Leveraging Computational Complexity Theory for Provably Correct Combinatorial Optimization

Jakob Nordström

University of Copenhagen
and Lund University

2023 North American Annual Meeting
of the Association for Symbolic Logic
March 25, 2023

The Success Story of Combinatorial Solving and Optimization

- Rich field of mathematics and computer science
- Impact in other areas of science and also industry, e.g.:
  - airline scheduling
  - hardware verification
  - donor-recipients matching for kidney transplants [MO12, BvdKM^{+}21]
- Computationally very challenging problems (NP-complete or worse)
- Lots of effort last couple of decades spent on 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, BMN22]
- Even worse: No way of knowing for sure when errors happen
- Solvers even get feasibility of solutions wrong (though this should be straightforward!)
- But how to check the absence of solutions?
- Or that a solution is optimal? (Even off-by-one mistakes can snowball into large errors if solver used as subroutine)

What Can Be Done About Solver Bugs?

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

What Can Be Done About Solver Bugs?

- **Software testing**
  Hard to get good test coverage for sophisticated solvers
  Inherently can only detect presence of bugs, not absence
- **Formal verification**
  Prove that solver implementation adheres to formal specification
  Current techniques cannot scale to this level of complexity

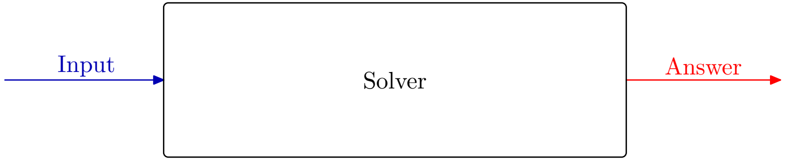
What Can Be Done About Solver Bugs?

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

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

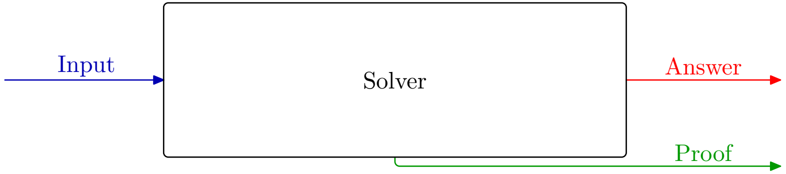
- **Proof logging**
  Make solver certifying [ABM$^+$11, MMNS11] by outputting
  1. not only answer but also
  2. simple, machine-verifiable proof that answer is correct

Proof Logging with Certifying Solvers: Workflow



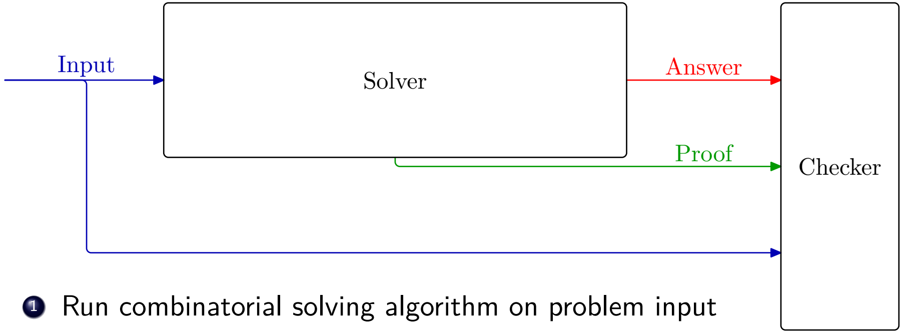
1. Run combinatorial solving algorithm on problem input

Proof Logging with Certifying Solvers: Workflow



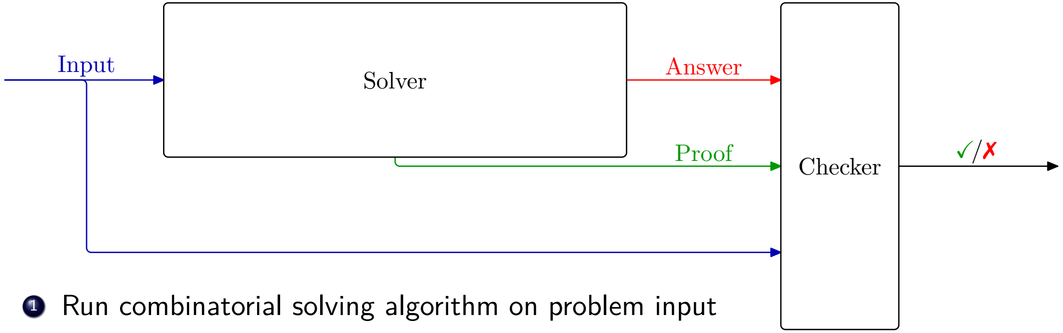
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof

Proof Logging with Certifying Solvers: Workflow



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

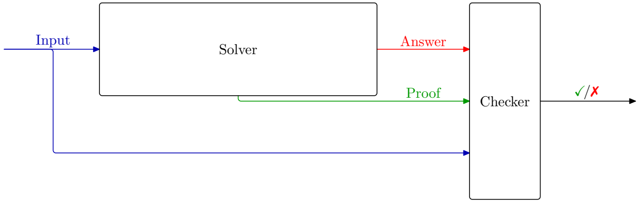
Proof Logging with Certifying Solvers: Workflow



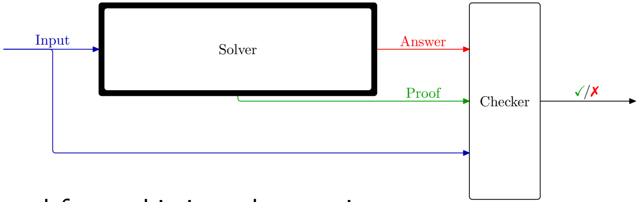
1. Run combinatorial solving algorithm on problem input
2. Get as output not only answer but also proof
3. Feed input + answer + proof to proof checker
4. Verify that proof checker says answer is correct

Proof Logging Wishlist

Proof format for certifying solver should be



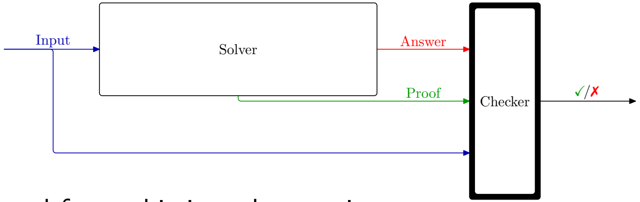
Proof Logging Wishlist



Proof format for certifying solver should be

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

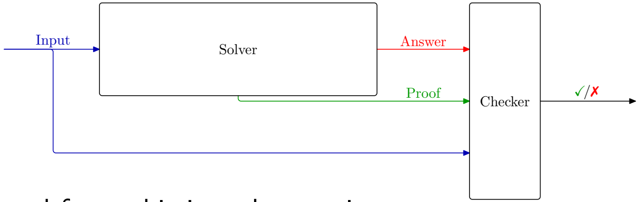
Proof Logging Wishlist



Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning
- **dead simple:** checking correctness of proofs should be trivial

Proof Logging Wishlist

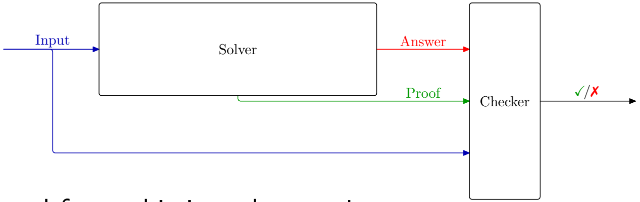


Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning
- **dead simple:** checking correctness of proofs should be trivial

Clear conflict expressivity vs. simplicity!

Proof Logging Wishlist



Proof format for certifying solver should be

- **very powerful:** minimal overhead for sophisticated reasoning
- **dead simple:** checking correctness of proofs should be trivial

Clear conflict expressivity vs. simplicity!

Asking for both perhaps a little bit too good to be true?

This Talk

Proof logging for combinatorial optimization is possible!

This Talk

Proof logging for combinatorial optimization is possible!

- Only need propositional logic
- Represent constraints as 0–1 integer linear inequalities
- Formalize reasoning using extended resolution [Tse68] and cutting planes [CCT87] proof systems
- Add well-chosen strengthening rules [Goc22, GN21, BGMN22]
- Implemented in VeriPB (`https://gitlab.com/MIAOresearch/software/VeriPB`)

This Talk

Proof logging for combinatorial optimization is possible!

- Only need propositional logic
- Represent constraints as 0–1 integer linear inequalities
- Formalize reasoning using extended resolution [Tse68] and cutting planes [CCT87] proof systems
- Add well-chosen strengthening rules [Goc22, GN21, BGMN22]
- Implemented in VeriPB (`https://gitlab.com/MIAOresearch/software/VeriPB`)

Making constructive use of computational complexity theory!

Outline of This Talk

Combinatorial Problems (1/2)

Boolean satisfiability (SAT)
Decide if exists satisfying assignment to conjunctive normal form (CNF) formula

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

Combinatorial Problems (1/2)

Boolean satisfiability (SAT)
Decide if exists satisfying assignment to conjunctive normal form (CNF) formula

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

SAT-based optimization (MaxSAT)
Minimize 0–1 linear expression

$$p + q + 2\overline{r} + 3u + 5\overline{w}$$

subject to constraints in a CNF formula

Combinatorial Problems (2/2)

Mixed integer linear programming (MIP)
Minimize $\sum_i w_i x_i$ subject to $A\mathbf{x} \geq \mathbf{b}$
Variables x_i Boolean, integral, or real-valued

Combinatorial Problems (2/2)

Mixed integer linear programming (MIP)
Minimize $\sum_i w_i x_i$ subject to $A\mathbf{x} \geq \mathbf{b}$
Variables x_i Boolean, integral, or real-valued

Constraint programming (CP)
Also non-Boolean variables
More expressive constraints (e.g., all-different)

Combinatorial Problems (2/2)

Mixed integer linear programming (MIP)
Minimize $\sum_i w_i x_i$ subject to $A\mathbf{x} \geq \mathbf{b}$
Variables x_i Boolean, integral, or real-valued

Constraint programming (CP)
Also non-Boolean variables
More expressive constraints (e.g., all-different)

Satisfiability modulo theories (SMT)
Propositional logic formula with variables express statements in theories, e.g.:

- uninterpreted functions
- linear arithmetic
- arrays

Computational Complexity of Combinatorial Problems

- These problems are NP-complete [Coo71, Lev73] or worse
- Believed to require exponential time in the worst case [IP01, CIP09]
- Proving such lower bounds is the goal of computational complexity theory [GW08]
- Has not stopped practitioners from solving problems very efficiently in practice (often, not always)

What Is a Proof? (From a Computational Perspective)

Claim: "N is the product of two primes" (think $N = 25957$, say)

What is an acceptable proof of such a claim?

What Is a Proof? (From a Computational Perspective)

Claim: "N is the product of two primes" (think $N = 25957$, 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 that this is hard [RSA78]

What Is a Proof? (From a Computational Perspective)

Claim: "N is the product of two primes" (think $N = 25957$, 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 that this is hard [RSA78]

- $$\begin{array}{llllll} 25957 \equiv 1 & \pmod{2} & 25957 \equiv 19 & \pmod{99} & 25957 \equiv 202 & \pmod{255} \\ 25957 \equiv 1 & \pmod{3} & 25957 \equiv 0 & \pmod{101} & 25957 \equiv 0 & \pmod{257} \\ 25957 \equiv 2 & \pmod{5} & 25957 \equiv 1 & \pmod{103} & 25957 \equiv 57 & \pmod{259} \\ \vdots & & \vdots & & \vdots & \end{array}$$
  OK, but maybe even a bit of overkill

What Is a Proof? (From a Computational Perspective)

Claim: "N is the product of two primes" (think $N = 25957$, 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 that this is hard [RSA78]

- $$\begin{array}{llllll} 25957 \equiv 1 & \pmod{2} & 25957 \equiv 19 & \pmod{99} & 25957 \equiv 202 & \pmod{255} \\ 25957 \equiv 1 & \pmod{3} & 25957 \equiv 0 & \pmod{101} & 25957 \equiv 0 & \pmod{257} \\ 25957 \equiv 2 & \pmod{5} & 25957 \equiv 1 & \pmod{103} & 25957 \equiv 57 & \pmod{259} \\ \vdots & & \vdots & & \vdots & \end{array}$$
  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 [CR79] of "true claims":

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

- for all $x \in L$ there exists 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 [CR79] of "true claims":

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

- for all $x \in L$ there exists 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 [CR79] of "true claims":

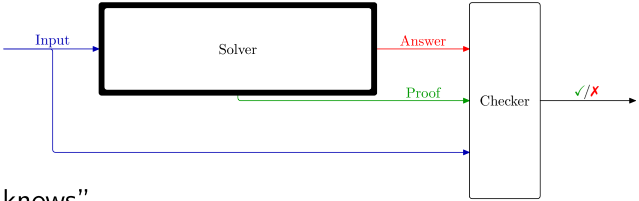
Deterministic algorithm $P(x, \pi)$ that runs in time polynomial in $|x|$ and $|\pi|$ such that

- for all $x \in L$ there exists 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 running time in $|x| + |\pi|$
Goal of proof complexity: establish lower bounds on proof size

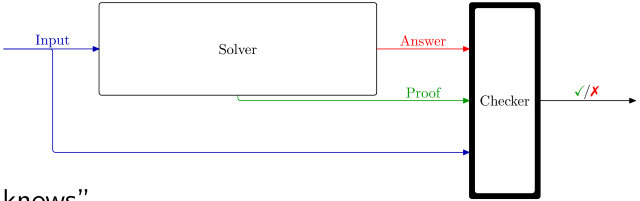
Proof Logging with Certifying Solvers: Practical Requirements



Proof logging should

- work for existing algorithms
- incur minimal overhead
- only use what solver "already knows"

Proof Logging with Certifying Solvers: Practical Requirements



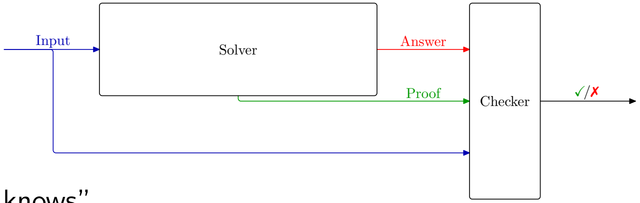
Proof logging should

- work for existing algorithms
- incur minimal overhead
- only use what solver “already knows”

Proof checking should

- scale linearly with solver running time
- not require knowledge of inner workings of solver
- be very easy (so that proof checker can be trusted, or even formally verified)

Proof Logging with Certifying Solvers: Practical Requirements



Proof logging should

- work for existing algorithms
- incur minimal overhead
- only use what solver "already knows"

Proof checking should

- scale linearly with solver running time
- not require knowledge of inner workings of solver
- be very easy (so that proof checker can be trusted, or even formally verified)

Fully automated process — no proof assistants
Higher-order logics too complicated and/or too slow(?)

The Sales Pitch for Proof Logging

1. ***Certifies correctness*** of solver output
2. ***Detects errors*** even if due to compiler bugs, hardware failures, or cosmic rays
3. Helps with ***debugging*** during development [EG21, GMM^{+}20, KM21, BBN^{+}23]
4. Facilitates ***performance analysis***
5. Helps identify potential for ***further improvements***
6. Enables ***auditability*** by third parties
7. Serves as stepping stone towards ***explainability***

The Sales Pitch for Proof Logging

1. ***Certifies correctness*** of solver output
2. ***Detects errors*** even if due to compiler bugs, hardware failures, or cosmic rays
3. Helps with ***debugging*** during development [EG21, GMM+20, KM21, BBN+23]
4. Facilitates ***performance analysis***
5. Helps identify potential for ***further improvements***
6. Enables ***auditability*** by third parties
7. Serves as stepping stone towards ***explainability***
8. Can ***validate computer-generated proofs*** in mathematics [HK17]

The Proof Logging Story So Far

Huge success for Boolean satisfiability (SAT) solving

- Proof formats such as
  - DRAT [HHW13a, HHW13b, WHH14]
  - GRIT [CMS17]
  - LRAT [CHH$^+$17]
- Compulsory DRAT proof logging in main track of SAT competition

The Proof Logging Story So Far

Huge success for Boolean satisfiability (SAT) solving

- Proof formats such as
  - DRAT [HHW13a, HHW13b, WHH14]
  - GRIT [CMS17]
  - LRAT [CHH$^+$17]
- Compulsory DRAT proof logging in main track of SAT competition

But has remained out of reach for stronger combinatorial solving paradigms

The Proof Logging Story So Far

Huge success for Boolean satisfiability (SAT) solving

- Proof formats such as
  - DRAT [HHW13a, HHW13b, WHH14]
  - GRIT [CMS17]
  - LRAT [CHH$^+$17]
- Compulsory DRAT proof logging in main track of SAT competition

But has remained out of reach for stronger combinatorial solving paradigms

And, in fact, even for advanced SAT solving techniques such as

- cardinality detection
- parity reasoning
- symmetry breaking

The Boolean Satisfiability (SAT) Problem

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

The SAT problem

Given a CNF formula F, is it satisfiable?

For instance, what about:

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

Proofs for SAT

For satisfiable instances: just specify a satisfying assignment

For unsatisfiability: a sequence of clauses

- Each clause follows "obviously" from everything we know so far
- Final clause is empty, meaning contradiction (written $\perp$)
- Means original formula must be inconsistent

What Is Obvious? Unit Propagation

Unit propagation

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

What Is Obvious? Unit Propagation

Unit propagation

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

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

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

What Is Obvious? Unit Propagation

Unit propagation

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

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

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

What Is Obvious? Unit Propagation

Unit propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit propagation

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

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

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

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

Proof checker should know how to unit propagate until saturation

Davis-Putman-Logemann-Loveland (DPLL)

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

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

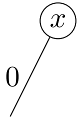
$$(p \lor \overline{u}) \land (q \lor r) \land (\overline{r} \lor w) \land (u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{p} \lor \overline{u})$$

Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

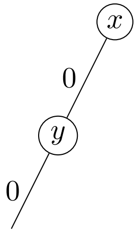


Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

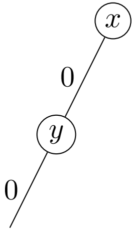


Davis-Putman-Logemann-Loveland (DPLL)

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

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

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



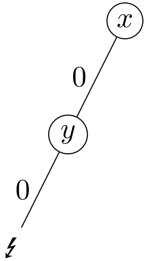
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

1. $x \lor y$



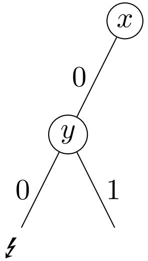
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

1. $x \lor y$



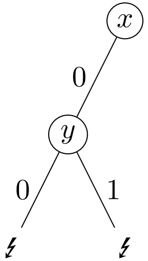
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



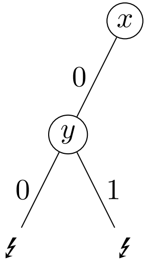
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



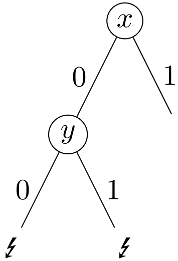
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



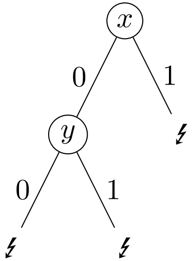
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



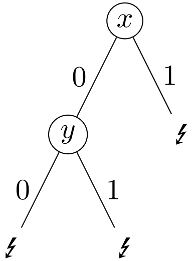
Davis-Putman-Logemann-Loveland (DPLL)

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

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

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

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



Reverse Unit Propagation (RUP)

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

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

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

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

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

Reverse Unit Propagation (RUP)

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

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

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

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

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

- Backtrack clauses from DPLL solver generate RUP proofs

Reverse Unit Propagation (RUP)

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

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

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

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

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

- Backtrack clauses from DPLL solver generate RUP proofs
- True also for learned clauses in modern conflict-driven clause learning (CDCL) SAT solvers [MS96, BS97, MMZ^{+}01]

Writing Proofs in the DRAT Format

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

Writing Proofs in the DRAT Format

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

Formula in DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

Writing Proofs in the DRAT Format

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

Formula in DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \lor y$

$x \lor \overline{y}$

x

$\overline{x}$

$\perp$

Writing Proofs in the DRAT Format

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

Formula in DIMACS

```
p cnf 8 9
1 -4 0
2 3 0
-2 5 0
4 6 7 0
6 -7 8 0
-6 8 0
-7 -8 0
-6 -8 0
-1 -4 0
```

DPLL Proof in RUP

$x \lor y$

$x \lor \overline{y}$

x

$\overline{x}$

$\perp$

DPLL Proof in DRAT

```
6 7 0
6 -7 0
6 0
-6 0
0
```

More Ingredients in Proof Logging for SAT

Fact

RUP proofs are shorthand for so-called resolution proofs

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

But RUP and resolution aren't enough for preprocessing, inprocessing, and some other kinds of reasoning

Extension Variables

Suppose SAT solver preprocessor wants to introduce new, fresh variable a encoding

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

Extended resolution: allow to introduce clauses

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

Should be fine, so long as a doesn't appear anywhere previously

Fact

Extended resolution (RUP + definition of new variables) is essentially equivalent to the DRAT proof logging system

Why Aren’t We Done?

Practical limitations of SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently
- Clausal proofs can’t easily reflect what other algorithms do

Why Aren't We Done?

Practical limitations of SAT proof logging technology:

- Difficulties dealing with stronger reasoning efficiently
- Clausal proofs can't easily reflect what other algorithms do

Surprising claim: a slight change to 0-1 integer linear inequalities does the job!
Can support proof logging for

- Graph reasoning without knowing what a graph is
- Constraint programming without knowing, e.g., what an integer variable is
- Advanced SAT techniques so far beyond reach for efficient DRAT proof logging

Pseudo-Boolean Constraints

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

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

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

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

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

Cutting Planes Proof System [CCT87]

Input axioms From the input

Cutting Planes Proof System [CCT87]

Input axioms From the input

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

Cutting Planes Proof System [CCT87]

Input axioms From the input

Literal axioms

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

Addition

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

Cutting Planes Proof System [CCT87]

Input axioms — From the input

Literal axioms

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

Addition

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

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

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

Cutting Planes Proof System [CCT87]

Input axioms — From the input

Literal axioms

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

Addition

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

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

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

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

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Cutting Planes Toy Example

$$
\text{Div by } 3\ \dfrac{\text{Add}\ \dfrac{\text{Add}\ \dfrac{\text{Mul by } 2\ \dfrac{w+2x+y\geq 2}{2w+4x+2y\geq 4} \qquad w+2x+4y+2z\geq 5}{3w+6x+6y+2z\geq 9} \qquad \dfrac{\overline{z}\geq 0}{2\overline{z}\geq 0}\ \text{Mul by } 2}{3w+6x+6y \qquad \geq 7}}{w+2x+2y\geq 2\tfrac{1}{3}}
$$

Cutting Planes Toy Example

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

Cutting Planes Toy Example

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

Such a calculation can be written in a proof line assuming handles

$$\begin{aligned} C_1 &\doteq 2x+y+w \geq 2 \\ C_2 &\doteq 2x+4y+2z+w \geq 5 \\ Ax(\overline{z}) &\doteq \overline{z} \geq 0 \end{aligned}$$

Cutting Planes Toy Example

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

Such a calculation can be written in a proof line assuming handles

$$\begin{aligned} C_1 &\doteq 2x+y+w\geq 2\\ C_2 &\doteq 2x+4y+2z+w\geq 5\\ Ax(\overline{z}) &\doteq \overline{z}\geq 0 \end{aligned}$$

using postfix notation something like

$$C_1\;\;2\;\;\texttt{Mul}\;\;C_2\;\;\texttt{Add}\;\;Ax(\overline{z})\;\;2\;\;\texttt{Mul}\;\;\texttt{Add}\;\;3\;\;\texttt{Div}$$

Pseudo-Boolean Proofs

For satisfiable instances: just specify a satisfying assignment

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

Each constraint follows "obviously" from what is known so far

- Either implicitly, by (generalization of) RUP...
- Or by an explicit cutting planes derivation...
- Or by (generalization of) extension rule

Final constraint is $0 \geq 1$

Proof Logging for Graph Solving and Constraint Programming

Pseudo-Boolean proof logging can also certify reasoning in

- graph solving for clique, subgraph isomorphism, and maximum common connected subgraph [GMN20, GMM$^+$20] without knowing anything about
  - vertices
  - edges
  - neighbours

Proof Logging for Graph Solving and Constraint Programming

Pseudo-Boolean proof logging can also certify reasoning in

- graph solving for clique, subgraph isomorphism, and maximum common connected subgraph [GMN20, GMM$^+$20] without knowing anything about
  - vertices
  - edges
  - neighbours
- constraint programming [EGMN20, GMN22] without knowing anything about
  - non-Boolean variables
  - arrays
  - tables

Proof Logging for Graph Solving and Constraint Programming

Pseudo-Boolean proof logging can also certify reasoning in

- graph solving for clique, subgraph isomorphism, and maximum common connected subgraph [GMN20, GMM^{+}20] without knowing anything about
    - vertices
    - edges
    - neighbours
- constraint programming [EGMN20, GMN22] without knowing anything about
    - non-Boolean variables
    - arrays
    - tables

Caveat: Need input pre-translated into 0–1 integer linear program
Such translations should be formally verified (work in progress)

Integer Variables

Represent integer a as sum of bits $\sum_i 2^i \cdot a_i$

Integer Variables

Represent integer a as sum of bits $\sum_i 2^i \cdot a_i$

Use extension rule to introduce new variables

$$a_{\geq k} \Leftrightarrow \sum_i 2^i \cdot a_i \geq k$$

$$a_{=k} \Leftrightarrow (a_{\geq k} \wedge \overline{a}_{\geq k+1})$$

Integer Variables

Represent integer a as sum of bits $\sum_i 2^i \cdot a_i$

Use extension rule to introduce new variables

$$a_{\geq k} \Leftrightarrow \sum_i 2^i \cdot a_i \geq k$$

$$k \cdot \overline{a}_{\geq k} + \sum_i 2^i \cdot a_i \geq k$$

$$\left(\sum_i 2^i - k + 1\right) \cdot a_{\geq k} + \sum_i 2^i \cdot \overline{a}_i \geq \sum_i 2^i - k + 1$$

$$a_{=k} \Leftrightarrow (a_{\geq k} \wedge \overline{a}_{\geq k+1})$$

$$2 \cdot \overline{a}_{=k} + a_{\geq k} + \overline{a}_{\geq k+1} \geq 2$$

$$a_{=k} + \overline{a}_{\geq k} + a_{\geq k+1} \geq 1$$

(with definitions represented as 0–1 inequalities)

Integer Variables

Represent integer a as sum of bits $\sum_i 2^i \cdot a_i$

Use extension rule to introduce new variables

$$a_{\geq k} \Leftrightarrow \sum_i 2^i \cdot a_i \geq k$$

$$k \cdot \overline{a}_{\geq k} + \sum_i 2^i \cdot a_i \geq k$$

$$\left(\sum_i 2^i - k + 1\right) \cdot a_{\geq k} + \sum_i 2^i \cdot \overline{a}_i \geq \sum_i 2^i - k + 1$$

$$a_{=k} \Leftrightarrow (a_{\geq k} \wedge \overline{a}_{\geq k+1})$$

$$2 \cdot \overline{a}_{=k} + a_{\geq k} + \overline{a}_{\geq k+1} \geq 2$$

$$a_{=k} + \overline{a}_{\geq k} + a_{\geq k+1} \geq 1$$

(with definitions represented as 0–1 inequalities)

Go back and forth between representations to support efficient proof logging

All-Different Propagator

$$
\begin{array}{rl}
V \in \{ & 1 \quad \quad \quad 4 \quad 5 \, \} \\
W \in \{ & 1 \quad 2 \quad 3 \quad \quad \, \} \\
X \in \{ & \quad 2 \quad 3 \quad \quad \, \} \\
Y \in \{ & 1 \quad \quad 3 \quad \quad \, \} \\
Z \in \{ & 1 \quad \quad 3 \quad \quad \, \}
\end{array}
$$

All-Different Propagator

$$\begin{array}{rclcccccl}
V & \in & \{ & 1 & & & 4 & 5 & \} \\
W & \in & \{ & 1 & 2 & 3 & & & \} \\
X & \in & \{ & & 2 & 3 & & & \} \\
Y & \in & \{ & 1 & & 3 & & & \} \\
Z & \in & \{ & 1 & & 3 & & & \}
\end{array}$$

All-Different Propagator

$$
\begin{array}{rlllllll}
V & \in \{ & 1 & & & 4 & 5 & \} \\
W & \in \{ & 1 & 2 & 3 & & & \} \\
X & \in \{ & & 2 & 3 & & & \} \\
Y & \in \{ & 1 & & 3 & & & \} \\
Z & \in \{ & 1 & & 3 & & & \}
\end{array}
\qquad w_{=1} + w_{=2} + w_{=3} \geq 1
$$

All-Different Propagator

$$\begin{array}{llllllll}
V \in \{ & 1 & & & 4 & 5 \} & & \\
W \in \{ & 1 & 2 & 3 & & \} & w_{=1} + w_{=2} + w_{=3} & \geq 1 \\
X \in \{ & & 2 & 3 & & \} & x_{=2} + x_{=3} & \geq 1 \\
Y \in \{ & 1 & & 3 & & \} & y_{=1} + y_{=3} & \geq 1 \\
Z \in \{ & 1 & & 3 & & \} & z_{=1} + z_{=3} & \geq 1
\end{array}$$

All-Different Propagator

$$\begin{array}{llllllllllllllll}
V \in \{ & 1 & & & 4 & 5 \} & & & & & & & & & & \\
W \in \{ & 1 & 2 & 3 & & \} & w_{=1} & + & w_{=2} & + & w_{=3} & & & & & \geq 1 \\
X \in \{ & & 2 & 3 & & \} & & & x_{=2} & + & x_{=3} & & & & & \geq 1 \\
Y \in \{ & 1 & & 3 & & \} & y_{=1} & & & + & y_{=3} & & & & & \geq 1 \\
Z \in \{ & 1 & & 3 & & \} & z_{=1} & & & + & z_{=3} & & & & & \geq 1
\end{array}$$

$$\begin{array}{ll}
\rightarrow & -v_{=1} + -w_{=1} + \quad -y_{=1} + -z_{=1} \geq -1 \\
\quad\rightarrow & -w_{=2} + -x_{=2} \geq -1 \\
\qquad\rightarrow & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1
\end{array}$$

All-Different Propagator

$$\begin{array}{rl}
V \in \{1 \quad\quad 4 \quad 5\} & \\
W \in \{1 \quad 2 \quad 3 \quad\quad\} & w_{=1} + w_{=2} + w_{=3} \geq 1 \\
X \in \{\quad 2 \quad 3 \quad\quad\} & x_{=2} + x_{=3} \geq 1 \\
Y \in \{1 \quad\quad 3 \quad\quad\} & y_{=1} + y_{=3} \geq 1 \\
Z \in \{1 \quad\quad 3 \quad\quad\} & z_{=1} + z_{=3} \geq 1
\end{array}$$

$$\begin{array}{rl}
\rightarrow & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} \geq -1 \\
\rightarrow & -w_{=2} + -x_{=2} \geq -1 \\
\rightarrow & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1
\end{array}$$

$$-v_{=1} \geq 1$$

All-Different Propagator

$$\begin{array}{lllllllllll}
V \in \{ 1 & & & 4 & 5 \} & & & & & & \\
W \in \{ 1 & 2 & 3 & & \} & w_{=1} + & w_{=2} + & w_{=3} & & & \geq 1 \\
X \in \{ & 2 & 3 & & \} & & x_{=2} + & x_{=3} & & & \geq 1 \\
Y \in \{ 1 & & 3 & & \} & y_{=1} & + & y_{=3} & & & \geq 1 \\
Z \in \{ 1 & & 3 & & \} & z_{=1} & + & z_{=3} & & & \geq 1
\end{array}$$

$$\begin{array}{lllllll}
\rightarrow & -v_{=1} + -w_{=1} + & & & -y_{=1} + -z_{=1} & \geq -1 \\
\quad\rightarrow & & -w_{=2} + -x_{=2} & & & \geq -1 \\
\qquad\rightarrow & & -w_{=3} + -x_{=3} + & & -y_{=3} + -z_{=3} & \geq -1
\end{array}$$

$$\begin{array}{ll}
-v_{=1} & \geq 1 \\
v_{=1} & \geq 0
\end{array}$$

All-Different Propagator

$$
\begin{array}{l}
V \in \{1 \quad 4 \quad 5\} \\
W \in \{1 \quad 2 \quad 3\} \qquad w_{=1} + w_{=2} + w_{=3} \geq 1 \\
X \in \{2 \quad 3\} \qquad x_{=2} + x_{=3} \geq 1 \\
Y \in \{1 \quad 3\} \qquad y_{=1} + y_{=3} \geq 1 \\
Z \in \{1 \quad 3\} \qquad z_{=1} + z_{=3} \geq 1
\end{array}
$$

$$
\begin{array}{ll}
\rightarrow & -v_{=1} + -w_{=1} + -y_{=1} + -z_{=1} \geq -1 \\
\rightarrow & -w_{=2} + -x_{=2} \geq -1 \\
\rightarrow & -w_{=3} + -x_{=3} + -y_{=3} + -z_{=3} \geq -1
\end{array}
$$

$$
\begin{array}{l}
-v_{=1} \geq 1 \\
v_{=1} \geq 0
\end{array}
$$

$$0 \geq 1$$

Other Constraint Programming Reasoning

Efficient proof logging support for

- Table constraints
- Arrays
- Problem reformulations
- Backtracking during search
- Et cetera. . .

Other Constraint Programming Reasoning

Efficient proof logging support for

- Table constraints
- Arrays
- Problem reformulations
- Backtracking during search
- Et cetera. . .

Not at all trivial to implement
Lots of work left to get to full-fledged constraint programming solver
But so far everything has been possible to do [EGMN20, GMN22]

Actual Extension Rule: Redundance-Based Strengthening

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Adding redundant constraints should be OK

Actual Extension Rule: Redundance-Based Strengthening

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Adding redundant constraints should be OK

Redundance-based strengthening [BT19, GN21]

C is redundant with respect to F iff 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 $F \wedge C$

Actual Extension Rule: Redundance-Based Strengthening

C is redundant with respect to F if F and $F \wedge C$ are equisatisfiable

Adding redundant constraints should be OK

Redundance-based strengthening [BT19, GN21]

C is redundant with respect to F iff 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 $F \wedge C$

Witness ω should be specified, and implication be efficiently verifiable (which is the case, e.g., if all constraints in $(F \wedge C)\restriction_{\omega}$ are RUP)

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

Want to derive

$$2\overline{a} + x + y \geq 2 \qquad a + \overline{x} + \overline{y} \geq 1$$

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

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

Want to derive

$$2\overline{a} + x + y \geq 2 \qquad a + \overline{x} + \overline{y} \geq 1$$

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

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

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

Want to derive

$$2\overline{a} + x + y \geq 2 \qquad a + \overline{x} + \overline{y} \geq 1$$

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

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

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

Want to derive

$$2\overline{a} + x + y \geq 2 \qquad a + \overline{x} + \overline{y} \geq 1$$

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

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

2. $F \land (2\overline{a} + x + y \geq 2) \land \neg(a + \overline{x} + \overline{y} \geq 1) \models$
   $(F \land (2\overline{a} + x + y \geq 2) \land (a + \overline{x} + \overline{y} \geq 1)){\restriction_\omega}$

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

Want to derive

$$2\overline{a} + x + y \geq 2 \qquad a + \overline{x} + \overline{y} \geq 1$$

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

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

2. $F \wedge (2\overline{a} + x + y \geq 2) \wedge \neg(a + \overline{x} + \overline{y} \geq 1) \models$
   $(F \wedge (2\overline{a} + x + y \geq 2) \wedge (a + \overline{x} + \overline{y} \geq 1))\restriction_\omega$
   Choose $\omega = \{a \mapsto 1\}$ — F untouched; new constraint satisfied
   $\neg(a + \overline{x} + \overline{y} \geq 1)$ forces $x \mapsto 1$ and $y \mapsto 1$, hence $2\overline{a} + x + y \geq 2$ remains satisfied after forcing a to be true

Optimization Problems

Pseudo-Boolean optimization

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

Optimization Problems

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

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 $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \cdot \alpha(\ell_i)$

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

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

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

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

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

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

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

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

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

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

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

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 $f(((\alpha \circ \omega) \circ \omega) \circ \omega) < f((\alpha \circ \omega) \circ \omega)$
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. $\ldots$
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

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

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

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

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

The Challenge of Symmetries

Symmetries can be crucial for optimization problems [AW13, GSVW14]
Show up also in hard SAT benchmarks

The Challenge of Symmetries

Symmetries can be crucial for 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 can be crucial for 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]
- Adding rules for symmetric reasoning as in [TD20] breaks extension rule

Viewing Symmetry as an Optimization Problem

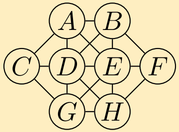
Deal with symmetries by switching focus to optimization

Invent objective function $\sum_{i=1}^{n} 2^i \cdot x_i$) corresponding to lexicographic order

Now dominance-based strengthening $=$ symmetry breaking!

Symmetry Elimination Example: Crystal Maze Puzzle

The Crystal Maze Puzzle



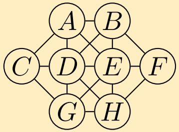
Place numbers 1 to 8 without repetition; adjacent circles cannot have consecutive numbers

Symmetry Elimination Example: Crystal Maze Puzzle

Human modellers might add:

- $A < G$ (mirror vertically)
- $A < B$ (mirror horizontally)
- $A \leq 4$ (value symmetry)

The Crystal Maze Puzzle



Place numbers 1 to 8 without repetition; adjacent circles cannot have consecutive numbers

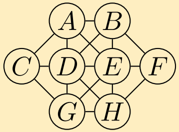
Symmetry Elimination Example: Crystal Maze Puzzle

Human modellers might add:

- $A < G$ (mirror vertically)
- $A < B$ (mirror horizontally)
- $A \leq 4$ (value symmetry)

Are these valid simultaneously?

The Crystal Maze Puzzle



Place numbers 1 to 8 without repetition; adjacent circles cannot have consecutive numbers

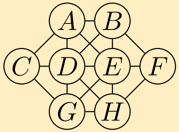
Symmetry Elimination Example: Crystal Maze Puzzle

Human modellers might add:

- $A < G$ (mirror vertically)
- $A < B$ (mirror horizontally)
- $A \leq 4$ (value symmetry)

Are these valid simultaneously?

The Crystal Maze Puzzle



Place numbers 1 to 8 without repetition; adjacent circles cannot have consecutive numbers

Can derive these constraints inside the proof rather than adding to input

- Witness ω: symmetry
- Order: Lexicographic $(A, B, \ldots, H)$
- No group theory required!

Directions for Future Research

Proof logging for combinatorial optimization

- Pseudo-Boolean optimization and MaxSAT solving (work in [GMNO22, VDB22, BBN^{+}23])
- General constraint programming (work in [EGMN20, GMN22])
- Mixed integer linear programming (work in [CGS17, EG21])

Directions for Future Research

Proof logging for combinatorial optimization

- Pseudo-Boolean optimization and MaxSAT solving (work in [GMNO22, VDB22, BBN$^+$23])
- General constraint programming (work in [EGMN20, GMN22])
- Mixed integer linear programming (work in [CGS17, EG21])

Logic and formal verification

- Formally verified proof checking
- Formally verified problem encoding/translation
- Higher-order logic for more efficient handling of repetitive proof fragments?
- SMT proof logging using stronger logics?

Directions for Future Research

Proof logging for combinatorial optimization

- Pseudo-Boolean optimization and MaxSAT solving (work in [GMNO22, VDB22, BBN$^+$23])
- General constraint programming (work in [EGMN20, GMN22])
- Mixed integer linear programming (work in [CGS17, EG21])

Logic and formal verification

- Formally verified proof checking
- Formally verified problem encoding/translation
- Higher-order logic for more efficient handling of repetitive proof fragments?
- SMT proof logging using stronger logics?

And more. . .

- Lots of challenging problems and interesting ideas!

Directions for Future Research

Proof logging for combinatorial optimization

- Pseudo-Boolean optimization and MaxSAT solving (work in [GMNO22, VDB22, BBN$^+$23])
- General constraint programming (work in [EGMN20, GMN22])
- Mixed integer linear programming (work in [CGS17, EG21])

Logic and formal verification

- Formally verified proof checking
- Formally verified problem encoding/translation
- Higher-order logic for more efficient handling of repetitive proof fragments?
- SMT proof logging using stronger logics?

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
- Requires powerful but simple proof systems — need for “computationally efficient logic”
- Cutting planes with strengthening rules operating on 0–1 linear inequalities seems to hit a sweet spot
- Potential for stronger logics and formal verification methods?

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
- Requires powerful but simple proof systems — need for “computationally efficient logic”
- Cutting planes with strengthening rules operating on 0–1 linear inequalities seems to hit a sweet spot
- Potential for stronger logics and formal verification methods?

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.

[BBN+23] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. Submitted manuscript, March 2023.

References II

[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)*, pages 3698–3707, February 2022.

[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.

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

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

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

References III

[BR07] Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming—A look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, February 2007.

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

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

[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.

References IV

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

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

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

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Revised Selected Papers from the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85. Springer, September 2009.

References V

[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.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*, pages 151–158, May 1971.

[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*.

[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.

References VI

[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.

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

References 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)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, August 2022.

[GMNO22] Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:25, August 2022.

References 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.

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

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

References IX

[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.

[GW08] Oded Goldreich and Avi Wigderson. Computational complexity. In Timothy Gowers, June Barrow-Green, and Imre Leader, editors, *The Princeton Companion to Mathematics*, chapter IV.20, pages 575–604. Princeton University Press, 2008.

[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 X

[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.

[HK17] Marijn J. H. Heule and Oliver Kullmann. The science of brute force. *Communications of the ACM*, 60(8):70–79, August 2017.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, March 2001. Preliminary version in *CCC '99*.

[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.

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973. In Russian. Available at `http://mi.mathnet.ru/ppi914`.

References XI

[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*.

[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.

References XII

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

[RM16] Olivier Roussel and Vasco M. Manquinho. Input/output format and solver requirements for the competitions of pseudo-Boolean solvers. Revision 2324. Available at `http://www.cril.univ-artois.fr/PB16/format.pdf`, January 2016.

[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.

[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.

[Tse68] Grigori Tseitin. On the complexity of derivation in propositional calculus. In A. O. Silenko, editor, *Structures in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Consultants Bureau, New York-London, 1968.

References XIII

[Van08] Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *10th International Symposium on Artificial Intelligence and Mathematics (ISAIM '08)*, 2008. Available at `http://isaim2008.unl.edu/index.php?page=proceedings`.

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

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