Proof Logging for Combinatorial Optimization Using Pseudo-Boolean Reasoning

Jakob Nordström

University of Copenhagen
and Lund University

Reunion workshop for
“Satisfiability: Theory, Practice, and Beyond”
Simons Institute of the Theory of Computing
June 15, 2022

What Is a Proof?

Claim: N is the product of two primes (think 25957, say)
What is an acceptable proof of such a claim?

What Is a Proof?

Claim: N is the product of two primes (think 25957, say)
What is an acceptable proof of such a claim?

- "Left to the listener. (Just factor and check yourself!)"
  No! Not known how to factor large integers efficiently
  Much of modern crypto rests on assumption this is hard [RSA78]

What Is a Proof?

Claim: N is the product of two primes (think 25957, say)
What is an acceptable proof of such a claim?

- "Left to the listener. (Just factor and check yourself!)"
  No! Not known how to factor large integers efficiently
  Much of modern crypto rests on assumption this is hard [RSA78]

- $$\begin{array}{llll} 25957 \equiv 1 \pmod{2} & & 25957 \equiv 0 \pmod{101} \\ 25957 \equiv 1 \pmod{3} & & 25957 \equiv 1 \pmod{103} \\ 25957 \equiv 2 \pmod{5} & & \vdots \\ \vdots & & 25957 \equiv 0 \pmod{257} \\ 25957 \equiv 19 \pmod{99} & & \vdots \end{array}$$
  OK, but maybe even a bit of overkill

What Is a Proof?

Claim: N is the product of two primes (think 25957, say)
What is an acceptable proof of such a claim?

- "Left to the listener. (Just factor and check yourself!)"
  No! Not known how to factor large integers efficiently
  Much of modern crypto rests on assumption this is hard [RSA78]

- $$\begin{array}{llll} 25957 \equiv 1 \pmod{2} & & 25957 \equiv 0 \pmod{101} \\ 25957 \equiv 1 \pmod{3} & & 25957 \equiv 1 \pmod{103} \\ 25957 \equiv 2 \pmod{5} & & \vdots \\ \vdots & & 25957 \equiv 0 \pmod{257} \\ 25957 \equiv 19 \pmod{99} & & \vdots \end{array}$$
  OK, but maybe even a bit of overkill

- "$25957 = 101 \cdot 257$; check yourself that these are primes."
  Concise! Primality easy to check [Mil76, Rab80, AKS04]

Key demand: Proofs should be short but efficiently verifiable

Proof System

Proof system for formal language L (adapted from [CR79]):

Deterministic algorithm $P(x, \pi)$ that runs in time polynomial in $|x|$ and $|\pi|$ such that

- for all $x \in L$ there is a string π (a proof) such that $P(x, \pi) = 1$,
- for all $x \notin L$ it holds for all strings π that $P(x, \pi) = 0$.

Proof System

Proof system for formal language L (adapted from [CR79]):

Deterministic algorithm $P(x, \pi)$ that runs in time polynomial in $|x|$ and $|\pi|$ such that

- for all $x \in L$ there is a string π (a proof) such that $P(x, \pi) = 1$,
- for all $x \notin L$ it holds for all strings π that $P(x, \pi) = 0$.

Proof π usually sequence of lines, each line following from previous lines
Think of P as "proof checker"

Proof System

Proof system for formal language L (adapted from [CR79]):

Deterministic algorithm $P(x, \pi)$ that runs in time polynomial in $|x|$ and $|\pi|$ such that

- for all $x \in L$ there is a string π (a proof) such that $P(x, \pi) = 1$,
- for all $x \notin L$ it holds for all strings π that $P(x, \pi) = 0$.

Proof π usually sequence of lines, each line following from previous lines
Think of P as "proof checker"

Note that proof π can be very large compared to x
Only have to achieve polynomial time in $|x| + |\pi|$

The Success of Combinatorial Solving and Optimization

- Rich field of math and computer science
- Impact in other areas of science and also industry:
  - airline scheduling
  - logistics
  - hardware verification
  - donor-recipients matching for kidney transplants [MO12, BvdKM$^+$21]
- Typically very challenging problems (NP-complete or worse)
- Lots of effort last decades into developing sophisticated so-called combinatorial solvers that often work surprisingly well in practice
  - Boolean satisfiability (SAT) solving [BHvMW21]
  - Constraint programming [RvBW06]
  - Mixed integer linear programming [AW13, BR07]
  - Satisfiability modulo theories (SMT) solving [BHvMW21]

The Dirty Little Secret...

- Solvers very fast, but sometimes wrong (even best commercial ones) [BLB10, CKSW13, AGJ+18, GSD19, GS19]
- Even worse: No way of knowing for sure when errors happen
- How to check the absence of solutions?
- Or that a solution is optimal?
- And solvers even get feasibility of solutions wrong (though this should be straightforward!)

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
  (Though there is some exciting recent work on SAT solvers [Fle20])

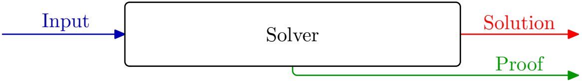
What Can Be Done About Solver Bugs?

- **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
  (Though there is some exciting recent work on SAT solvers [Fle20])
- **Proof logging**
  Make solver certifying [ABM$^+$11, MMNS11] by outputting
  1. not only solution but also
  2. simple, machine-verifiable proof that solution is correct

Proof Logging with Certifying Solvers: Workflow

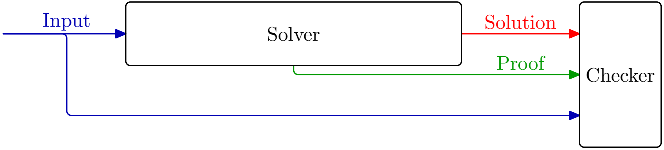
1. Run solver on problem input

Proof Logging with Certifying Solvers: Workflow



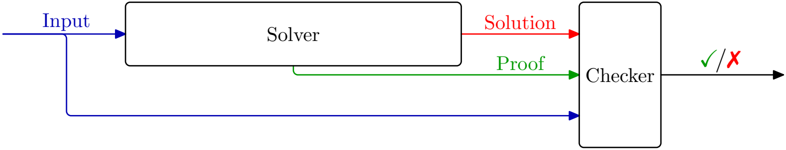
1. Run solver on problem input
2. Get as output not only solution but also proof

Proof Logging with Certifying Solvers: Workflow



1. Run solver on problem input
2. Get as output not only solution but also proof
3. Feed input + solution + proof to proof checker

Proof Logging with Certifying Solvers: Workflow



1. Run solver on problem input
2. Get as output not only solution but also proof
3. Feed input + solution + proof to proof checker
4. Verify that proof checker says solution is correct and/or optimal

Proof Logging with Certifying Solvers: Requirements

Proofs produced by certifying solver [ABM$^+$11, MMNS11] should

- be based on very simple rules
- be powerful enough to allow proof logging with minimal overhead
- not require knowledge of inner workings of solver
- allow verification by stand-alone (formally verified) proof checker

Proof Logging with Certifying Solvers: Requirements

Proofs produced by certifying solver [ABM+11, MMNS11] should

- be based on very simple rules
- be powerful enough to allow proof logging with minimal overhead
- not require knowledge of inner workings of solver
- allow verification by stand-alone (formally verified) proof checker

Does not prove solver correct, but proves solution correct

The Sales Pitch for Proof Logging

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

The Sales Pitch for Proof Logging

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

Success story for SAT solving: DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], ...

The Sales Pitch for Proof Logging

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

Success story for SAT solving: DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], ...

But has remained out of reach for stronger paradigms

The Sales Pitch for Proof Logging

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

Success story for SAT solving: DRAT [HHW13a, HHW13b, WHH14], GRIT [CMS17], LRAT [CHH+17], ...

But has remained out of reach for stronger paradigms
And, in fact, even for some advanced SAT solving techniques

Outline of This Talk

1. Introduction
    - Proofs
    - Combinatorial Solving and Optimization
    - Proof Logging
2. Basic SAT Solving
    - CDCL by Example
    - Resolution
    - Extension Rules
3. Advanced SAT Techniques
    - Cardinality Constraints and Pseudo-Boolean Reasoning
    - Translating Pseudo-Boolean Constraints to CNF
    - Parity Reasoning
4. Beyond SAT
    - Symmetry, Dominance, and Optimization
    - Constraint Programming
    - Further Challenges

A Quick Recap of Modern SAT Solving

DPLL method [DP60, DLL62]

- Assign values to variables (in some smart way)
- Backtrack when conflict with falsified clause

A Quick Recap of Modern SAT Solving

DPLL method [DP60, DLL62]

- Assign values to variables (in some smart way)
- Backtrack when conflict with falsified clause

Conflict-driven clause learning (CDCL) [MS96, BS97, MMZ^{+}01]

- Analyse conflicts in more detail — add new clauses to formula
- More efficient backtracking
- Also let conflicts guide other heuristics

A Quick Recap of Modern SAT Solving

DPLL method [DP60, DLL62]

- Assign values to variables (in some smart way)
- Backtrack when conflict with falsified clause

Conflict-driven clause learning (CDCL) [MS96, BS97, MMZ+01]

- Analyse conflicts in more detail — add new clauses to formula
- More efficient backtracking
- Also let conflicts guide other heuristics

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

Decision

Free choice to assign value to variable

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

Decision

Free choice to assign value to variable

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

decision level 3

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, else decide

Add to assignment trail

Until satisfying assignment or conflict

Conflict Analysis

Time to analyse this conflict and learn from it!

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

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

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

decision level 1

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

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

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

decision level 2

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

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

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

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

decision level 3

Conflict Analysis

Time to analyse this conflict and learn from it!

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

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

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

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

Could backtrack by removing last decision level & flipping last decision

Conflict Analysis

Time to analyse this conflict and learn from it!

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

$\boxed{p \overset{\mathsf{d}}{=} 0}$
$u \overset{p \vee \overline{u}}{=} 0$
decision level 1

$\boxed{q \overset{\mathsf{d}}{=} 0}$
$r \overset{q \vee r}{=} 1$
$w \overset{\overline{r} \vee w}{=} 1$
decision level 2

$\boxed{x \overset{\mathsf{d}}{=} 0}$
$y \overset{u \vee x \vee y}{=} 1$
$z \overset{x \vee \overline{y} \vee z}{=} 1$
$\overset{\overline{y} \vee \overline{z}}{\bot}$
decision level 3

Could backtrack by removing last decision level & flipping last decision

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

Conflict Analysis

Time to analyse this conflict and learn from it!

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

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

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

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

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

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

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

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

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

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

$x \lor \overline{y}$

Could backtrack by removing last decision level & flipping last decision

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

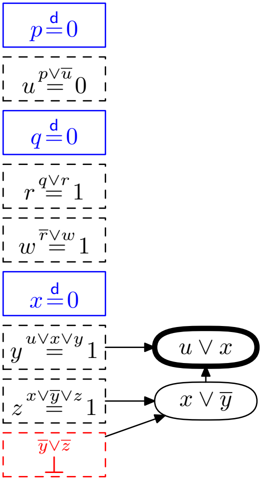
Case analysis over z for last two clauses:

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

Conflict Analysis

Time to analyse this conflict and learn from it!

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



Could backtrack by removing last decision level & flipping last decision

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

Case analysis over z for last two clauses:

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

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

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

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

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

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

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

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

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

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

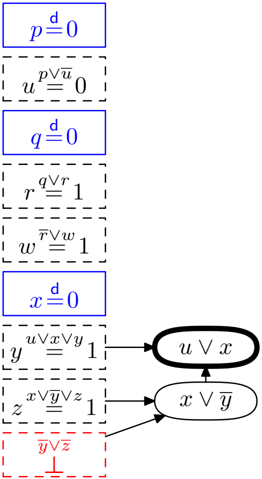
$z \stackrel{x \vee \overline{y} \vee z}{=} 1$ $\qquad x \vee \overline{y}$

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

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

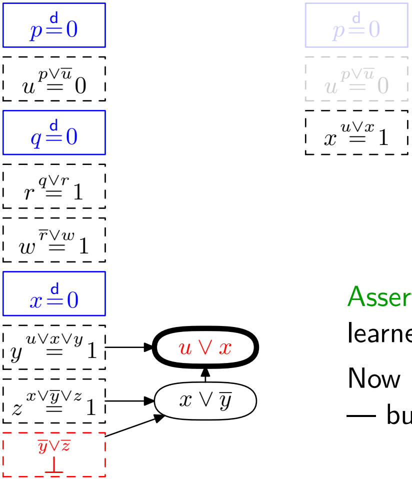


Assertion level 1 (max for non-UIP literal in learned clause) — trim trail to that level

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



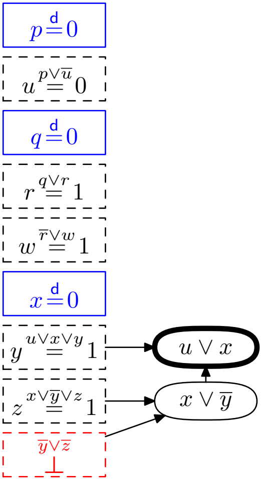
Assertion level 1 (max for non-UIP literal in learned clause) — trim trail to that level

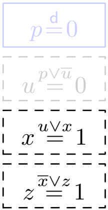
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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





Assertion level 1 (max for non-UIP literal in learned clause) — trim trail to that level

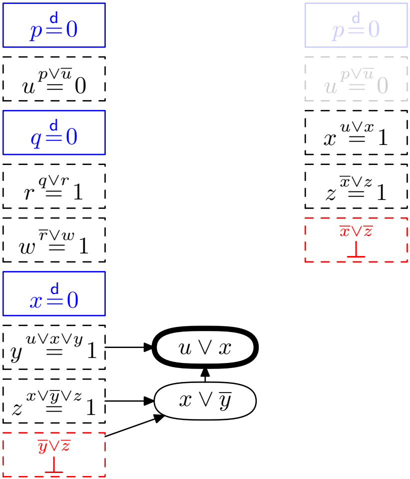
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Then continue as before...

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

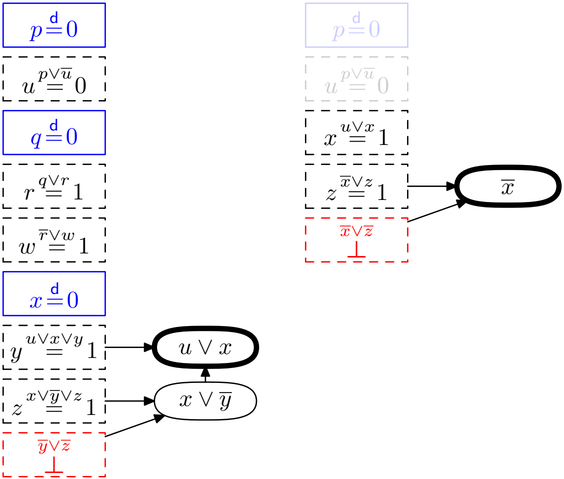
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

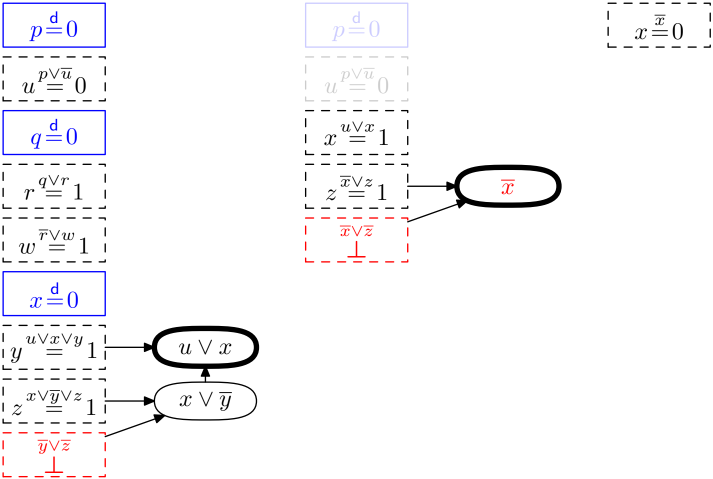
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

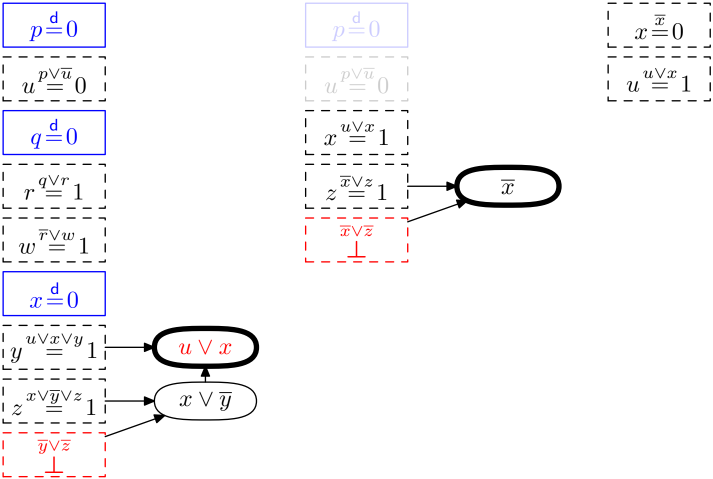
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

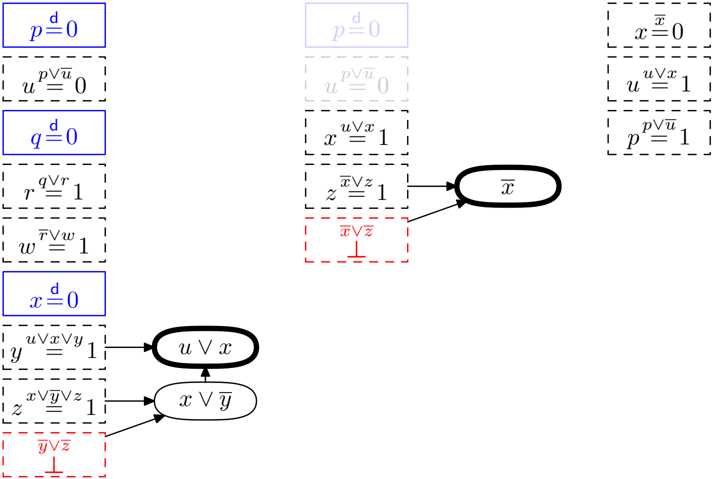
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

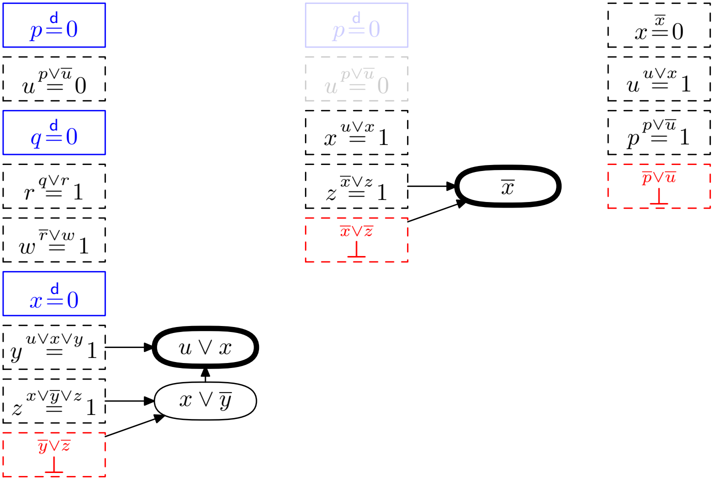
$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

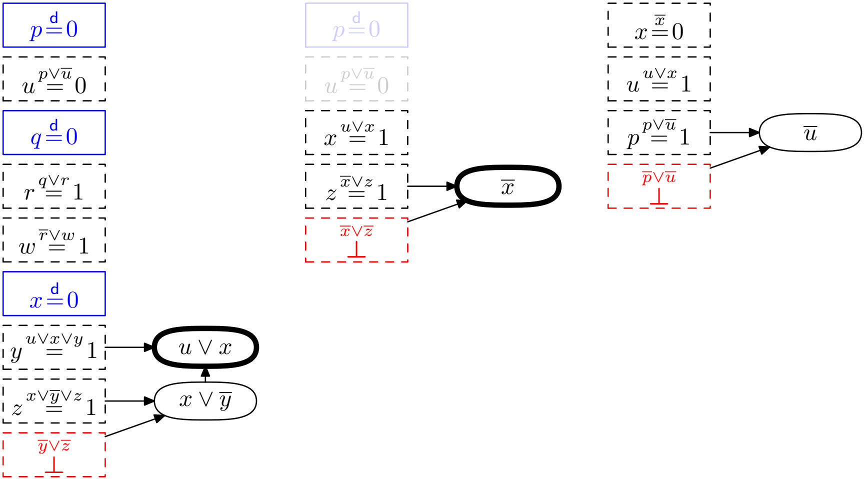
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

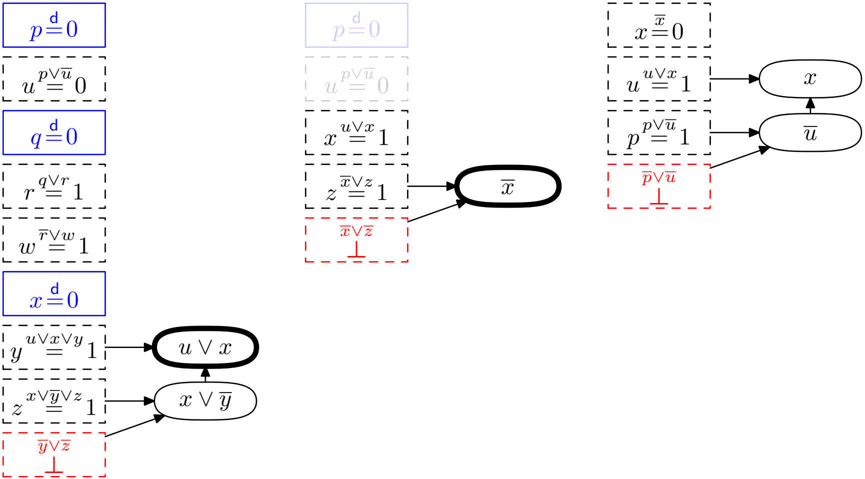
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

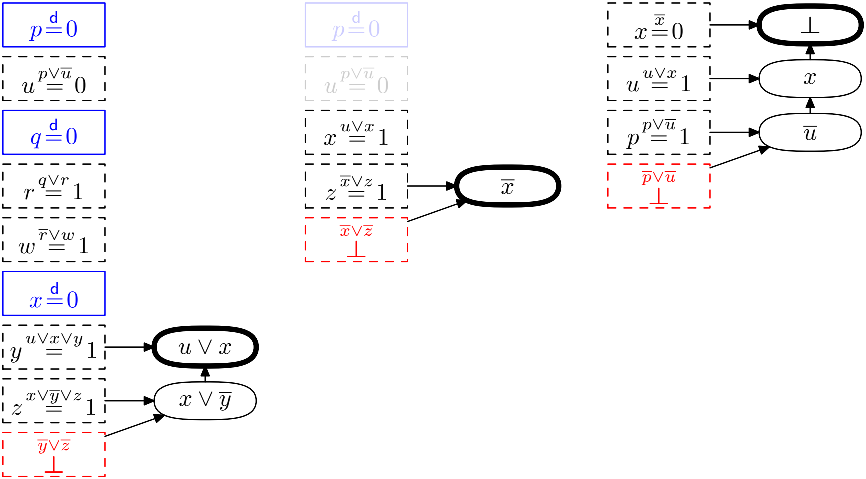
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



CDCL Reasoning and the Resolution Proof System

For CDCL proof logging, need proof system for unsatisfiable formulas

Focus on underlying method of reasoning

CDCL Reasoning and the Resolution Proof System

For CDCL proof logging, need proof system for unsatisfiable formulas
Focus on underlying method of reasoning

Resolution proof system [Bla37, Rob65]

- Start with clauses of formula
- Derive new clauses by **resolution rule**

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

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

CDCL Reasoning and the Resolution Proof System

For CDCL proof logging, need proof system for unsatisfiable formulas
Focus on underlying method of reasoning

Resolution proof system [Bla37, Rob65]

- Start with clauses of formula
- Derive new clauses by **resolution rule**

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

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

When run on unsatisfiable formula, CDCL generates resolution proof*

CDCL Reasoning and the Resolution Proof System

For CDCL proof logging, need proof system for unsatisfiable formulas
Focus on underlying method of reasoning

Resolution proof system [Bla37, Rob65]

- Start with clauses of formula
- Derive new clauses by **resolution rule**

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

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

When run on unsatisfiable formula, CDCL generates resolution proof*

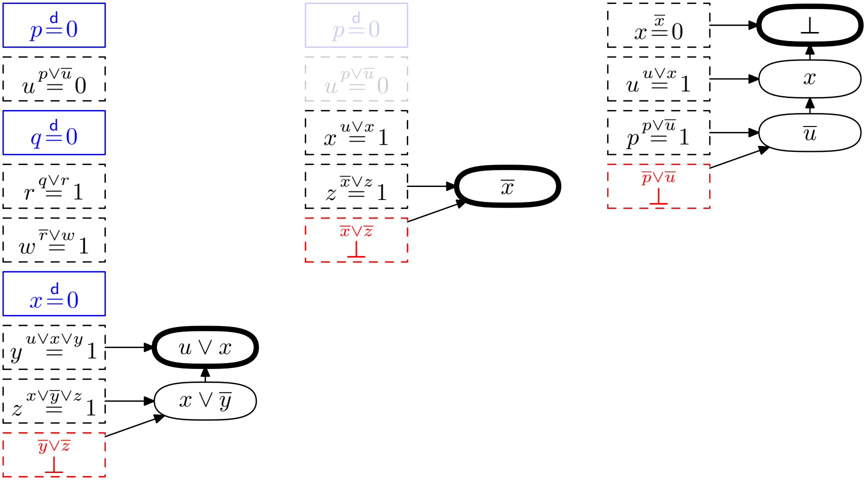
(*) Ignores pre- and inprocessing, but we will get there...

Resolution Proofs from CDCL Executions

Obtain resolution proof...

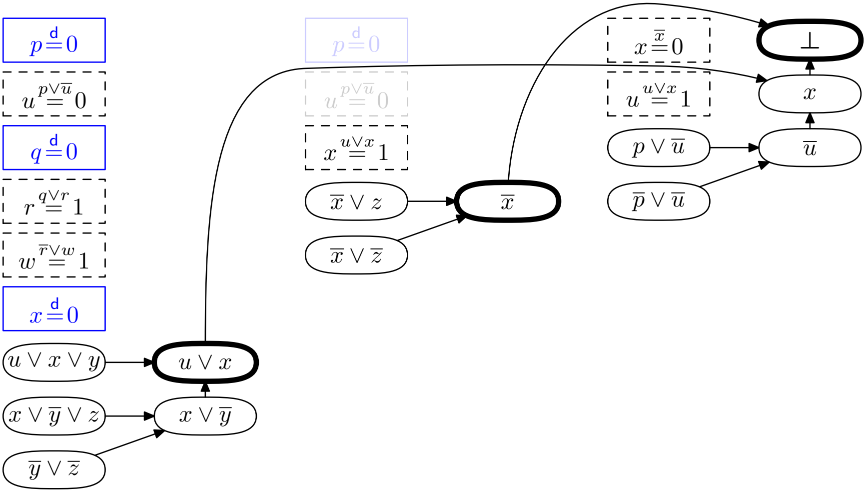
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution...



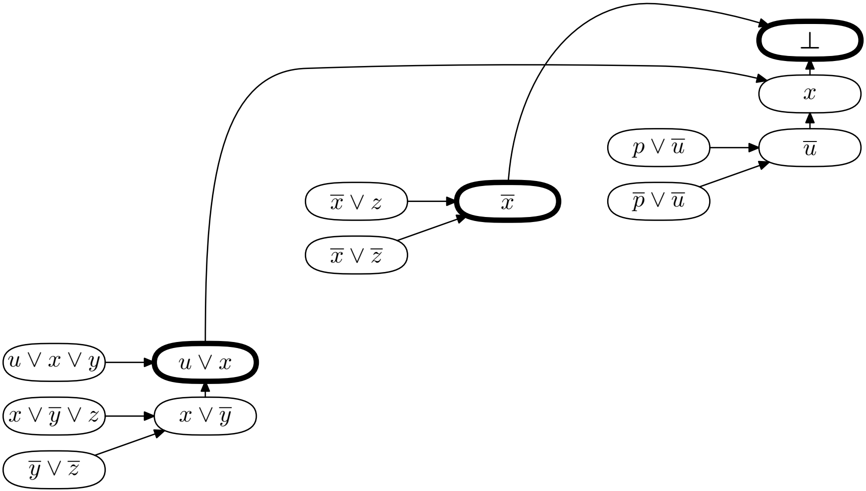
Resolution Proofs from CDCL Executions

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



Resolution Proofs from CDCL Executions

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



Reverse Unit Propagation (RUP)

CDCL SAT solvers could write down resolution proofs, but do something less labour-intensive

Reverse Unit Propagation (RUP)

CDCL SAT solvers could write down resolution proofs, but do something less labour-intensive

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

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

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

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

Reverse Unit Propagation (RUP)

CDCL SAT solvers could write down resolution proofs, but do something less labour-intensive

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

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

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

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

Fact

All clauses learned by CDCL solver are RUP clauses

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

More concise, but requires a little bit more trust
Namely in correct unit propagation

Extension Variables and Redundant Clauses

Say we want new, fresh variable a encoding

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

Extension Variables and Redundant Clauses

Say we want new, fresh variable a encoding

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

Introduce clauses

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

Extension Variables and Redundant Clauses

Say we want new, fresh variable a encoding

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

Introduce clauses

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

Should be in order if variable a doesn't appear anywhere else

Extension Variables and Redundant Clauses

Say we want new, fresh variable a encoding

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

Introduce clauses

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

Should be in order if variable a doesn't appear anywhere else

CDCL pre- and inprocessing techniques can do steps like this

Extension Variables and Redundant Clauses

Say we want new, fresh variable a encoding

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

Introduce clauses

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

Should be in order if variable a doesn't appear anywhere else

CDCL pre- and inprocessing techniques can do steps like this

But resolution proof system cannot certify such derivations (by definition)

Redundance-Based Strengthening

- C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable
- Adding redundant clauses should be OK
- Notions such as RAT [JHB12] and propagation redundancy [HKB17]

Redundance-Based Strengthening

- C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable
- Adding redundant clauses should be OK
- Notions such as RAT [JHB12] and propagation redundancy [HKB17]

Redundance-based strengthening [BT19, GN21]

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

$$F \wedge \neg C \models (F \wedge C)\restriction_\omega$$

Redundance-Based Strengthening

- C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable
- Adding redundant clauses should be OK
- Notions such as RAT [JHB12] and propagation redundancy [HKB17]

Redundance-based strengthening [BT19, GN21]

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

$$F \wedge \neg C \models (F \wedge C)\restriction_\omega$$

- Proof sketch for interesting direction: If α satisfies F but falsifies C, then $\alpha \circ \omega$ satisfies $F \wedge C$
- Implication should be efficiently verifiable (which is the case, e.g., if all clauses in $(F \wedge C)\restriction_\omega$ are RUP)

Deriving $a \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

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

using redundance-based strengthening condition $F \wedge \neg C \models (F \wedge C)\restriction_\omega$

Deriving $a \leftrightarrow (x \land y)$ Using the Redundance Rule

Want to derive

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

using redundance-based strengthening condition $F \land \neg C \models (F \land C)\restriction_\omega$

1. $F \land \neg(a \lor \overline{x} \lor \overline{y}) \models (F \land (a \lor \overline{x} \lor \overline{y}))\restriction_\omega$

Deriving $a \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

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

using redundance-based strengthening condition $F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$

1. $F \wedge \neg(a \vee \overline{x} \vee \overline{y}) \models (F \wedge (a \vee \overline{x} \vee \overline{y}))\restriction_{\omega}$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new clause satisfied

Deriving $a \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

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

using redundance-based strengthening condition $F \wedge \neg C \models (F \wedge C)\restriction_\omega$

1. $F \wedge \neg(a \vee \overline{x} \vee \overline{y}) \models (F \wedge (a \vee \overline{x} \vee \overline{y}))\restriction_\omega$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new clause satisfied

2. $F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge \neg(\overline{a} \vee x) \models (F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x))\restriction_\omega$

Deriving $a \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

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

using redundance-based strengthening condition $F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$

1. $F \wedge \neg(a \vee \overline{x} \vee \overline{y}) \models (F \wedge (a \vee \overline{x} \vee \overline{y}))\restriction_{\omega}$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new clause satisfied
2. $F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge \neg(\overline{a} \vee x) \models (F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x))\restriction_{\omega}$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new clause satisfied
   $\neg(\overline{a} \vee x)$ forces $x \mapsto 0$ which satisfies $a \vee \overline{x} \vee \overline{y}$

Deriving $a \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

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

using redundance-based strengthening condition $F \wedge \neg C \models (F \wedge C)\restriction_\omega$

1. $F \wedge \neg(a \vee \overline{x} \vee \overline{y}) \models (F \wedge (a \vee \overline{x} \vee \overline{y}))\restriction_\omega$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new clause satisfied
2. $F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge \neg(\overline{a} \vee x) \models (F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x))\restriction_\omega$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new clause satisfied
   $\neg(\overline{a} \vee x)$ forces $x \mapsto 0$ which satisfies $a \vee \overline{x} \vee \overline{y}$
3. $F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x) \wedge \neg(\overline{a} \vee y) \models (F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x) \wedge (\overline{a} \vee y))\restriction_\omega$

Deriving $a \leftrightarrow (x \wedge y)$ Using the Redundance Rule

Want to derive

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

using redundance-based strengthening condition $F \wedge \neg C \models (F \wedge C)\restriction_\omega$

1. $F \wedge \neg(a \vee \overline{x} \vee \overline{y}) \models (F \wedge (a \vee \overline{x} \vee \overline{y}))\restriction_\omega$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new clause satisfied

2. $F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge \neg(\overline{a} \vee x) \models (F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x))\restriction_\omega$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new clause satisfied
   $\neg(\overline{a} \vee x)$ forces $x \mapsto 0$ which satisfies $a \vee \overline{x} \vee \overline{y}$

3. $F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x) \wedge \neg(\overline{a} \vee y) \models (F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x) \wedge (\overline{a} \vee y))\restriction_\omega$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new clause satisfied
   $\omega = \{a \mapsto 0\}$ also satisfies $\overline{a} \vee x$
   $\neg(\overline{a} \vee y)$ forces $y \mapsto 0$ which satisfies $a \vee \overline{x} \vee \overline{y}$

Proof Logging for State-of-the-Art SAT Solving

Resolution + redundance rule is as strong as extended Frege proof system

Proof Logging for State-of-the-Art SAT Solving

Resolution + redundance rule is as strong as extended Frege proof system

Should be enough to provide proof logging for state-of-the-art CDCL SAT solvers

Proof Logging for State-of-the-Art SAT Solving

Resolution + redundance rule is as strong as extended Frege proof system

Should be enough to provide proof logging for state-of-the-art CDCL SAT solvers

Except we really care about efficiency, and for some important advanced techniques don't know of efficient enough proof logging methods

Reasoning with Cardinality Constraints

Given clauses

$$x_1 \vee x_2 \vee x_3$$
$$x_1 \vee x_2 \vee x_4$$
$$x_1 \vee x_3 \vee x_4$$
$$x_2 \vee x_3 \vee x_4$$

can deduce that

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

Reasoning with Cardinality Constraints

Given clauses

$$x_1 \lor x_2 \lor x_3$$
$$x_1 \lor x_2 \lor x_4$$
$$x_1 \lor x_3 \lor x_4$$
$$x_2 \lor x_3 \lor x_4$$

can deduce that

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

How provide proof logging for reasoning with such cardinality constraints?

Can solve pigeonhole principle efficiently, which is exponentially hard for basic CDCL [Hak85, BKS04]

Implemented in LINGELING [Lin], but not with DRAT proof logging
Resolution + extension rule can do it in theory, but efficiently in practice?!

Pseudo-Boolean Constraints

Pseudo-Boolean constraints are 0-1 integer linear constraints

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

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

Some Types of Pseudo-Boolean Constraints

1. Clauses

$$x \lor \overline{y} \lor z \quad \Leftrightarrow \quad x + \overline{y} + z \geq 1$$

Some Types of Pseudo-Boolean Constraints

1. Clauses

$$x \vee \overline{y} \vee z \quad \Leftrightarrow \quad x + \overline{y} + z \geq 1$$

2. Cardinality constraints

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

Some Types of Pseudo-Boolean Constraints

1. Clauses

$$x \vee \overline{y} \vee z \quad \Leftrightarrow \quad x + \overline{y} + z \geq 1$$

2. Cardinality constraints

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

3. General pseudo-Boolean constraints

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$$

Division $$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $\dfrac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$

Division $\dfrac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$

Toy example:

Lin comb $\dfrac{2x + 4y + 2z + w \geq 5 \qquad 2x + y + w \geq 2}{}$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$$

Division $$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$$

Toy example:

Lin comb $$\frac{2x + 4y + 2z + w \geq 5 \qquad 2x + y + w \geq 2}{(2x + 4y + 2z + w) + 2 \cdot (2x + y + w) \geq 5 + 2 \cdot 2}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$$

Division $$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$$

Toy example:

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $\dfrac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$

Division $\dfrac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$

Toy example:

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $$\frac{\sum_i a_i\ell_i \geq A \qquad \sum_i b_i\ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$$

Division $$\frac{\sum_i c a_i\ell_i \geq A}{\sum_i a_i\ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$$

Toy example:

$$\text{Lin comb}\ \frac{\text{Lin comb}\ \dfrac{2x + 4y + 2z + w \geq 5 \qquad 2x + y + w \geq 2}{6x + 6y + 2z + 3w \geq 9} \qquad \overline{z} \geq 0}{6x + 6y + 2z + 3w + 2 \cdot \overline{z} \geq 9 + 2 \cdot 0}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $$\frac{\sum_i a_i\ell_i \geq A \qquad \sum_i b_i\ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$$

Division $$\frac{\sum_i c a_i\ell_i \geq A}{\sum_i a_i\ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$$

Toy example:

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$$

Division $$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$$

Toy example:

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

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$$

Division $$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$$

Toy example:

$$\dfrac{\text{Lin comb}\ \dfrac{2x + 4y + 2z + w \geq 5 \qquad 2x + y + w \geq 2}{6x + 6y + 2z + 3w \geq 9} \qquad \overline{z} \geq 0}{\text{Div}\ \dfrac{6x + 6y + 3w \geq 7}{2x + 2y + w \geq 2\frac{1}{3}}}\ \text{Lin comb}$$

Pseudo-Boolean Reasoning: Cutting Planes [CCT87]

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

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B \in \mathbb{N}]$$

Division $$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c \in \mathbb{N}^+]$$

Toy example:

$$\dfrac{\text{Lin comb}\;\dfrac{2x+4y+2z+w \geq 5 \qquad 2x+y+w \geq 2}{6x+6y+2z+3w \geq 9} \qquad \overline{z} \geq 0}{\text{Div}\;\dfrac{6x+6y+3w \geq 7}{2x+2y+w \geq 3}}\;\text{Lin comb}$$

Recovering cardinality constraints from CNF

Clauses

$$x_1 \vee x_2 \vee x_3$$
$$x_1 \vee x_2 \vee x_4$$
$$x_1 \vee x_3 \vee x_4$$
$$x_2 \vee x_3 \vee x_4$$

Recovering cardinality constraints from CNF

Clauses

$$x_1 \lor x_2 \lor x_3$$
$$x_1 \lor x_2 \lor x_4$$
$$x_1 \lor x_3 \lor x_4$$
$$x_2 \lor x_3 \lor x_4$$

Pseudo-Boolean constraints

$$x_1 + x_2 + x_3 \geq 1$$
$$x_1 + x_2 + x_4 \geq 1$$
$$x_1 + x_3 + x_4 \geq 1$$
$$x_2 + x_3 + x_4 \geq 1$$

Add all up

$$3x_1 + 3x_2 + 3x_3 + 3x_4 \geq 4$$

and divide by 3 to get

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

Extended Cutting Planes

Combine cutting planes method with **redundance** rule

Extended Cutting Planes

Combine cutting planes method with **redundance** rule

Redundance-based strengthening [BT19, GN21]

Add constraint C to formula F if exists witness substitution ω such that

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

Extended Cutting Planes

Combine cutting planes method with **redundance** rule

Redundance-based strengthening [BT19, GN21]

Add constraint C to formula F if exists witness substitution ω such that

$$F \land \neg C \models (F \land C)\restriction_\omega$$

- Cutting planes can do efficiently anything that resolution can do
- Reverse unit propagation works also for 0-1 linear inequalities
- RAT $=$ redundance rule with witness flipping RAT literal

$\Rightarrow$ Strict extension of DRAT

Extended Cutting Planes

Combine cutting planes method with **redundance** rule

Redundance-based strengthening [BT19, GN21]

Add constraint C to formula F if exists witness substitution ω such that

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

- Cutting planes can do efficiently anything that resolution can do
- Reverse unit propagation works also for 0-1 linear inequalities
- RAT $=$ redundance rule with witness flipping RAT literal

$\Rightarrow$ Strict extension of DRAT

- Lifts reasoning from clauses to 0-1 inequalities (still simple objects)
- Implemented in proof checker VeriPB [Ver]
- Yields surprisingly expressive proof logging system

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

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

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

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

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

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

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

$\overline{s}_{1,1} \vee x_1$

$\overline{s}_{2,1} \vee s_{1,1} \vee x_2$

$\overline{s}_{2,2} \vee s_{1,1}$

$\overline{s}_{2,2} \vee x_2$

$\overline{s}_{3,1} \vee s_{2,1} \vee x_3$

$\overline{s}_{3,2} \vee s_{2,1}$

$\overline{s}_{3,2} \vee s_{2,2} \vee x_3$

$\overline{s}_{4,1} \vee s_{3,1} \vee x_4$

$\overline{s}_{4,2} \vee s_{3,1}$

$\overline{s}_{4,2} \vee s_{3,2} \vee x_4$

$s_{4,2}$

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

- MINISAT+ [ES06]
- OPEN-WBO [MML14]
- NAPS [SN15]

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

$\overline{s}_{1,1} \lor x_1$

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

$\overline{s}_{2,2} \lor s_{1,1}$

$\overline{s}_{2,2} \lor x_2$

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

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

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

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

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

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

$s_{4,2}$

How know translation correct?

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

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

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

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

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

$\overline{s}_{1,1} \lor x_1$

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

$\overline{s}_{2,2} \lor s_{1,1}$

$\overline{s}_{2,2} \lor x_2$

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

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

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

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

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

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

$s_{4,2}$

How know translation correct?

CDCL Solvers on Pseudo-Boolean Inputs

Can re-encode to CNF and run CDCL:

- MINISAT+ [ES06]
- OPEN-WBO [MML14]
- NAPS [SN15]

E.g., encode pseudo-Boolean constraint

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

to clauses with extension variables

$$s_{i,k} \Leftrightarrow \sum_{j=1}^{i} x_j \geq k$$

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

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

$\overline{s}_{1,1} \lor x_1$

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

$\overline{s}_{2,2} \lor s_{1,1}$

$\overline{s}_{2,2} \lor x_2$

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

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

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

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

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

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

$s_{4,2}$

How know translation correct?

VERIPB can certify pseudo-Boolean-to-CNF rewriting [GMNO22]

Parity (XOR) Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Parity (XOR) Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

Parity (XOR) Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Parity (XOR) Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Exponentially hard for CDCL [Urq87]
But used in CryptoMiniSat [Cry]

Parity (XOR) Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Exponentially hard for CDCL [Urq87]
But used in CryptoMiniSat [Cry]

DRAT proof logging like [PR16] too inefficient in practice!

Parity (XOR) Reasoning

Given clauses

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

and

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

want to derive

$$x \vee \overline{w}$$
$$\overline{x} \vee w$$

This is just parity reasoning:

$$x + y + z = 1 \pmod 2$$
$$y + z + w = 1 \pmod 2$$

imply

$$x + w = 0 \pmod 2$$

Exponentially hard for CDCL [Urq87]
But used in CryptoMiniSat [Cry]

DRAT proof logging like [PR16] too inefficient in practice!

Could add XORs to language, but prefer to keep things super-simple and verifiable...

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Use redundance rule with fresh variables a,b to derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("$=$" syntactic sugar for "$\geq$" plus "$\leq$")

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Use redundance rule with fresh variables a,b to derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("$=$" syntactic sugar for "$\geq$" plus "$\leq$")
Add to get

$$x + w + 2y + 2z + 2a + 2b = 6$$

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Use redundance rule with fresh variables a,b to derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("$=$" syntactic sugar for "$\geq$" plus "$\leq$")
Add to get

$$x + w + 2y + 2z + 2a + 2b = 6$$

From this can extract

$$x + \overline{w} \geq 1$$
$$\overline{x} + w \geq 1$$

Pseudo-Boolean Proof Logging for XOR Reasoning

Given clauses

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

and

$$y \lor z \lor w$$
$$y \lor \overline{z} \lor \overline{w}$$
$$\overline{y} \lor z \lor \overline{w}$$
$$\overline{y} \lor \overline{z} \lor w$$

want to derive

$$x \lor \overline{w}$$
$$\overline{x} \lor w$$

Use redundance rule with fresh variables a,b to derive

$$x + y + z + 2a = 3$$
$$y + z + w + 2b = 3$$

("$=$" syntactic sugar for "$\geq$" plus "$\leq$")
Add to get

$$x + w + 2y + 2z + 2a + 2b = 6$$

From this can extract

$$x + \overline{w} \geq 1$$
$$\overline{x} + w \geq 1$$

VeriPB can certify XOR reasoning [GN21]

The Challenge of Symmetries

Symmetries

- crucial for some optimization problems [AW13, GSVW14]
- show up also in hard SAT benchmarks

The Challenge of Symmetries

Symmetries

- crucial for some optimization problems [AW13, GSVW14]
- show up also in hard SAT benchmarks

Symmetry breaking

- Add clauses filtering out symmetric solutions [DBBD16]
- DRAT proof logging for limited cases only [HHW15]

The Challenge of Symmetries

Symmetries

- crucial for some optimization problems [AW13, GSVW14]
- show up also in hard SAT benchmarks

Symmetry breaking

- Add clauses filtering out symmetric solutions [DBBD16]
- DRAT proof logging for limited cases only [HHW15]

Symmetric learning

- Allow to add all symmetric versions of learned clause [DBB17]
- Recently proposed proof logging in [TD20]
  1. Special-purpose, specific approach
  2. Requires adding explicit concept of symmetries
  3. Not compatible with preprocessing techniques

  Better to keep proof system super-simple and verifiable...

Optimization Problems

Deal with symmetries by switching focus to optimization
(which the title of the talk kind of promised anyway)

Optimization Problems

Deal with symmetries by switching focus to optimization
(which the title of the talk kind of promised anyway)

Pseudo-Boolean optimization

Minimize $f = \sum_i w_i \ell_i$ (for $w_i \in \mathbb{N}$) subject to constraints in F

Optimization Problems

Deal with symmetries by switching focus to optimization
(which the title of the talk kind of promised anyway)

Pseudo-Boolean optimization

Minimize $f = \sum_i w_i \ell_i$ (for $w_i \in \mathbb{N}$) subject to constraints in F

Proof of optimality:

- F satisfied by α
- $F \wedge \left(\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)\right)$ is infeasible

Optimization Problems

Deal with symmetries by switching focus to optimization
(which the title of the talk kind of promised anyway)

Pseudo-Boolean optimization

Minimize $f = \sum_i w_i \ell_i$ (for $w_i \in \mathbb{N}$) subject to constraints in F

Proof of optimality:

- F satisfied by α
- $F \wedge \left(\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)\right)$ is infeasible

Note that $\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)$ means $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \cdot \alpha(\ell_i)$

Optimization Problems

Deal with symmetries by switching focus to optimization
(which the title of the talk kind of promised anyway)

Pseudo-Boolean optimization

Minimize $f = \sum_i w_i \ell_i$ (for $w_i \in \mathbb{N}$) subject to constraints in F

Proof of optimality:

- F satisfied by α
- $F \wedge \left(\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)\right)$ is infeasible

Note that $\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)$ means $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \cdot \alpha(\ell_i)$

Spoiler alert:
For decision problem, nothing stops us from inventing objective function
(like lexicographic order $\sum_{i=1}^{n} 2^i \cdot x_i$)

Proof Logging for Optimization Problems

How does proof system change?

Proof Logging for Optimization Problems

How does proof system change?

Rules must preserve (at least one) optimal solution

Proof Logging for Optimization Problems

How does proof system change?
Rules must preserve (at least one) optimal solution

1. Standard cutting planes rules OK — derive constraints that must hold for any satisfying assignment

Proof Logging for Optimization Problems

How does proof system change?
Rules must preserve (at least one) optimal solution

1. Standard cutting planes rules OK — derive constraints that must hold for any satisfying assignment
2. Once solution α has been found, allow constraint $\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)$ to force search for better solutions

Proof Logging for Optimization Problems

How does proof system change?
Rules must preserve (at least one) optimal solution

1. Standard cutting planes rules OK — derive constraints that must hold for any satisfying assignment

2. Once solution α has been found, allow constraint $\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)$ to force search for better solutions

3. Redundance rule must not destroy good solutions

Proof Logging for Optimization Problems

How does proof system change?
Rules must preserve (at least one) optimal solution

1. Standard cutting planes rules OK — derive constraints that must hold for any satisfying assignment
2. Once solution α has been found, allow constraint $\sum_i w_i \ell_i < \sum_i w_i \cdot \alpha(\ell_i)$ to force search for better solutions
3. Redundance rule must not destroy good solutions

Redundance-based strengthening, optimization version [BGMN22]

Add constraint C to formula F if exists witness substitution ω such that

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

Redundance and Dominance Rules

Redundance-based strengthening, optimization version [BGMN22]

Add constraint C to formula F if exists witness substitution ω such that

$$F \wedge \neg C \models (F \wedge C)\restriction_\omega \wedge f\restriction_\omega \leq f$$

Redundance and Dominance Rules

Redundance-based strengthening, optimization version [BGMN22]

Add constraint C to formula F if exists witness substitution ω such that

$$F \land \neg C \models (F \land C)\restriction_\omega \land f\restriction_\omega \leq f$$

Can be more aggressive if witness ω strictly improves solution

Redundance and Dominance Rules

Redundance-based strengthening, optimization version [BGMN22]

Add constraint C to formula F if exists witness substitution ω such that

$$F \land \neg C \models (F \land C)\restriction_\omega \ \land \ f\restriction_\omega \leq f$$

Can be more aggressive if witness ω strictly improves solution

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \land \neg D \models F\restriction_\omega \ \land \ f\restriction_\omega < f$$

Soundness of Dominance Rule

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \wedge \neg D \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Why is this sound?

Soundness of Dominance Rule

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \land \neg D \models F\restriction_\omega \land f\restriction_\omega < f$$

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \wedge \neg D \models F\restriction_\omega \ \wedge \ f\restriction_\omega < f$$

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \land \neg D \models F\restriction_\omega \ \land\ f\restriction_\omega < f$$

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \wedge \neg D \models F\restriction_\omega \;\wedge\; f\restriction_\omega < f$$

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \land \neg D \models F\restriction_\omega \;\land\; f\restriction_\omega < f$$

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \wedge \neg D \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \land \neg D \models F\restriction_\omega \land f\restriction_\omega < f$$

Why is this sound?

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

Soundness of Dominance Rule

Dominance-based strengthening (simplified) [BGMN22]

Add constraint D to formula F if exists witness substitution ω such that

$$F \land \neg D \models F\restriction_\omega \;\land\; f\restriction_\omega < f$$

Why is this sound?

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

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified) [BGMN22]

If $D_1, D_2, \ldots, D_{m-1}$ have been derived from F (maybe using dominance), then can derive also D_m if exists witness substitution ω such that

$$F \wedge \bigwedge_{i=1}^{m-1} D_i \wedge \neg D_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified) [BGMN22]

If $D_1, D_2, \ldots, D_{m-1}$ have been derived from F (maybe using dominance), then can derive also D_m if exists witness substitution ω such that

$$F \wedge \bigwedge_{i=1}^{m-1} D_i \wedge \neg D_m \models F{\upharpoonright_\omega} \wedge f{\upharpoonright_\omega} < f$$

Why is this sound?

- Same inductive proof as before, but nested

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified) [BGMN22]

If $D_1, D_2, \ldots, D_{m-1}$ have been derived from F (maybe using dominance), then can derive also D_m if exists witness substitution ω such that

$$F \wedge \bigwedge_{i=1}^{m-1} D_i \wedge \neg D_m \models F{\restriction_\omega} \wedge f{\restriction_\omega} < f$$

Why is this sound?

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

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified) [BGMN22]

If $D_1, D_2, \ldots, D_{m-1}$ have been derived from F (maybe using dominance), then can derive also D_m if exists witness substitution ω such that

$$F \wedge \bigwedge_{i=1}^{m-1} D_i \wedge \neg D_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Why is this sound?

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

Further extensions:

- Define dominance rule w.r.t. order independent of objective function

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified) [BGMN22]

If $D_1, D_2, \ldots, D_{m-1}$ have been derived from F (maybe using dominance), then can derive also D_m if exists witness substitution ω such that

$$F \wedge \bigwedge_{i=1}^{m-1} D_i \wedge \neg D_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Why is this sound?

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

Further extensions:

- Define dominance rule w.r.t. order independent of objective function
- Switch between different orders in same proof

Strength of Dominance Rule

Dominance-based strengthening (stronger, still simplified) [BGMN22]

If $D_1, D_2, \ldots, D_{m-1}$ have been derived from F (maybe using dominance), then can derive also D_m if exists witness substitution ω such that

$$F \wedge \bigwedge_{i=1}^{m-1} D_i \wedge \neg D_m \models F\restriction_\omega \wedge f\restriction_\omega < f$$

Why is this sound?

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

Further extensions:

- Define dominance rule w.r.t. order independent of objective function
- Switch between different orders in same proof
- More in Stephan Gocht's talk (after lunch)

Proof Logging for Constraint Programming

Pseudo-Boolean proof logging can also certify reasoning in

- subgraph solvers [GMN20, GMM$^+$20]
- general constraint programming solvers [EGMN20, GMN22]

Proof Logging for Constraint Programming

Pseudo-Boolean proof logging can also certify reasoning in

- subgraph solvers [GMN20, GMM$^+$20]
- general constraint programming solvers [EGMN20, GMN22]

More in Ciaran McCreesh's talk (after lunch)

Further Challenges Beyond SAT

Proof logging for combinatorial optimization

- Symmetric learning and recycling (substitution) of subproofs
- Maximum satisfiability (MaxSAT) and pseudo-Boolean optimization
- Mixed integer linear programming (some work in [CGS17, EG21])
- Satisfiability modulo theories (SMT) solving
- Analysis of power of proof logging systems

Further Challenges Beyond SAT

Proof logging for combinatorial optimization

- Symmetric learning and recycling (substitution) of subproofs
- Maximum satisfiability (MaxSAT) and pseudo-Boolean optimization
- Mixed integer linear programming (some work in [CGS17, EG21])
- Satisfiability modulo theories (SMT) solving
- Analysis of power of proof logging systems

Explainability, robustness, fairness

- Certified explanations for decisions or classifications?
- Proofs of neural network robustness to limited input perturbations?
- Rigorous symbolic reasoning about fairness?

Further Challenges Beyond SAT

Proof logging for combinatorial optimization

- Symmetric learning and recycling (substitution) of subproofs
- Maximum satisfiability (MaxSAT) and pseudo-Boolean optimization
- Mixed integer linear programming (some work in [CGS17, EG21])
- Satisfiability modulo theories (SMT) solving
- Analysis of power of proof logging systems

Explainability, robustness, fairness

- Certified explanations for decisions or classifications?
- Proofs of neural network robustness to limited input perturbations?
- Rigorous symbolic reasoning about fairness?

And more. . .

- Lots of challenging problems and interesting ideas

Further Challenges Beyond SAT

Proof logging for combinatorial optimization

- Symmetric learning and recycling (substitution) of subproofs
- Maximum satisfiability (MaxSAT) and pseudo-Boolean optimization
- Mixed integer linear programming (some work in [CGS17, EG21])
- Satisfiability modulo theories (SMT) solving
- Analysis of power of proof logging systems

Explainability, robustness, fairness

- Certified explanations for decisions or classifications?
- Proofs of neural network robustness to limited input perturbations?
- Rigorous symbolic reasoning about fairness?

And more...

- Lots of challenging problems and interesting ideas
- We're hiring! Talk to me to join the proof logging revolution!

Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like a promising approach
- Well established for Boolean satisfiability (SAT) solving, but even there advanced techniques have remained out of reach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity

Summing up

- Combinatorial solving and optimization is a true success story
- But ensuring correctness is a crucial, and not yet satisfactorily addressed, concern
- Certifying solvers producing machine-verifiable proofs of correctness seems like a promising approach
- Well established for Boolean satisfiability (SAT) solving, but even there advanced techniques have remained out of reach
- Cutting planes reasoning with pseudo-Boolean constraints seems to hit a sweet spot between simplicity and expressivity

Thank you for your attention!

References I

[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.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, September 2004.

[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.

[BGMN22] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI '22)*, February 2022. To appear.

References II

[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.

[BKS04] Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, December 2004. Preliminary version in *IJCAI '03*.

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

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

[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.

References III

[BS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

[BT19] Samuel R. Buss and Neil Thapen. DRAT proofs, propagation redundancy, and extended resolution. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT '19)*, volume 11628 of *Lecture Notes in Computer Science*, pages 71–89. Springer, July 2019.

[BvdKM+21] Péter Biró, Joris van de Klundert, David F. Manlove, William Pettersson, Tommy Andersson, Lisa Burnapp, Pavel Chromy, Pablo Delgado, Piotr Dworczak, Bernadette Haase, Aline Hemke, Rachel Johnson, Xenia Klimentova, Dirk Kuypers, Alessandro Nanni Costa, Bart Smeulders, Frits C. R. Spieksma, María O. Valentín, and Ana Viana. Modelling and optimisation in European kidney exchange programmes. *European Journal of Operational Research*, 291(2):447–456, June 2021.

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

References IV

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

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

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

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

References V

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, March 1979. Preliminary version in *STOC '74*.

[Cry] CryptoMiniSat. `https://github.com/msoos/cryptominisat/`.

[DBB17] Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe. Symmetric explanation learning: Effective dynamic symmetry handling for SAT. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 83–100. Springer, August 2017.

[DBBD16] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Improved static symmetry breaking for SAT. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *Lecture Notes in Computer Science*, pages 104–122. Springer, July 2016.

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

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

References VI

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

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

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, March 2006.

[Fle20] Mathias Fleury. *Formalization of Logical Calculi in Isabelle/HOL*. PhD thesis, Universität des Saarlandes, 2020. Available at `https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/28722`.

References VII

[GMM$^+$20] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.

[GMN20] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 1134–1140, July 2020.

[GMN22] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In *Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP '22)*, August 2022. To appear.

[GMNO22] Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, August 2022. To appear.

References VIII

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

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

[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.

[GSVW14] Maria Garcia de la Banda, Peter J. Stuckey, Pascal Van Hentenryck, and Mark Wallace. The future of optimization technology. *Constraints*, 19(2):126–138, April 2014.

References IX

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

[HHW13a] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, pages 181–188, October 2013.

[HHW13b] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, June 2013.

[HHW15] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Expressing symmetry breaking in DRAT proofs. In *Proceedings of the 25th International Conference on Automated Deduction (CADE-25)*, volume 9195 of *Lecture Notes in Computer Science*, pages 591–606. Springer, August 2015.

[HKB17] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Short proofs without new variables. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 130–147. Springer, August 2017.

References X

[JHB12] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR '12)*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, June 2012.

[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.

[Lin] Lingeling, Plingeling and Treengeling. `http://fmv.jku.at/lingeling/`.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, December 1976. Preliminary version in *STOC '75*.

[MML14] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, July 2014.

References XI

[MMNS11] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, May 2011.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[MO12] David F. Manlove and Gregg O'Malley. Paired and altruistic kidney donation in the UK: Algorithms and experimentation. In *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA '12)*, volume 7276 of *Lecture Notes in Computer Science*, pages 271–282. Springer, June 2012.

[MS96] João P. Marques-Silva and Karem A. Sakallah. GRASP—a new search algorithm for satisfiability. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '96)*, pages 220–227, November 1996.

[PR16] Tobias Philipp and Adrián Rebola-Pardo. DRAT proofs for XOR reasoning. In *Proceedings of the 15th European Conference on Logics in Artificial Intelligence (JELIA '16)*, volume 10021 of *Lecture Notes in Computer Science*, pages 415–429. Springer, November 2016.

References XII

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, February 1980.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[RSA78] Ron L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[SN15] Masahiko Sakai and Hidetomo Nabeshima. Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. *IEICE Transactions on Information and Systems*, 98-D(6):1121–1127, June 2015.

[TD20] Rodrigue Konan Tchinda and Clémentin Tayou Djamégni. On certifying the UNSAT result of dynamic symmetry-handling-based SAT solvers. *Constraints*, 25(3–4):251–279, December 2020.

References XIII

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.

[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`.

[Ver] VeriPB: Verifier for pseudo-Boolean proofs. `https://gitlab.com/MIAOresearch/VeriPB`.

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