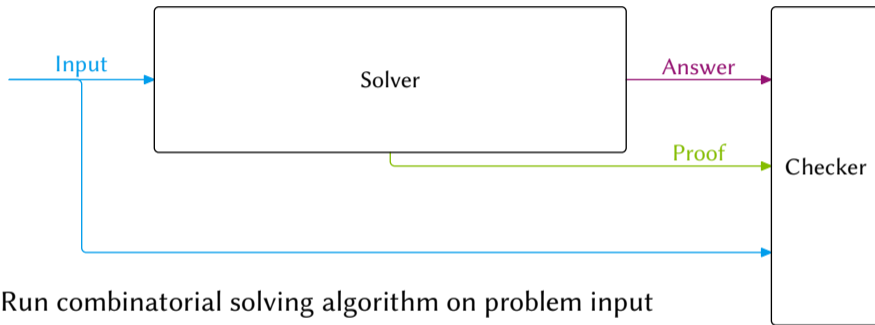
1. Run combinatorial solving algorithm on problem input

PROOF LOGGING WITH CERTIFYING SOLVERS: WORKFLOW



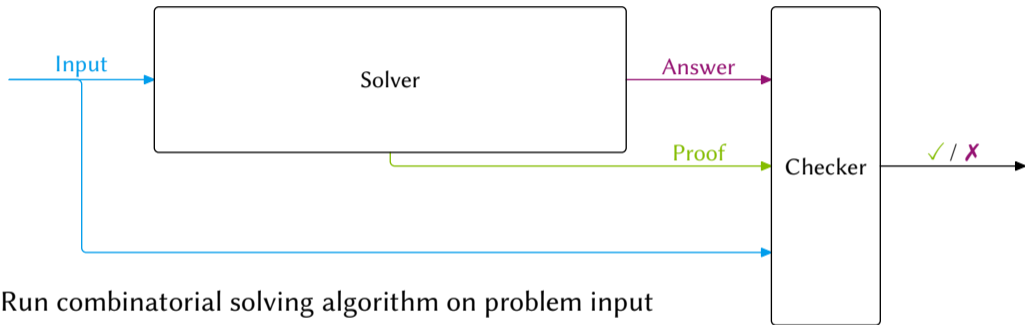
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

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

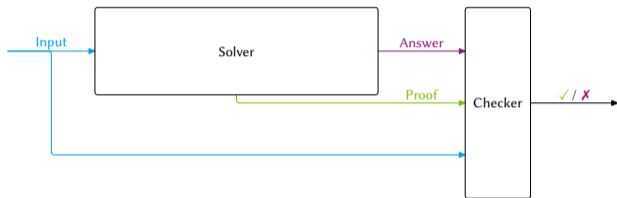
PROOF LOGGING WITH CERTIFYING SOLVERS: WORKFLOW



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

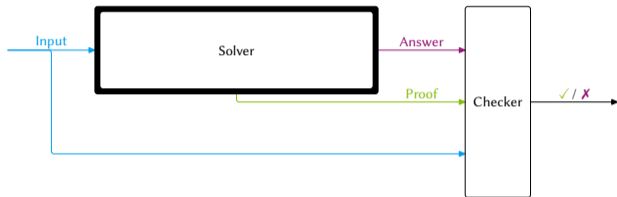
Proof format for certifying solver should be



PROOF LOGGING DESIDERATA

Proof format for certifying solver
should be

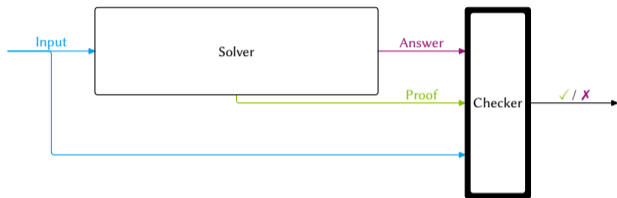
- ▶ **very powerful:** minimal overhead for sophisticated reasoning



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

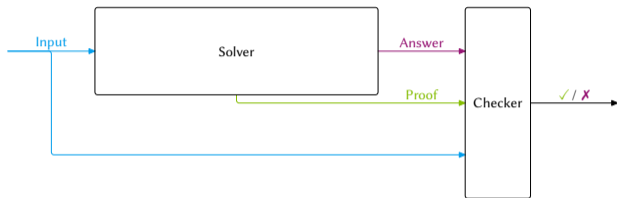


PROOF LOGGING DESIDERATA

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!



1. Introduction

1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
2. Ensuring Correctness with the Help of Proof Logging
3. Advantages of Proof Logging

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. Taking a step back

1. Recent Work
2. Outlook



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 [BFB⁺26]



THE ELEPHANT IN THE ROOM

"This all sounds like someone who is interested in exact solvers; it is of no value to me who is interested in heuristic algorithms."

Focus has indeed been on exact solvers, but...

- ▶ ... can certify bounds for optimisation problems
- ▶ ... can certify correctness of pre-processing methods
 - ▶ does the method accidentally introduce better than-optimal solutions?
 - ▶ does the method remove all/scene optimal solutions?
- ▶ Opportunities for future work?

THE ELEPHANT IN THE ROOM

“This all sounds like someone who is interested in exact solvers; it is of no value to me who is interested in heuristic algorithms.”

THE ELEPHANT IN THE ROOM

“This all sounds like someone who is interested in exact solvers; it is of no value to me who is interested in heuristic algorithms.”

Focus has indeed been on **exact** solvers, but...

- ▶ ...can certify **bounds** for optimisation problems

THE ELEPHANT IN THE ROOM

“This all sounds like someone who is interested in exact solvers; it is of no value to me who is interested in heuristic algorithms.”

Focus has indeed been on **exact** solvers, but...

- ▶ ...can certify **bounds** for optimisation problems
- ▶ ...can certify **correctness** of pre-processing methods
 - ▶ does the method accidentally introduce better-than-optimal solutions?
 - ▶ does the method remove all/some optimal solutions?

THE ELEPHANT IN THE ROOM

“This all sounds like someone who is interested in exact solvers; it is of no value to me who is interested in heuristic algorithms.”

Focus has indeed been on **exact** solvers, but...

- ▶ ...can certify **bounds** for optimisation problems
- ▶ ...can certify **correctness** of pre-processing methods
 - ▶ does the method accidentally introduce better-than-optimal solutions?
 - ▶ does the method remove all/some optimal solutions?
- ▶ Opportunities for future work?

WHAT'S NEXT?

(Technical) introduction about:

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

WHAT'S NEXT?

(Technical) introduction about:

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

Overview of some of our recent work

1. Introduction
 1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
 2. Ensuring Correctness with the Help of Proof Logging
 3. Advantages of Proof Logging
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. Taking a step back
 1. Recent Work
 2. Outlook



1. Introduction
 1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
 2. Ensuring Correctness with the Help of Proof Logging
 3. Advantages of Proof Logging
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. Taking a step back
 1. Recent Work
 2. Outlook



THE SAT PROBLEM

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

The SAT Problem

Given a CNF formula F , is it satisfiable?

For instance, what about:

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

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

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

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

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

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

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

► $p \vee \bar{u}$ propagates $u \mapsto 0$

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

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

WHAT IS OBVIOUS? UNIT PROPAGATION

Unit Propagation

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

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

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

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

Proof checker should know how to unit propagate until saturation

1. Introduction
 1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
 2. Ensuring Correctness with the Help of Proof Logging
 3. Advantages of Proof Logging
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. Taking a step back
 1. Recent Work
 2. Outlook



DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

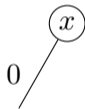
$$(p \vee \bar{u}) \wedge (q \vee r) \wedge (\bar{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{p} \vee \bar{u})$$

DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

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

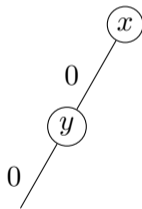


DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

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

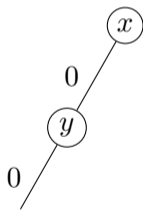


DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

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



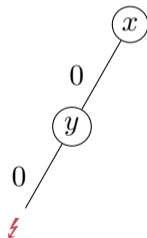
DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

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

1. $x \vee y$



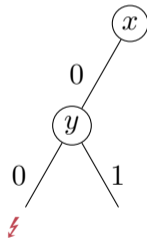
DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

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

1. $x \vee y$



DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

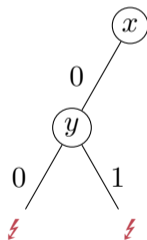
DPLL [DP60, DLL62]: Assign variables and propagate; backtrack when clause violated

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

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

1. $x \vee y$

2. $x \vee \bar{y}$



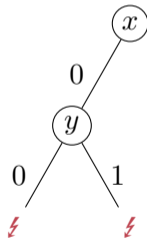
DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

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

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



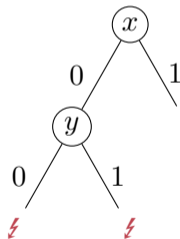
DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

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

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



DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

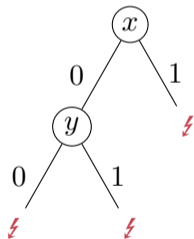
$$(p \vee \bar{u}) \wedge (q \vee r) \wedge (\bar{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{p} \vee \bar{u})$$

1. $x \vee y$

2. $x \vee \bar{y}$

3. x

4. \bar{x}



DAVIS-PUTNAM-LOGEMANN-LOVELAND (DPLL)

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

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

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

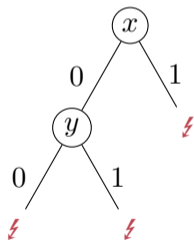
1. $x \vee y$

2. $x \vee \bar{y}$

3. x

4. \bar{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:

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

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

Decision

Free choice to assign value to variable

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

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

Decision

Free choice to assign value to variable

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

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \stackrel{p \vee \bar{u}}{=} 0$ ($p \vee \bar{u}$ is **reason clause**)

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \stackrel{p \vee \bar{u}}{=} 0$ ($p \vee \bar{u}$ is **reason clause**)

WHAT ABOUT CONFLICT-DRIVEN CLAUSE LEARNING (CDCL)?

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z}$$



Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z}$$

⊥

decision
level 1

Decision

Free choice to assign value to variable

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

decision
level 2

Unit propagation

Forced choice to avoid falsifying clause

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

Notation $u \stackrel{p \vee \bar{u}}{=} 0$ ($p \vee \bar{u}$ is **reason clause**)

decision
level 3

Always propagate if possible, otherwise decide

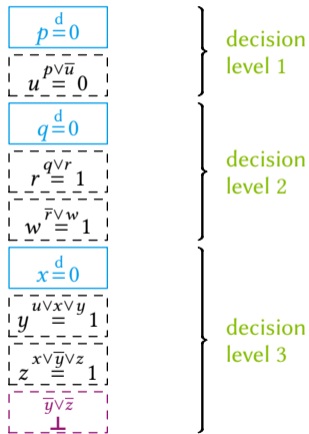
Add to assignment **trail**

Continue until satisfying assignment or **conflict**

CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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



CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z} \\ \perp$$

decision
level 1

decision
level 2

decision
level 3

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

CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z} \\ \perp$$

decision
level 1

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

decision
level 2

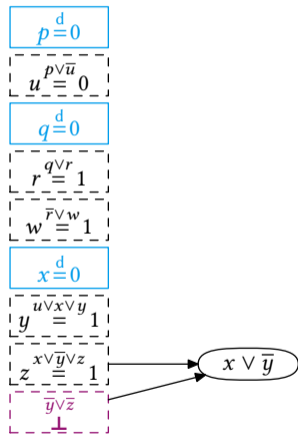
But want to **learn** from conflict and cut away as much of search space as possible

decision
level 3

CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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



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

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

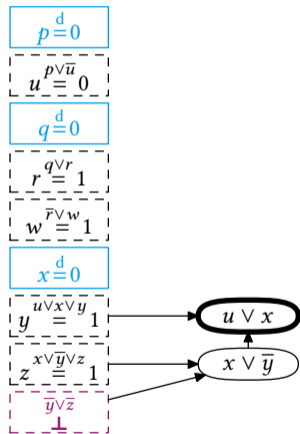
Case analysis over z for last two clauses:

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

CONFLICT ANALYSIS

Time to analyse this conflict and learn from it!

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



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

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

Case analysis over z for last two clauses:

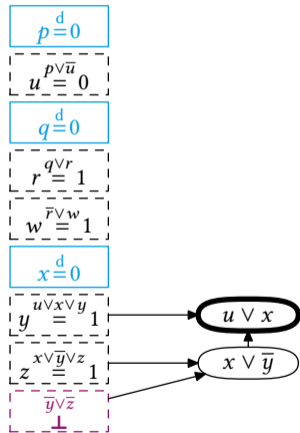
- ▶ $x \vee \bar{y} \vee z$ wants $z = 1$
- ▶ $\bar{y} \vee \bar{z}$ wants $z = 0$
- ▶ **Resolve** clauses by merging them & removing z — must satisfy $x \vee \bar{y}$

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

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z}$$

$$u \vee x$$

$$x \vee \bar{y}$$

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

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

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

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z}$$

$$\perp$$

$$u \vee x$$

$$x \vee \bar{y}$$

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

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

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

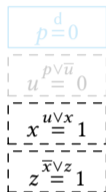
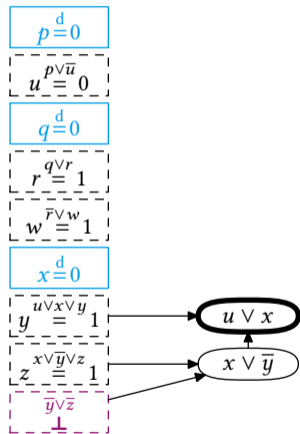
Assertion level 1 (2nd largest level in learned clause) – trim trail to that level

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

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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



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

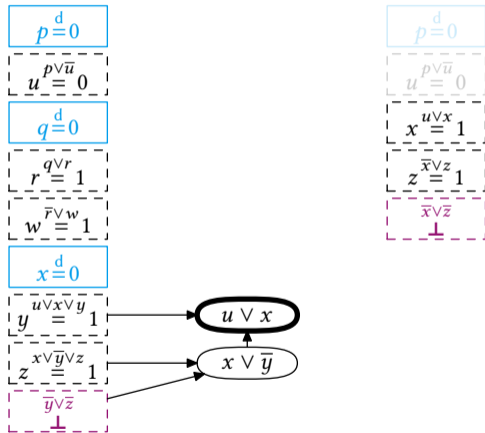
Now UIP literal guaranteed to flip (**assert**) – but this is a **propagation**, not a decision

Then continue as before...

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

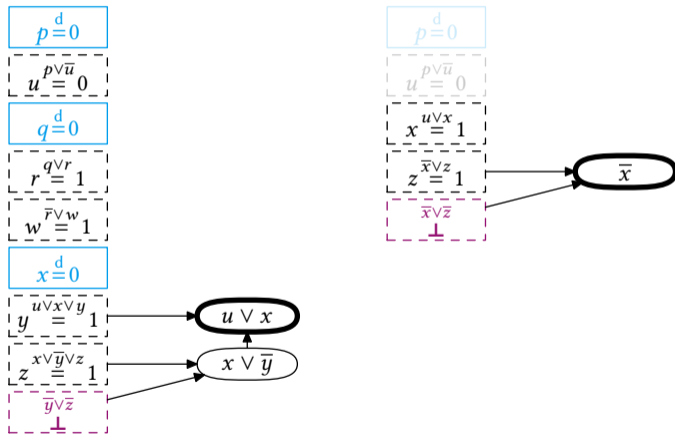
$$(p \vee \bar{u}) \wedge (q \vee r) \wedge (\bar{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{p} \vee \bar{u})$$



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z} \perp$$

$$u \vee x$$

$$x \vee \bar{y}$$

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

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

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

$$z \stackrel{\bar{x} \vee z}{=} 1$$

$$\bar{x} \vee \bar{z} \perp$$

 \bar{x}

$$x \stackrel{\bar{x}}{=} 0$$

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z} \perp$$

$$u \vee x$$

$$x \vee \bar{y}$$

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

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

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

$$z \stackrel{\bar{x} \vee z}{=} 1$$

$$\bar{x} \vee \bar{z} \perp$$

$$\bar{x}$$

$$x \stackrel{\bar{x}}{=} 0$$

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

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z} \perp$$

$$u \vee x$$

$$x \vee \bar{y}$$

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

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

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

$$z \stackrel{\bar{x} \vee z}{=} 1$$

$$\bar{x} \vee \bar{z} \perp$$

$$\bar{x}$$

$$x \stackrel{\bar{x}}{=} 0$$

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

$$p \stackrel{p \vee \bar{u}}{=} 1$$

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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

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

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

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

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

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

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

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

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

$$\bar{y} \vee \bar{z} \perp$$

$$u \vee x$$

$$x \vee \bar{y}$$

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

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

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

$$z \stackrel{\bar{x} \vee z}{=} 1$$

$$\bar{x} \vee \bar{z} \perp$$

 \bar{x}

$$x \stackrel{\bar{x}}{=} 0$$

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

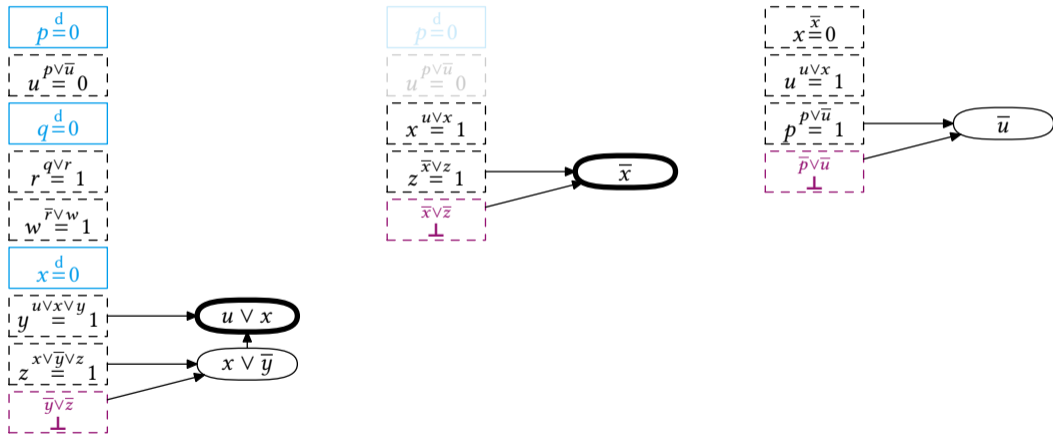
$$p \stackrel{p \vee \bar{u}}{=} 1$$

$$\bar{p} \vee \bar{u} \perp$$

COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

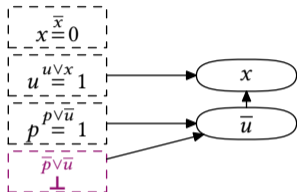
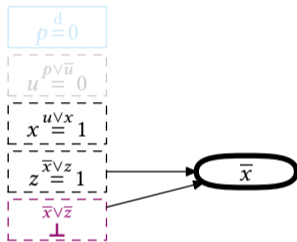
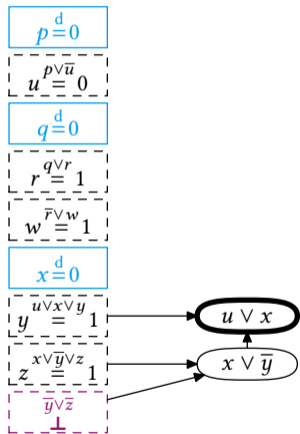
$$(p \vee \bar{u}) \wedge (q \vee r) \wedge (\bar{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{p} \vee \bar{u})$$



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

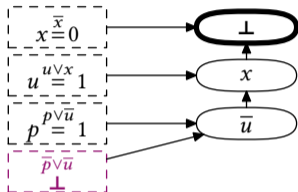
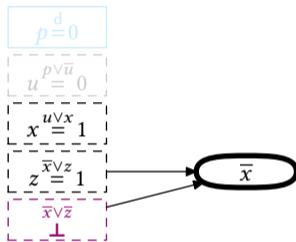
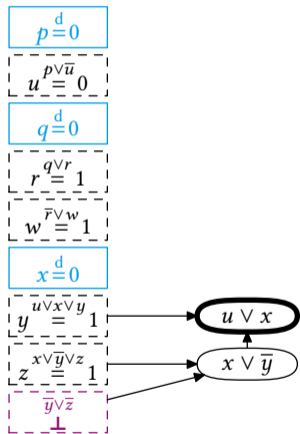
$$(p \vee \bar{u}) \wedge (q \vee r) \wedge (\bar{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee \bar{z}) \wedge (\bar{p} \vee \bar{u})$$



COMPLETE EXAMPLE OF CDCL EXECUTION

Backjump: undo max #decisions while learned clause propagates

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



1. Introduction
 1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
 2. Ensuring Correctness with the Help of Proof Logging
 3. Advantages of Proof Logging
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. Taking a step back
 1. Recent Work
 2. Outlook



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**

$$\frac{C \vee x \quad D \vee \bar{x}}{C \vee D}$$

- ▶ 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**

$$\frac{C \vee x \quad D \vee \bar{x}}{C \vee D}$$

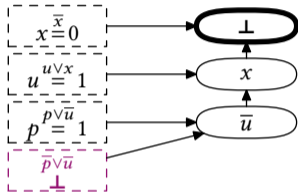
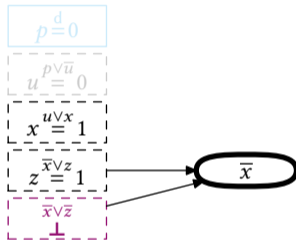
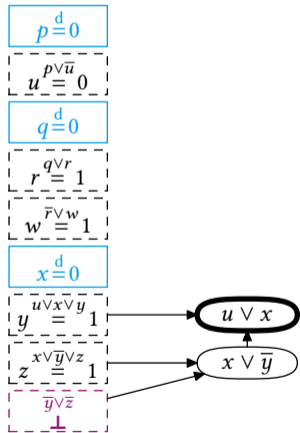
- ▶ Done when contradiction \perp in form of empty clause derived

When run on unsatisfiable formula, **CDCL generates resolution proof***

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

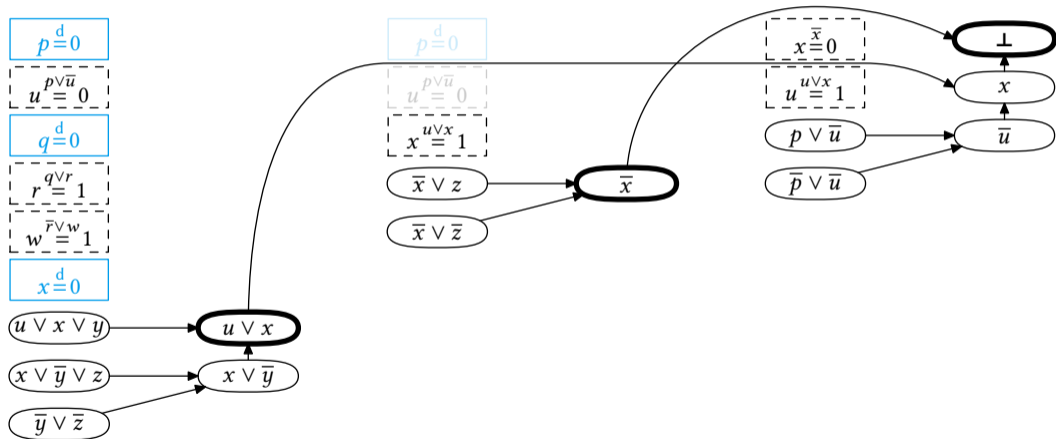
RESOLUTION PROOFS FROM CDCL EXECUTIONS

Obtain resolution proof from our example CDCL execution...



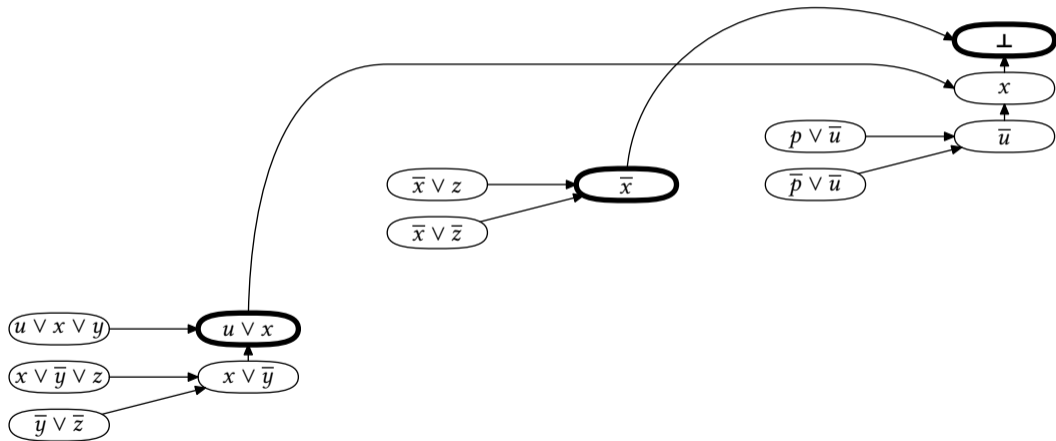
RESOLUTION PROOFS FROM CDCL EXECUTIONS

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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

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

is sequence of reverse unit propagation (RUP) clauses

1. $u \vee x$
2. \bar{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

$$a \vee \bar{x} \vee \bar{y} \quad \bar{a} \vee x \quad \bar{a} \vee y$$

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

EXTENSION VARIABLES, PART 1

Suppose we want a variable a encoding

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

Extended resolution [Tse68]

Resolution rule plus **extension rule** introducing clauses

$$a \vee \bar{x} \vee \bar{y} \quad \bar{a} \vee x \quad \bar{a} \vee y$$

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

EXTENSION VARIABLES, PART 1

Suppose we want a variable a encoding

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

Extended resolution [Tse68]

Resolution rule plus **extension rule** introducing clauses

$$a \vee \bar{x} \vee \bar{y} \quad \bar{a} \vee x \quad \bar{a} \vee y$$

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

1. Introduction
 1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
 2. Ensuring Correctness with the Help of Proof Logging
 3. Advantages of Proof Logging
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. Taking a step back
 1. Recent Work
 2. Outlook



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

Using **0-1 integer linear inequalities** has enabled proof logging for

- ▶ **advanced SAT techniques** so far beyond reach for efficient DRAT proof logging:
 - ▶ Cardinality reasoning
 - ▶ Gaussian elimination
 - ▶ Symmetry breaking
- ▶ **optimisation problems (MaxSAT)**
- ▶ **graph reasoning**
- ▶ **constraint programming** inference

1. Introduction
 1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
 2. Ensuring Correctness with the Help of Proof Logging
 3. Advantages of Proof Logging
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. Taking a step back
 1. Recent Work
 2. Outlook



PSEUDO-BOOLEAN CONSTRAINTS

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

$$\sum_i a_i l_i \geq A$$

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

PSEUDO-BOOLEAN CONSTRAINTS

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

$$\sum_i a_i l_i \geq A$$

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

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

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

2. Cardinality constraints

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

3. General pseudo-Boolean constraints

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

PSEUDO-BOOLEAN REASONING: CUTTING PLANES

Input/model axioms

From the input

PSEUDO-BOOLEAN REASONING: CUTTING PLANES

Input/model axioms

From the input

Literal axioms

$$\overline{l_i \geq 0}$$

PSEUDO-BOOLEAN REASONING: CUTTING PLANES

Input/model axioms

From the input

Literal axioms

$$\overline{l_i \geq 0}$$

Addition

$$\frac{\sum_i a_i l_i \geq A \quad \sum_i b_i l_i \geq B}{\sum_i (a_i + b_i) l_i \geq A + B}$$

PSEUDO-BOOLEAN REASONING: CUTTING PLANES

Input/model axioms

From the input

Literal axioms

$$\overline{l_i \geq 0}$$

Addition

$$\frac{\sum_i a_i l_i \geq A \quad \sum_i b_i l_i \geq B}{\sum_i (a_i + b_i) l_i \geq A + B}$$

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

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

PSEUDO-BOOLEAN REASONING: CUTTING PLANES

Input/model axioms

From the input

Literal axioms

$$\overline{l_i \geq 0}$$

Addition

$$\frac{\sum_i a_i l_i \geq A \quad \sum_i b_i l_i \geq B}{\sum_i (a_i + b_i) l_i \geq A + B}$$

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

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

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

$$\frac{\sum_i a_i l_i \geq A}{\sum_i \lceil \frac{a_i}{c} \rceil l_i \geq \lceil \frac{A}{c} \rceil}$$

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

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

CUTTING PLANES TOY EXAMPLE

$$\begin{array}{r} \text{Multiply by 2} \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad w + 2x + 4y + 2z \geq 5 \\ \text{Add} \quad \hline 3w + 6x + 6y + 2z \geq 9 \end{array}$$

CUTTING PLANES TOY EXAMPLE

$$\begin{array}{r} \text{Multiply by 2} \\ \text{Add} \end{array} \frac{\begin{array}{r} w + 2x + y \geq 2 \\ 2w + 4x + 2y \geq 4 \end{array} \quad w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \quad \bar{z} \geq 0$$

CUTTING PLANES TOY EXAMPLE

$$\begin{array}{r} \text{Multiply by 2} \\ \text{Add} \end{array} \frac{\begin{array}{r} w + 2x + y \geq 2 \\ 2w + 4x + 2y \geq 4 \end{array}}{3w + 6x + 6y + 2z \geq 9} \quad \begin{array}{r} w + 2x + 4y + 2z \geq 5 \\ \bar{z} \geq 0 \end{array} \quad \frac{\bar{z} \geq 0}{2\bar{z} \geq 0} \quad \begin{array}{r} \text{Multiply by 2} \end{array}$$

CUTTING PLANES TOY EXAMPLE

$$\begin{array}{r} \text{Multiply by 2} \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{w + 2x + 4y + 2z \geq 5} \quad \frac{\bar{z} \geq 0}{\bar{z} \geq 0} \\ \text{Add} \quad \frac{2w + 4x + 2y \geq 4}{3w + 6x + 6y + 2z \geq 9} \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z + 2\bar{z} \geq 9} \\ \text{Add} \quad \frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y + 2z + 2\bar{z} \geq 9} \quad \frac{\bar{z} \geq 0}{2\bar{z} \geq 0} \quad \text{Multiply by 2} \end{array}$$

CUTTING PLANES TOY EXAMPLE

$$\begin{array}{r} \text{Multiply by 2} \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{w + 2x + 4y + 2z \geq 5} \quad \frac{\bar{z} \geq 0}{\bar{z} \geq 0} \\ \text{Add} \quad \frac{2w + 4x + 2y \geq 4}{3w + 6x + 6y + 2z \geq 9} \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \quad \frac{\bar{z} \geq 0}{2\bar{z} \geq 0} \\ \text{Add} \quad \frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y + 2z \geq 9} \quad \frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y + 2z \geq 9} \quad \frac{2\bar{z} \geq 0}{2\bar{z} \geq 0} \end{array} \quad \begin{array}{l} \\ \\ \text{Multiply by 2} \end{array}$$

CUTTING PLANES TOY EXAMPLE

$$\begin{array}{r} \text{Multiply by 2} \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{w + 2x + 4y + 2z \geq 5} \quad \frac{\bar{z} \geq 0}{\bar{z} \geq 0} \\ \text{Add} \quad \frac{2w + 4x + 2y \geq 4}{3w + 6x + 6y + 2z \geq 9} \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \quad \frac{\bar{z} \geq 0}{2\bar{z} \geq 0} \\ \text{Add} \quad \frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y} \quad \frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y} \quad \frac{2\bar{z} \geq 0}{\geq 7} \end{array} \quad \begin{array}{l} \\ \\ \text{Multiply by 2} \end{array}$$

CUTTING PLANES TOY EXAMPLE

$$\begin{array}{l} \text{Multiply by 2} \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \quad \frac{\bar{z} \geq 0}{2\bar{z} \geq 0} \\ \text{Add} \quad \frac{\quad}{3w + 6x + 6y} \geq 7 \\ \text{Add} \quad \frac{\quad}{3w + 6x + 6y + 2z} \geq 9 \\ \text{Divide by 3} \quad \frac{w + 2x + 2y}{w + 2x + 2y} \geq 2\frac{1}{3} \end{array} \quad \begin{array}{l} \\ \\ \\ \text{Multiply by 2} \end{array}$$

CUTTING PLANES TOY EXAMPLE

$$\begin{array}{r} \text{Multiply by 2} \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{w + 2x + 4y + 2z \geq 5} \quad \frac{\bar{z} \geq 0}{\bar{z} \geq 0} \\ \text{Add} \quad \frac{2w + 4x + 2y \geq 4}{3w + 6x + 6y + 2z \geq 9} \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \quad \frac{\bar{z} \geq 0}{2\bar{z} \geq 0} \\ \text{Add} \quad \frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y} \quad \frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y} \quad \frac{2\bar{z} \geq 0}{2\bar{z} \geq 0} \\ \text{Divide by 3} \quad \frac{3w + 6x + 6y}{w + 2x + 2y} \quad \frac{3w + 6x + 6y}{w + 2x + 2y} \quad \frac{2\bar{z} \geq 0}{2\bar{z} \geq 0} \end{array} \quad \begin{array}{l} \\ \\ \text{Multiply by 2} \end{array}$$

CUTTING PLANES TOY EXAMPLE

$$\begin{array}{r}
 \text{Multiply by 2} \quad \frac{w + 2x + y \geq 2}{2w + 4x + 2y \geq 4} \quad \frac{w + 2x + 4y + 2z \geq 5}{w + 2x + 4y + 2z \geq 5} \quad \frac{\bar{z} \geq 0}{\bar{z} \geq 0} \\
 \text{Add} \quad \frac{2w + 4x + 2y \geq 4}{3w + 6x + 6y + 2z \geq 9} \quad \frac{w + 2x + 4y + 2z \geq 5}{3w + 6x + 6y + 2z \geq 9} \quad \frac{\bar{z} \geq 0}{2\bar{z} \geq 0} \\
 \text{Add} \quad \frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y} \quad \frac{3w + 6x + 6y + 2z \geq 9}{3w + 6x + 6y} \quad \frac{2\bar{z} \geq 0}{2\bar{z} \geq 0} \\
 \text{Divide by 3} \quad \frac{3w + 6x + 6y}{w + 2x + 2y} \geq 7 \quad \frac{3w + 6x + 6y}{w + 2x + 2y} \geq 7 \quad \frac{2\bar{z} \geq 0}{2\bar{z} \geq 0}
 \end{array}$$

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

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

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

$$\sim z \doteq \bar{z} \geq 0$$

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

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

1. Introduction
 1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
 2. Ensuring Correctness with the Help of Proof Logging
 3. Advantages of Proof Logging
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. Taking a step back
 1. Recent Work
 2. Outlook



RESOLUTION AND CUTTING PLANES

To simulate resolution step such as

$$\frac{\bar{y} \vee \bar{z} \quad x \vee \bar{y} \vee z}{x \vee \bar{y}}$$

we can perform the cutting planes steps

$$\begin{array}{l} \text{Add} \\ \hline \bar{y} + \bar{z} \geq 1 \quad x + \bar{y} + z \geq 1 \\ \hline x + 2\bar{y} \geq 1 \\ \text{Divide by 2} \\ \hline x + \bar{y} \geq 1 \end{array}$$

RESOLUTION AND CUTTING PLANES

To simulate resolution step such as

$$\frac{\bar{y} \vee \bar{z} \quad x \vee \bar{y} \vee z}{x \vee \bar{y}}$$

we can perform the cutting planes steps

$$\begin{array}{l} \text{Add} \\ \hline \bar{y} + \bar{z} \geq 1 \quad x + \bar{y} + z \geq 1 \\ \text{Divide by 2} \\ \hline x + 2\bar{y} \geq 1 \\ \hline x + \bar{y} \geq 1 \end{array}$$

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

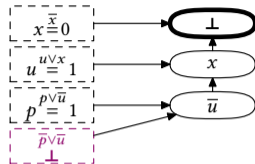
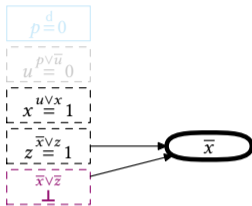
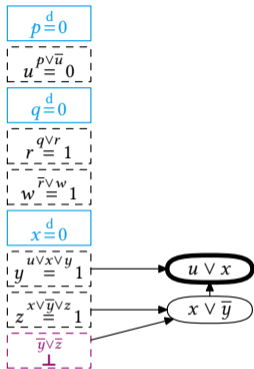
$$\text{Constraint 7} \doteq \bar{y} + \bar{z} \geq 1$$

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

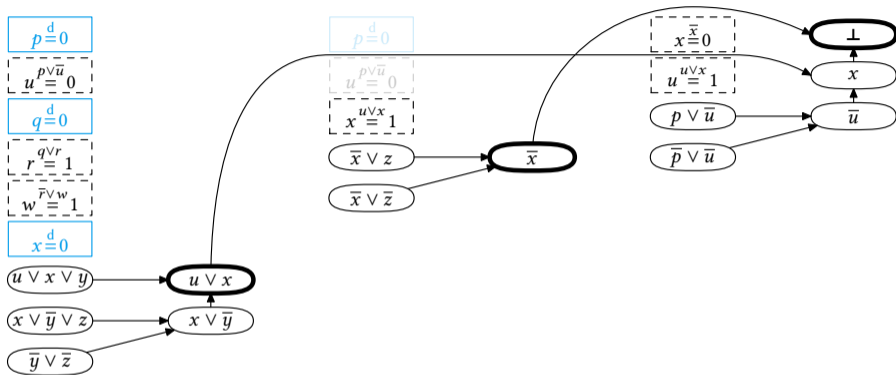
we can write this in the proof log as

$$\text{pol } 7 \ 5 \ + \ 2 \ d$$

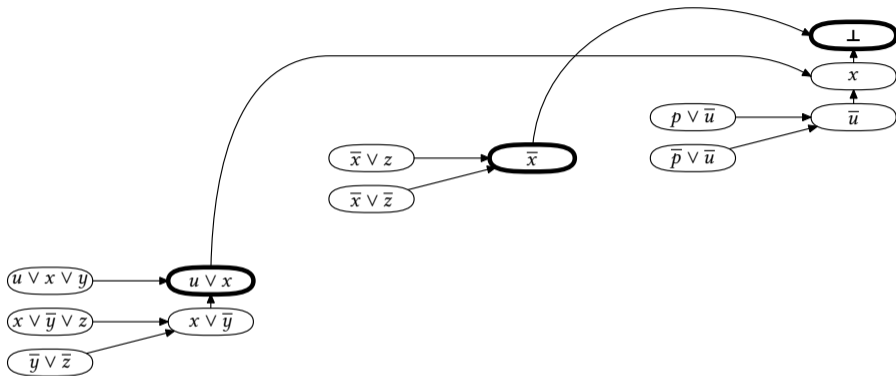
PSEUDO-BOOLEAN PROOF LOGGING FOR CDCL EXAMPLE



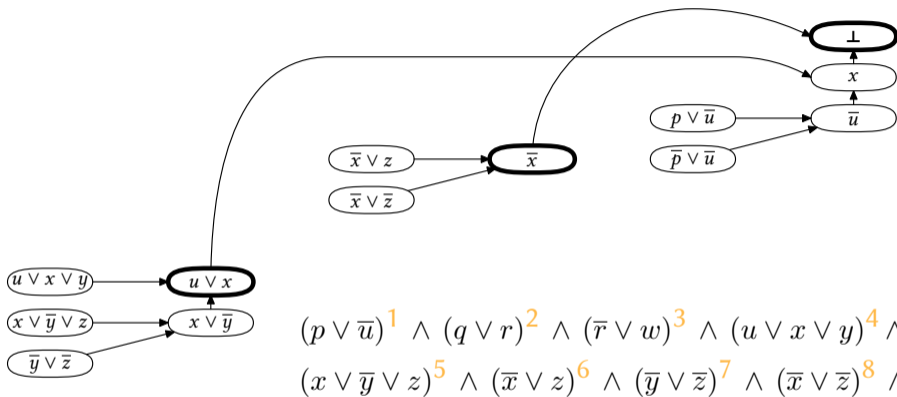
PSEUDO-BOOLEAN PROOF LOGGING FOR CDCL EXAMPLE



PSEUDO-BOOLEAN PROOF LOGGING FOR CDCL EXAMPLE



PSEUDO-BOOLEAN PROOF LOGGING FOR CDCL EXAMPLE



$$\text{pol } 7 \ 5 \ + \ 2 \ d \ 4 \ + \ 2 \ d$$

$$\text{pol } 8 \ 6 \ + \ 2 \ d$$

$$\text{pol } 9 \ 1 \ + \ 2 \ d \ 10 \ + \ 2 \ d \ 11 \ + \ 2 \ d$$

$$\rightsquigarrow \text{Constraint 10} \doteq u + x \geq 1$$

$$\rightsquigarrow \text{Constraint 11} \doteq \bar{x} \geq 1$$

$$\rightsquigarrow \text{Constraint 12} \doteq 0 \geq 1$$



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,

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

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

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,

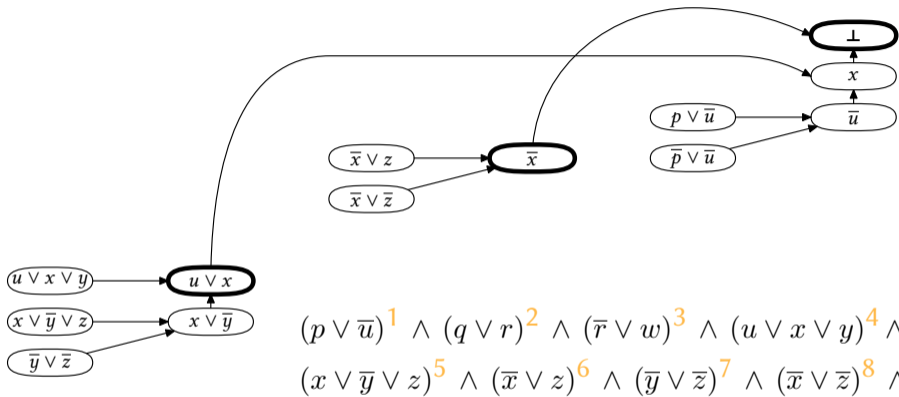
$$x_1 + 2\bar{x}_2 + 3x_3 + 4\bar{x}_4 + 5x_5 \geq 7$$

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

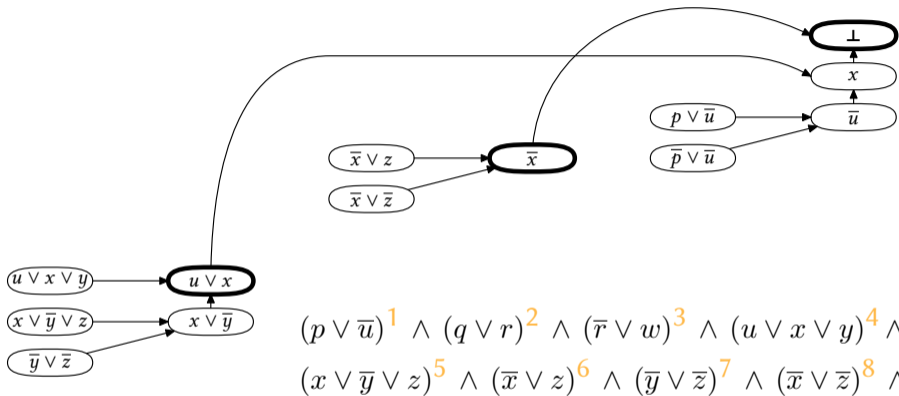
Risk for confusion:

- ▶ 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



PB PROOF LOGGING FOR EXAMPLE CDCL EXECUTION WITH RUP



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

rup 1 u 1 x >= 1 ;

rup 1 ~x >= 1 ;

rup >= 1 ;

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

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

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



1. Introduction

1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
2. Ensuring Correctness with the Help of Proof Logging
3. Advantages of Proof Logging

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. Taking a step back

1. Recent Work
2. Outlook



EXTENSION VARIABLES, PART 2

Suppose we want new, fresh variable a encoding

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

This time, introduce constraints

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

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...

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 talk 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 $f = \sum_i w_i l_i$, $w_i \in \mathbb{Z}$, to minimise subject to the constraints in the formula
- ▶ To maximise, negate objective
- ▶ Log a solution α ; get an objective-improving constraint $\sum_i w_i l_i \leq -1 + \sum_i w_i \alpha(l_i)$
- ▶ 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

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

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

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:

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

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

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

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:

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

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

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

$$9r + \bar{x}_1 + 2x_2 + 3\bar{x}_3 + 4x_4 + 5\bar{x}_5 \geq 9$$

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

1. Introduction
 1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
 2. Ensuring Correctness with the Help of Proof Logging
 3. Advantages of Proof Logging
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. Taking a step back
 1. Recent Work
 2. Outlook



1. Introduction

1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
2. Ensuring Correctness with the Help of Proof Logging
3. Advantages of Proof Logging

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. Taking a step back

1. [Recent Work](#)
2. Outlook



CERTIFICATES FOR MAXSAT

Uniform certificates for MaxSAT solving algorithms (including multi-objective MaxSAT [JBBJ25])

- ▶ Linear SAT-UNSAT search (proof logging [VDB22, Van23, BBN⁺24])
 1. Call SAT solver to find some solution
 2. Add clauses encoding “I want a better solution”
 3. Repeat (last found solution is optimal)

CERTIFICATES FOR MAXSAT

Uniform certificates for MaxSAT solving algorithms (including multi-objective MaxSAT [JBBJ25])

- ▶ Linear SAT-UNSAT search (proof logging [VDB22, Van23, BBN⁺24])
 1. Call SAT solver to find some solution
 2. Add clauses encoding “I want a better solution”
 3. Repeat (last found solution is optimal)
- ▶ Core-guided search (proof logging [BBN⁺23])
 1. Call SAT solver to find solution under most optimistic assumptions
 2. If impossible, rewrite objective given output of SAT solver
 3. Repeat (first solution is optimal)

CERTIFICATES FOR MAXSAT

Uniform certificates for MaxSAT solving algorithms (including multi-objective MaxSAT [JBBJ25])

- ▶ Linear SAT-UNSAT search (proof logging [VDB22, Van23, BBN⁺24])
 1. Call SAT solver to find some solution
 2. Add clauses encoding “I want a better solution”
 3. Repeat (last found solution is optimal)
- ▶ Core-guided search (proof logging [BBN⁺23])
 1. Call SAT solver to find solution under most optimistic assumptions
 2. If impossible, rewrite objective given output of SAT solver
 3. Repeat (first solution is optimal)
- ▶ Branch-and-bound search (proof logging [VCB26])
 1. Run CDCL SAT solver
 2. While running, add bounding constraints

CERTIFICATES FOR MAXSAT

Uniform certificates for MaxSAT solving algorithms (including multi-objective MaxSAT [JBBJ25])

- ▶ Linear SAT-UNSAT search (proof logging [VDB22, Van23, BBN⁺24])
 1. Call SAT solver to find some solution
 2. Add clauses encoding “I want a better solution”
 3. Repeat (last found solution is optimal)
- ▶ Core-guided search (proof logging [BBN⁺23])
 1. Call SAT solver to find solution under most optimistic assumptions
 2. If impossible, rewrite objective given output of SAT solver
 3. Repeat (first solution is optimal)
- ▶ Branch-and-bound search (proof logging [VCB26])
 1. Run CDCL SAT solver
 2. While running, add bounding constraints
- ▶ Implicit Hitting Set (proof logging [IVS⁺26])
 1. Call SAT solver to find solution under most optimistic assumptions
 2. Use hitting set solver (MIP solver) to recompute what most possible optimistic assumptions are
 3. Repeat (first solution is optimal)

CERTIFICATES FOR SYMMETRY BREAKING

- ▶ Important pre-processing technique in many applications
- ▶ How to get without-loss-of-generality arguments inside a formal proof?
- ▶ Solved recently in general [BGMN22, BGMN23, ABB⁺26]

CERTIFICATES FOR HARDWARE MODEL CHECKING

- ▶ Task: check if a **hardware design (circuit)** satisfies desired safety/liveness properties

CERTIFICATES FOR HARDWARE MODEL CHECKING

- ▶ Task: check if a **hardware design (circuit)** satisfies desired safety/liveness properties
- ▶ Certificate: **another circuit** that:
 - ▶ can **easily** be seen to simulate the original one, and
 - ▶ can **easily** be seen to satisfy the desired properties

CERTIFICATES FOR HARDWARE MODEL CHECKING

- ▶ Task: check if a **hardware design (circuit)** satisfies desired safety/liveness properties
- ▶ Certificate: **another circuit** that:
 - ▶ can **easily** be seen to simulate the original one, and
 - ▶ can **easily** be seen to satisfy the desired properties

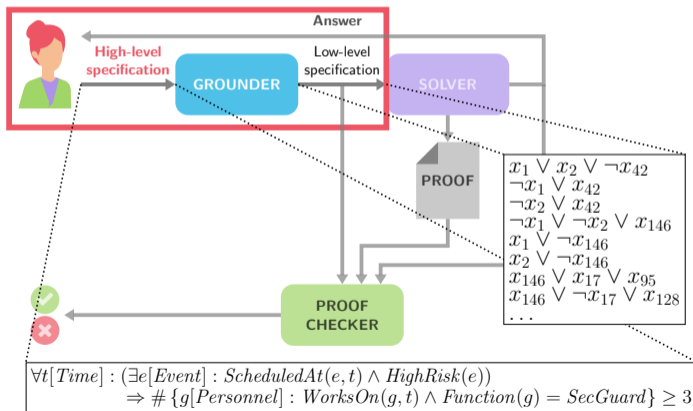
Everything is relative... **easy** checks here still involve a co-NP oracle [FYP⁺25, FYBH26, FY26, FYB⁺26]...

CERTIFICATES FOR MODEL EXPANSION PROBLEMS

- ▶ Often users specify problems in a high-level language
- ▶ Compilation process largely ignored in proof logging work so far, until recently [VECB26]

CERTIFICATES FOR MODEL EXPANSION PROBLEMS

- ▶ Often users specify problems in a high-level language
- ▶ Compilation process largely ignored in proof logging work so far, until recently [VECB26]



1. Introduction
 1. The Success of Combinatorial Solving (and the Dirty Little Secret...)
 2. Ensuring Correctness with the Help of Proof Logging
 3. Advantages of Proof Logging
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. Taking a step back
 1. Recent Work
 2. Outlook



PLANS FOR THE FUTURE

- ▶ Lots of research challenges related to efficient **grounding** algorithms (and certificates)
- ▶ Formally verified proof checkers for grounding
- ▶ Use cases of proofs (for explainability purposes, debugging, auditability, ...)
- ▶ Heuristic algorithms?

REFERENCES I

- [ABB⁺26] Markus Anders, Bart Bogaerts, Benjamin Bogø, Arthur Gontier, Wietze Koops, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Adrian Rebola-Pardo, and Yong Kiam Tan. Faster certified symmetry breaking using orders with auxiliary variables. In *Koenig et al. [KJT26]*, pages 14140–14148.
- [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.

REFERENCES II

- [BBN⁺24] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, Tobias Paxian, and Dieter Vandesande. Certifying without loss of generality reasoning in solution-improving maximum satisfiability. In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2–6, 2024, Girona, Spain*, volume 307 of *LIPICs*, pages 4:1–4:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [BFB⁺26] Ignace Bleukx, Maarten Flippo, Bart Bogaerts, Emir Demirovic, and Tias Guns. Using certifying constraint solvers for generating step-wise explanations. In Koenig et al. [KJT26], pages 14192–14200.
- [BGMN22] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 3698–3707. AAAI Press, 2022.
- [BGMN23] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *J. Artif. Intell. Res.*, 77:1539–1589, 2023.
- [BHvMW21] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, February 2021.
- [Bla37] Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

REFERENCES III

- [BLB10] Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, July 2010.
- [BMN22] Bart Bogaerts, Ciaran McCreesh, and Jakob Nordström. Solving with provably correct results: Beyond satisfiability, and towards constraint programming. Tutorial at the *28th International Conference on Principles and Practice of Constraint Programming*. Slides available at <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.
- [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.
- [DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

REFERENCES IV

- [DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.
- [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.
- [FY26] Nils Froylyks and Emily Yu. Hardware model checking certification with certifaiger and cerbtora. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR) 2026, Part of FLoC 2026, Lisbon, Portugal, July 26–29, 2026*, *Lecture Notes in Computer Science*, Lisbon, Portugal, 2026. Springer.
- [FYB⁺26] Nils Froylyks, Emily Yu, Bart Bogaerts, Armin Biere, and Keijo Heljanko. Liveness proofs for hardware model checking. In *Proceedings of the 38th International Conference on Computer Aided Verification (CAV)*, 2026. accepted.
- [FYBH26] Nils Froylyks, Emily Yu, Armin Biere, and Keijo Heljanko. Certifying constraints in hardware model checking. In *Formal Methods - 27th International Symposium, FM 2026, Tokyo, Japan, May 18–22, 2026, Proceedings*, *Lecture Notes in Computer Science*, Tokyo, Japan, 2026. Springer.
- [FYP⁺25] Nils Froylyks, Emily Yu, Mathias Preiner, Armin Biere, and Keijo Heljanko. Introducing certificates to the hardware model checking competition. In Ruzica Piskac and Zvonimir Rakamaric, editors, *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23–25, 2025, Proceedings, Part I*, volume 15931 of *Lecture Notes in Computer Science*, pages 281–295. Springer, 2025.

REFERENCES V

- [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.
- [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.
- [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.
- [IVS⁺26] Hannes Ihalainen, Dieter Vandesande, André Schidler, Jeremias Berg, Bart Bogaerts, and Matti Järvisalo. Efficient and reliable hitting-set computations for the implicit hitting set approach. In Koenig et al. [KJT26], pages 14251–14260.

REFERENCES VI

- [JBB]25] Christoph Jabs, Jeremias Berg, Bart Bogaerts, and Matti Järvisalo. Certifying pareto optimality in multi-objective maximum satisfiability. In Arie Gurfinkel and Marijn Heule, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part II*, volume 15697 of *Lecture Notes in Computer Science*, pages 108–129. Springer, 2025.
- [KJT26] Sven Koenig, Chad Jenkins, and Matthew E. Taylor, editors. *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*. AAAI Press, 2026.
- [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.
- [MMNS11] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Comput. Sci. Rev.*, 5(2):119–161, 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*.

REFERENCES VII

- [RM16] Olivier Roussel and Vasco M. Manquinho. Input/output format and solver requirements for the competitions of pseudo-Boolean solvers. *Revision 2324*. Available at <http://www.cril.univ-artois.fr/PB16/format.pdf>, January 2016.
- [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>.
- [Van23] Dieter Vandesande. Towards certified MaxSAT solving — certified MaxSAT solving with SAT oracles and encodings of pseudo-Boolean constraints. *Master's thesis, Vrije Universiteit Brussel*, 2023. To appear.
- [VCB26] Dieter Vandesande, Jordi Coll, and Bart Bogaerts. Certified branch-and-bound MaxSAT solving. In Koenig et al. [KJT26], pages 14342–14351.

REFERENCES VIII

- [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.
- [VECB26] Daimy Van Caudenberg, Alexander Ek, Carlos Cantero, and Bart Bogaerts. Towards a certifying grounder. In *Proceedings of 42nd International Conference on Logic Programming (ICLP)*, 2026. accepted.

ACKNOWLEDGEMENTS



This work was partially supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G070521N).



Co-funded 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.