Proofs, Proof Logging, Trust, and Certification

Jakob Nordström

University of Copenhagen
and Lund University

Extending the Synergies Between SAT and Description Logics
Schloss Dagstuhl — Leibniz-Zentrum für Informatik
September 9, 2021

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]

- $$\begin{aligned} 25957 &\equiv 1 \pmod{2} & \quad 25957 &\equiv 0 \pmod{101} \\ 25957 &\equiv 1 \pmod{3} & 25957 &\equiv 1 \pmod{103} \\ 25957 &\equiv 2 \pmod{5} & &\vdots \\ &\vdots & 25957 &\equiv 0 \pmod{257} \\ 25957 &\equiv 19 \pmod{99} & &\vdots \end{aligned}$$
  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: A proof should be 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 $P(x, \pi) = 0$.

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 Story of Combinatorial Solving

- Rich field of math and computer science
- Impact far beyond math/CS in other areas of science and also industry
- Typically very challenging problems mathematically speaking (NP-complete or worse)
- Show up all over the place, e.g.:
  - airline scheduling
  - logistics
  - hardware verification
  - donor-recipients matching for kidney transplants [MO12, BvdKM$^+$21]
- 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]

But Can We Trust the Results?

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
- Checking that a solution is feasible should be straightforward
- But how to check the absence of solutions?
- Or that a solution is optimal?

What can be done about this?

- **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 this?

- **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 this?

- **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:

- Run solver on a problem
- Feed solution + proof to proof checker
- Verify that proof checker says solution is correct and/or optimal

Proof Logging with Certifying Solvers

Workflow:

- Run solver on a problem
- Feed solution + proof to proof checker
- Verify that proof checker says solution is correct and/or optimal

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

Workflow:

- Run solver on a problem
- Feed solution + proof to proof checker
- Verify that proof checker says solution is correct and/or optimal

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
And, in fact, even for some advanced SAT solving techniques

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:

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

Decision

Free choice to assign value to variable

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

Decision

Free choice to assign value to variable

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

$\boxed{u \stackrel{\overline{u} \lor w}{=} 0}$

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

$w \overset{\mathsf{d}}{=} 0$

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Until satisfying assignment or conflict clause

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

$\boxed{u \stackrel{\overline{u} \lor w}{=} 0}$

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Until satisfying assignment or conflict clause

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Until satisfying assignment or conflict clause

Variable Assignments

Two kinds of assignments — illustrate on example formula:

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

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

$\boxed{u \stackrel{\overline{u} \lor w}{=} 0}$

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

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

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

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

Decision

Free choice to assign value to variable

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

Unit propagation

Forced choice to avoid falsifying clause

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

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

Always propagate if possible, otherwise decide

Until satisfying assignment or conflict clause

Conflict-Driven Clause Learning

Time to analyse this conflict!

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

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

$\boxed{u \overset{\overline{u} \lor w}{=} 0}$

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

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

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

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

Conflict-Driven Clause Learning

Time to analyse this conflict!

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

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

$\boxed{u \overset{\overline{u} \lor w}{=} 0}$

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

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

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

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

Could backtrack by flipping last decision

Conflict-Driven Clause Learning

Time to analyse this conflict!

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

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

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

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

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

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

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

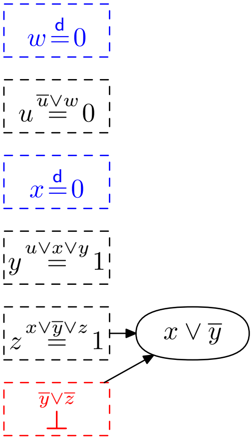
Could backtrack by flipping last decision

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

Conflict-Driven Clause Learning

Time to analyse this conflict!

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



Could backtrack by flipping last decision

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

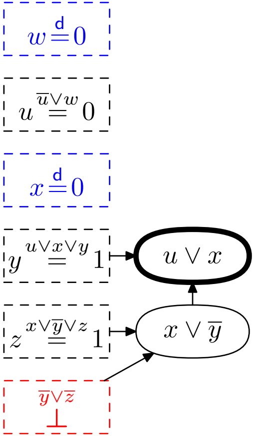
Case analysis over z for last two clauses:

- $x \vee \overline{y} \vee z$ wants $z = 1$
- $\overline{y} \vee \overline{z}$ wants $z = 0$
- Merge & remove z — must satisfy $x \vee \overline{y}$

Conflict-Driven Clause Learning

Time to analyse this conflict!

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



Could backtrack by 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:

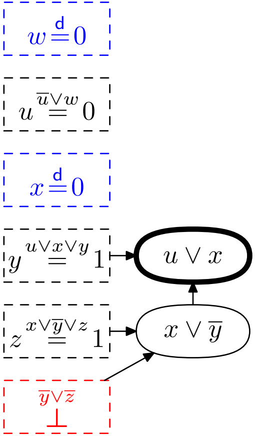
- $x \lor \overline{y} \lor z$ wants $z = 1$
- $\overline{y} \lor \overline{z}$ wants $z = 0$
- Merge & remove z — must satisfy $x \lor \overline{y}$

Repeat until only 1 variable after last decision — learn that clause (1UIP) and backjump

Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

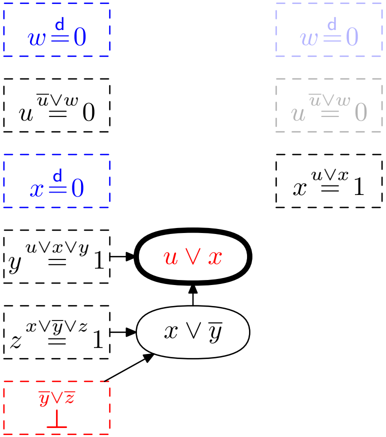
$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

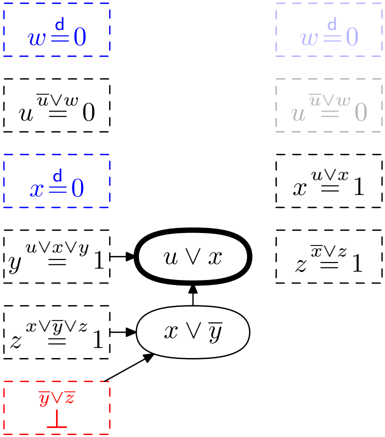
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

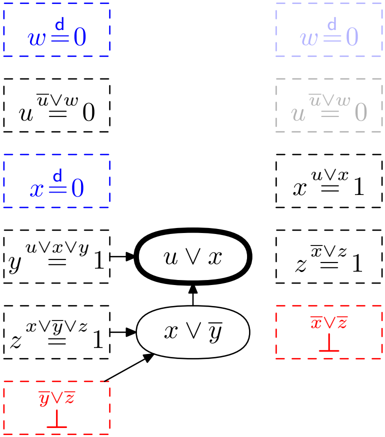
$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

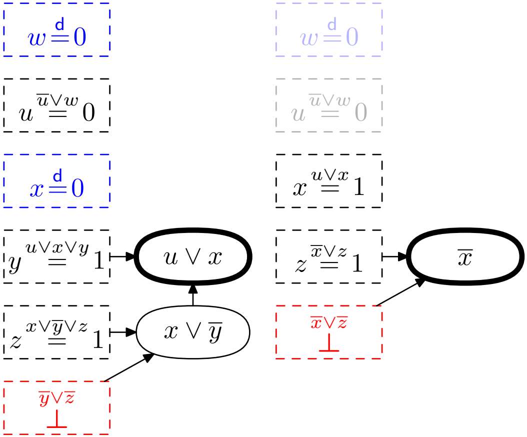
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

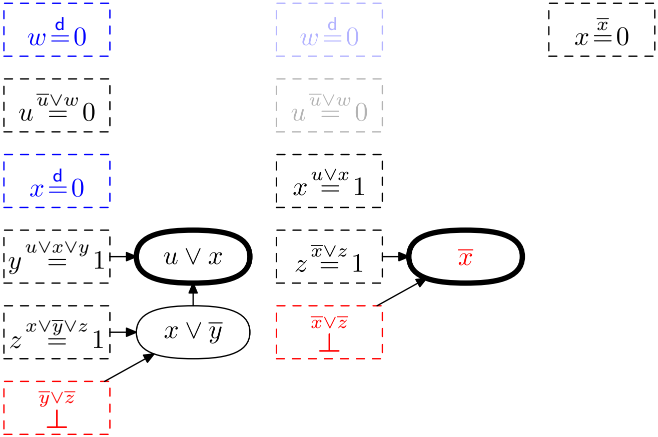
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

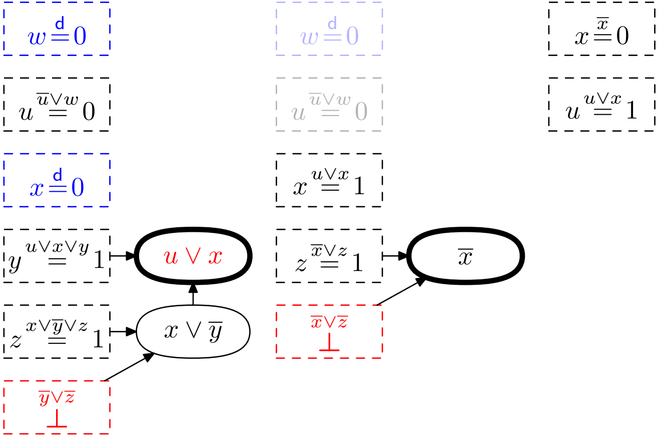
$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

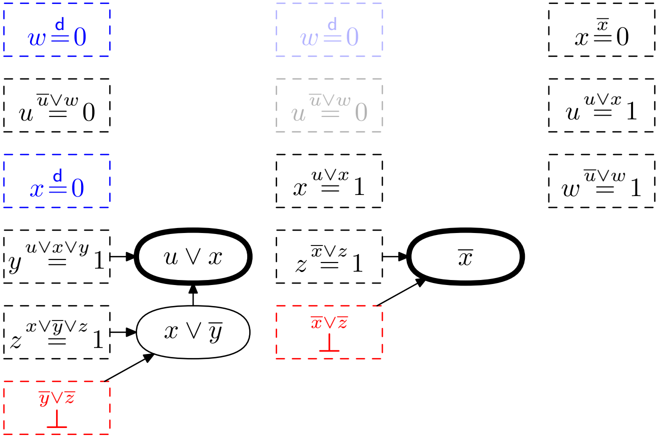
$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

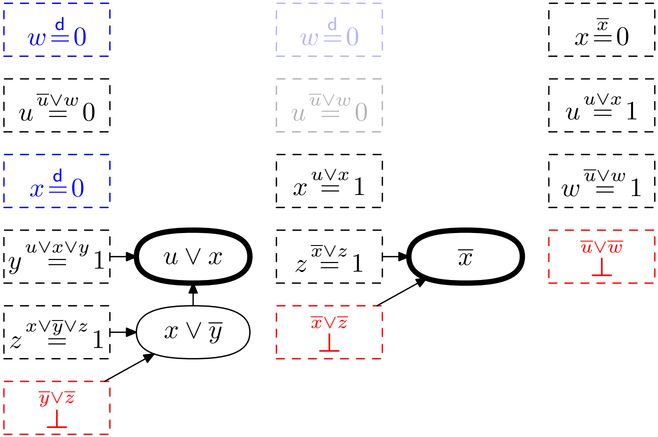
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

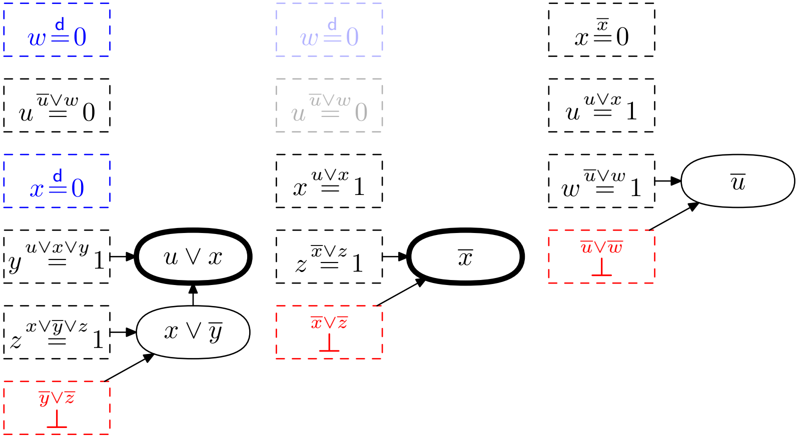
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

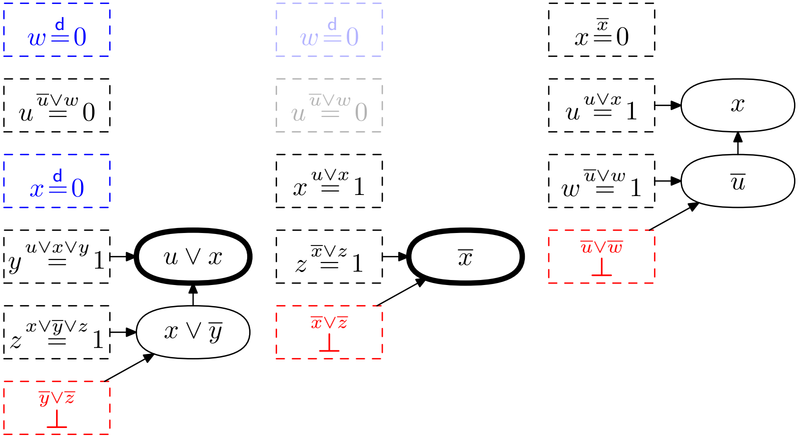
$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

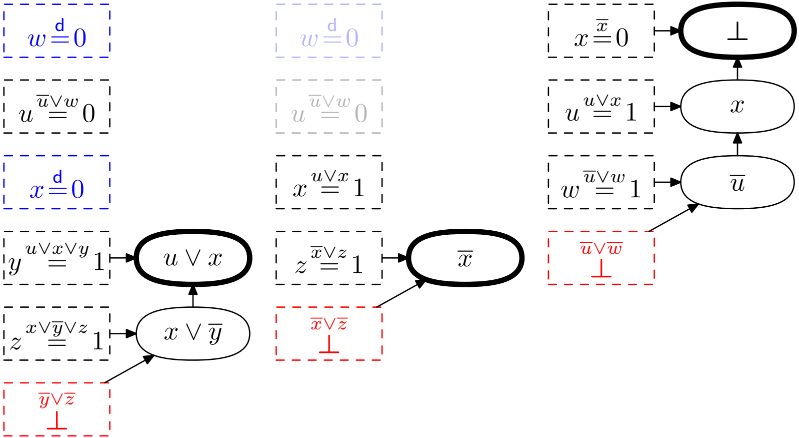
$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$



Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

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



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**

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

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

CDCL Reasoning and the Resolution Proof System

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**

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

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

When run on unsatisfiable formula, CDCL generates resolution proof*
(So lower bounds on proof size $\Rightarrow$ lower bounds on running time)

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**

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

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

When run on unsatisfiable formula, CDCL generates resolution proof*
(So lower bounds on proof size $\Rightarrow$ lower bounds on running time)

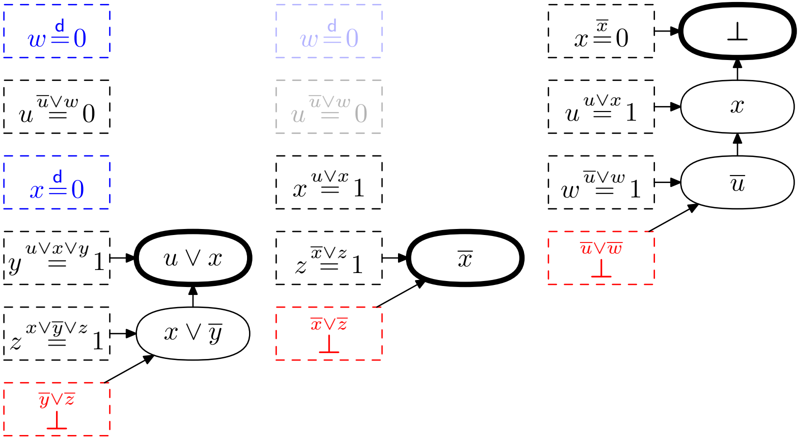
(*) Ignores pre- and inprocessing, but we will get there...

Resolution Proofs from CDCL Executions

Obtain resolution proof. . .

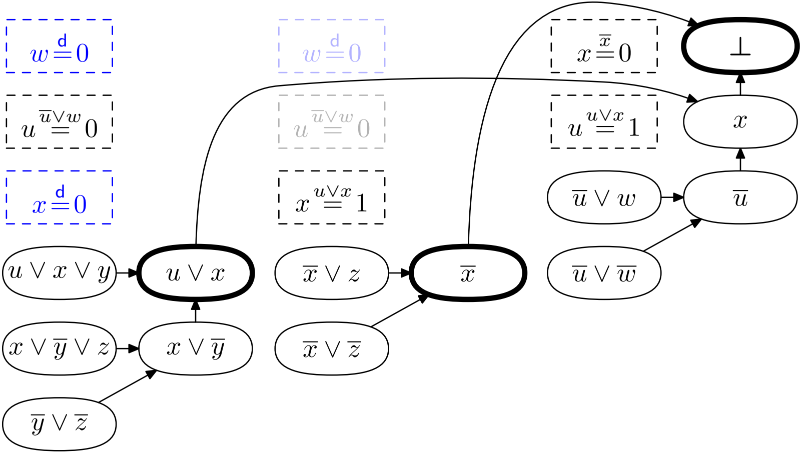
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution...



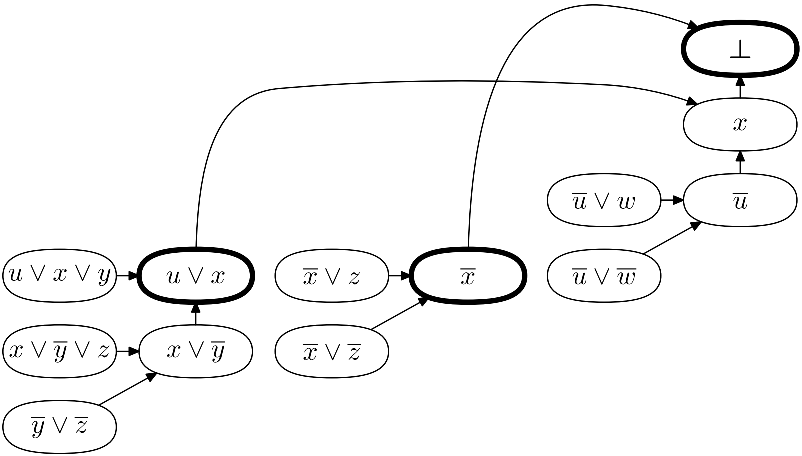
Resolution Proofs from CDCL Executions

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

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

C is a 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

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

C is a 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

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

RUP Proofs

So shorter proof of unsatisfiability for

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

is sequence of RUP clauses

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

Requires a little bit more trust, though
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

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

Introduce clauses

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

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

CDCL pre- and inprocessing could to 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 could to steps like this

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

Substitution Redundancy

- 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]

Substitution Redundancy

- 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]

Substitution redundancy [BT19, GN21]

C is redundant with respect to F if and only if there is a substitution ω, called a *witness*, for which it holds that

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

Substitution Redundancy

- 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]

Substitution redundancy [BT19, GN21]

C is redundant with respect to F if and only if there is a substitution ω, called a *witness*, for which it holds that

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

- 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 (e.g., all clauses in $(F \wedge C)\restriction_{\omega}$ RUP)

Deriving $a \leftrightarrow (x \wedge y)$ with Substitution Redundancy

Want to derive

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

using substitution redundancy condition $F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$

Deriving $a \leftrightarrow (x \wedge y)$ with Substitution Redundancy

Want to derive

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

using substitution redundancy condition $F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$

1. $F \wedge \neg(a \vee \overline{x} \vee \overline{y}) \models (F \wedge (a \vee \overline{x} \vee \overline{y}))\restriction_{\omega}$
   Any satisfying α must set $\{a \mapsto 0, x \mapsto 1, y \mapsto 1\}$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new clause satisfied

Deriving $a \leftrightarrow (x \land y)$ with Substitution Redundancy

Want to derive

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

using substitution redundancy condition $F \land \neg C \models (F \land C)\restriction_\omega$

1. $F \land \neg(a \lor \overline{x} \lor \overline{y}) \models (F \land (a \lor \overline{x} \lor \overline{y}))\restriction_\omega$
   Any satisfying α must set $\{a \mapsto 0, x \mapsto 1, y \mapsto 1\}$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new clause satisfied
2. $F \land (a \lor \overline{x} \lor \overline{y}) \land \neg(\overline{a} \lor x) \models (F \land (a \lor \overline{x} \lor \overline{y}) \land (\overline{a} \lor x))\restriction_\omega$
   Any satisfying α must set $\{a \mapsto 1, x \mapsto 0\}$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new clauses satisfied

Deriving $a \leftrightarrow (x \wedge y)$ with Substitution Redundancy

Want to derive

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

using substitution redundancy condition $F \wedge \neg C \models (F \wedge C)\restriction_{\omega}$

1. $F \wedge \neg(a \vee \overline{x} \vee \overline{y}) \models (F \wedge (a \vee \overline{x} \vee \overline{y}))\restriction_{\omega}$
   Any satisfying α must set $\{a \mapsto 0, x \mapsto 1, y \mapsto 1\}$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new clause satisfied
2. $F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge \neg(\overline{a} \vee x) \models (F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x))\restriction_{\omega}$
   Any satisfying α must set $\{a \mapsto 1, x \mapsto 0\}$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new clauses satisfied
3. $F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x) \wedge \neg(\overline{a} \vee y) \models (F \wedge (a \vee \overline{x} \vee \overline{y}) \wedge (\overline{a} \vee x) \wedge (\overline{a} \vee y))\restriction_{\omega}$
   Any satisfying α must set $\{a \mapsto 1, x \mapsto 1, y \mapsto 0\}$
   Choose $\omega = \{a \mapsto 0\}$ — F untouched; new clauses satisfied

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

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

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

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

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 solver LINGELING [Lin], but no 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 $1 = \mathit{true}$

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

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

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 Proof Logging

Cutting planes proof system [CCT87]

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

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B \geq 0]$$

Division $$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c > 0]$$

Pseudo-Boolean Proof Logging

Cutting planes proof system [CCT87]

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

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B \geq 0]$$

Division $$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c > 0]$$

Combine with **substitution redundancy** rule

Pseudo-Boolean Proof Logging

Cutting planes proof system [CCT87]

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

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B} \quad [c_A, c_B \geq 0]$$

Division $$\frac{\sum_i c a_i \ell_i \geq A}{\sum_i a_i \ell_i \geq \lceil A/c \rceil} \quad [c > 0]$$

Combine with **substitution redundancy** rule

Yields VeriPB proof system [EGMN20, GMN20, GMM+20, GN21]
(Now trusting 0-1 linear inequalities instead of just clauses)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$s_{4,2}$

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

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

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

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

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

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

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

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

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

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

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

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

$s_{4,2}$

How to know translation correct?
VeriPB can certify pseudo-Boolean-to-CNF rewriting [GMN21]

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

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

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 XOR reasoning:

$$x + y + z = 1 \pmod{2}$$

$$y + z + w = 1 \pmod{2}$$

imply

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

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 XOR 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]

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 XOR 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!

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 XOR 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 substitution redundancy and 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 substitution redundancy and 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 substitution redundancy and 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 substitution redundancy and 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

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

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

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

Interesting challenges for proof logging!

Challenges Beyond SAT

Proof logging for combinatorial optimization

- Maximum satisfiability (MaxSAT) solving
- Pseudo-Boolean optimization
- Mixed integer linear programming (some work in [CGS17, EG21])
- Constraint programming (some work in [EGMN20, GMN20, GMM+20])

Challenges Beyond SAT

Proof logging for combinatorial optimization

- Maximum satisfiability (MaxSAT) solving
- Pseudo-Boolean optimization
- Mixed integer linear programming (some work in [CGS17, EG21])
- Constraint programming (some work in [EGMN20, GMN20, GMM+20])

Explainability, robustness, fairness

- Certified explanations for decisions or classifications?
- Formal proofs that neural networks are robust to limited perturbations of the input?
- Rigorous symbolic reasoning about fairness?

Challenges Beyond SAT

Proof logging for combinatorial optimization

- Maximum satisfiability (MaxSAT) solving
- Pseudo-Boolean optimization
- Mixed integer linear programming (some work in [CGS17, EG21])
- Constraint programming (some work in [EGMN20, GMN20, GMM+20])

Explainability, robustness, fairness

- Certified explanations for decisions or classifications?
- Formal proofs that neural networks are robust to limited perturbations of the input?
- Rigorous symbolic reasoning about fairness?

And more...

- Lots of challenging problems and interesting ideas
- This talk would (hopefully) sound quite different in a year or two

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 might hit a sweet spot between simplicity and expressibility
- Some very recent promising results in this direction

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 might hit a sweet spot between simplicity and expressibility
- Some very recent promising results in this direction

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.

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

References II

[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.

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

References III

[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.

[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.

References IV

[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.

[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/`.

References V

[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.

[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.

References VI

[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`.

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

References VII

[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.

[GMN21] Stephan Gocht, Ruben Martins, and Jakob Nordström. Certifying CNF encodings of pseudo-Boolean constraints. Work in progress presented at the workshop *Pragmatics of SAT 2021*, July 2021.

[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.

References VIII

[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.

[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.

References IX

[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.

[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/`.

References X

[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.

[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.

References XI

[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.

[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.

References XII

[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.

[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`.

[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 Internatjuional Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, July 2014.