Auditable Constraint Programming

Ciaran McCreesh

With numerous co-conspirators, including Bart Bogaerts, Jan Elffers, Stephan Gocht, Ross McBride, Matthew McIlree, Jakob Nordström, Andy Oertel, Patrick Prosser, and James Trimble

University of Glasgow

Royal Academy of Engineering

Demotivation

My first experience of research: a summer internship reimplementing a clique algorithm from the literature.

My code produced the "wrong" answer on a few instances.

Demotivation

My first experience of research: a summer internship reimplementing a clique algorithm from the literature.

My code produced the “wrong” answer on a few instances.

I spent a month trying to find and fix it.

Demotivation

My first experience of research: a summer internship reimplementing a clique algorithm from the literature.

My code produced the “wrong” answer on a few instances.

I spent a month trying to find and fix it.

The published answers were wrong.

How Do We Know Our Solvers Are Correct?

I’ve wanted to write a CP solver for years.

How will I know it’s right? What if I ruin some poor student’s life by publishing wrong answers?

How Do We Know Our Solvers Are Correct?

I've wanted to write a CP solver for years.

How will I know it's right? What if I ruin some poor student's life by publishing wrong answers?

What if someone uses my solver for kidney exchange or workplace allocation or deciding adoptive parents?

The Slide That Keeps Getting Me Into Trouble

2021 MiniZinc challenge: for 1.28% of instances, wrong solutions were claimed.

- False claims of unsatisfiability.
- False claims of optimality.
- Infeasible solutions produced.
- Not limited to a single solver, problem, or constraint.
- Not even consistent—same solver on same hardware and same instance can give different results on different runs.

I don't want my solver to produce wrong answers!

The Slide That Keeps Getting Me Into Trouble

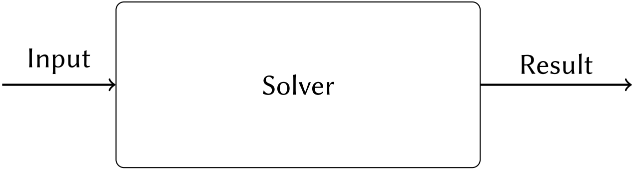
2021 MiniZinc challenge: for 1.28% of instances, wrong solutions were claimed.

- False claims of unsatisfiability.
- False claims of optimality.
- Infeasible solutions produced.
- Not limited to a single solver, problem, or constraint.
- Not even consistent—same solver on same hardware and same instance can give different results on different runs.

I don't want my solver to produce wrong answers!

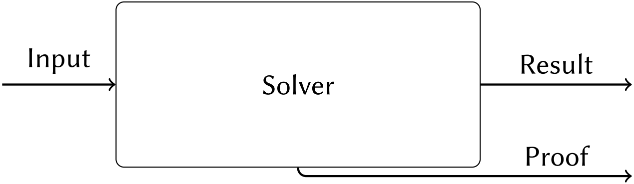
Or at least, when it's wrong, I want a guaranteed way of detecting it.

Proof Logging



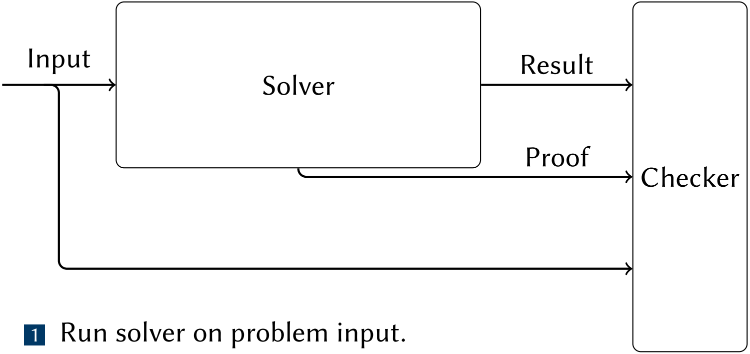
1. Run solver on problem input.

Proof Logging



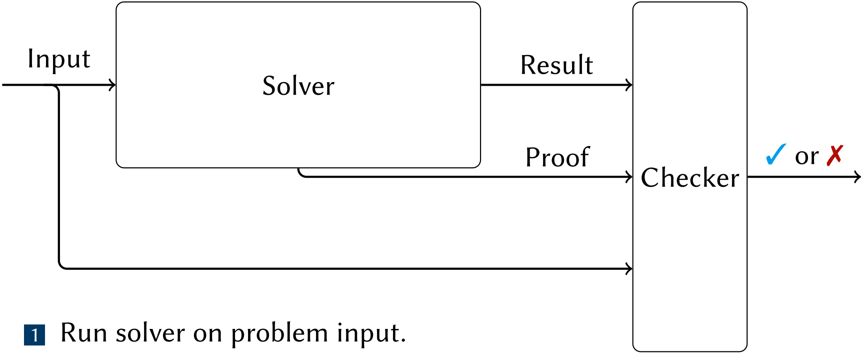
1. Run solver on problem input.
2. Get as output not only result but also proof.

Proof Logging



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

Proof Logging



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

What Is A Proof?

COUNTEREXAMPLE TO EULER'S CONJECTURE ON SUMS OF LIKE POWERS

BY L. J. LANDER AND T. R. PARKIN

Communicated by J. D. Swift, June 27, 1966

A direct search on the CDC 6600 yielded

$$27^5 + 84^5 + 110^5 + 133^5 = 144^5$$

as the smallest instance in which four fifth powers sum to a fifth power. This is a counterexample to a conjecture by Euler [1] that at least n nth powers are required to sum to an nth power, $n > 2$.

REFERENCE

1. L. E. Dickson, *History of the theory of numbers*, Vol. 2, Chelsea, New York, 1952, p. 648.

The SAT Problem

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

The SAT Problem

Given a CNF formula F, is it satisfiable?

For instance, what about:

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

Proofs for SAT

For satisfiable instances: just specify a satisfying assignment.

For unsatisfiability: a sequence of clauses (CNF constraints).

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

What Is Obvious? Unit Propagation

Unit Propagation

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

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

What Is Obvious? Unit Propagation

Unit Propagation

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

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

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

- $p \lor \overline{u}$ 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

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

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

Proof checker should know how to unit propagate until saturation.

Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

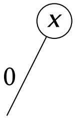
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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

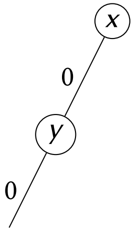


Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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

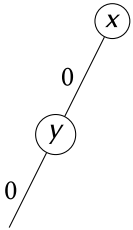


Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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



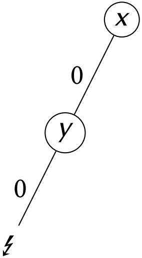
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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

1. $x \vee y$



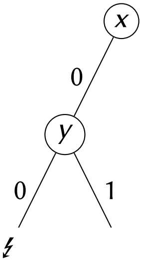
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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

1. $x \vee y$



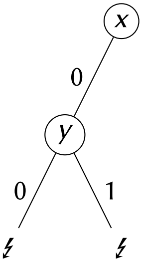
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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

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



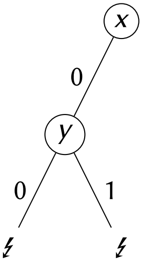
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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

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



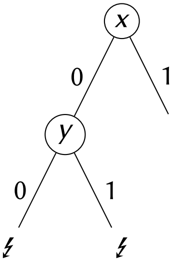
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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

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



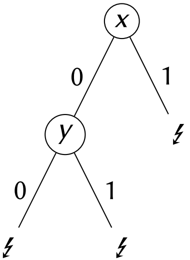
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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

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



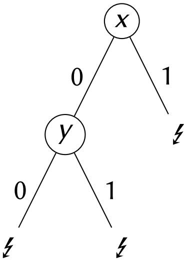
Davis-Putman-Logemann-Loveland (DPLL)

DPLL: Assign variables and propagate; backtrack when clause violated.

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

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

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



Reverse Unit Propagation (RUP)

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

Reverse Unit Propagation (RUP)

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

Reverse unit propagation (RUP) clause

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 condition easy to verify efficiently

Reverse Unit Propagation (RUP)

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

Reverse unit propagation (RUP) clause

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 condition easy to verify efficiently

Fact

Backtrack clauses from DPLL solver generate a RUP proof.

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

RUP Proofs and CDCL

Fact

All learned clauses generated by CDCL solver are RUP clauses.

So short proof of unsatisfiability for

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

is sequence of reverse unit propagation (RUP) clauses

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

Writing Proofs in the DRAT Format

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

Writing Proofs in the DRAT Format

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

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

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

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

$\overline{x}$

$\bot$

Writing Proofs in the DRAT Format

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

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

$x \vee y$

$x \vee \overline{y}$

x

$\overline{x}$

$\perp$

DPLL Proof in DRAT

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

Writing Proofs in the DRAT Format

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

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

$x \vee y$

$x \vee \overline{y}$

x

$\overline{x}$

$\perp$

DPLL Proof in DRAT

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

CDCL Proof in RUP

$u \vee x$

$\overline{x}$

$\perp$

Writing Proofs in the DRAT Format

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

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

$x \vee y$

$x \vee \overline{y}$

x

$\overline{x}$

$\perp$

DPLL Proof in DRAT

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

CDCL Proof in RUP

$u \vee x$

$\overline{x}$

$\perp$

CDCL Proof in DRAT

```
4 6 0
-6 0
0
```

Resolution Proofs

Fact

RUP proofs can be seen as shorthand for Resolution proofs.

Model axioms From the input

Resolution
$$\frac{x_1 \lor x_2 \lor \ldots \lor x_i \lor c \qquad \overline{c} \lor y_1 \lor y_2 \lor \ldots y_j}{x_1 \lor x_2 \lor \ldots \lor x_i \lor y_1 \lor y_2 \lor \ldots \lor y_j}$$

- To prove unsatisfiability: resolve until you reach the empty clause.

Reusing DRAT Isn't Feasible

- Stronger reasoning is hard in theory and in practice.
  - Resolution can't count efficiently.
- Closely tied to how MiniSAT works:
  - Proofs are (mostly) sequences of learned clauses.
  - Something special and strange happens to learned unit clauses.
- Preprocessing is possible (sometimes), but not easy.
  - We need to do full-on reformulation, though.
- Not clear how to do optimisation, enumeration, counting, …

Opinionated Requirements For This To Work

1. Efficiently work with what solvers actually do, not idealised algorithms.

Opinionated Requirements For This To Work

1. Efficiently work with what solvers actually do, not idealised algorithms.
2. No need for a new proof format for every new propagator or solver.
   - Constraint programming has 423 different global constraints, many of which have several different propagators.
   - Some propagators are buggy, and at least one has faulty theory behind it…

Opinionated Requirements For This To Work

1. Efficiently work with what solvers actually do, not idealised algorithms.
2. No need for a new proof format for every new propagator or solver.
   - Constraint programming has 423 different global constraints, many of which have several different propagators.
   - Some propagators are buggy, and at least one has faulty theory behind it…
3. Proof format must still be simple and well-founded.
   - Need to be able to trust the verifier.
   - Interactions between features can be subtle: even deletions aren't that easy to get right.

Unexpected and Remarkable Claim

- We can do everything we want with a proof format which is only slightly more sophisticated than DRAT.

Unexpected and Remarkable Claim

- We can do everything we want with a proof format which is only slightly more sophisticated than DRAT.
- Using proof logs during development leads to faster development than not doing proof logging.

Unexpected and Remarkable Claim

- We can do everything we want with a proof format which is only slightly more sophisticated than DRAT.
- Using proof logs during development leads to faster development than not doing proof logging.
- You should make your students and postdocs adopt this technology right now.

From CNF to Pseudo-Boolean

- A set of $\{0, 1\}$-valued variables x_i, 1 means true.
- Constraints are linear inequalities

$$\sum_i c_i x_i \geq C$$

- Write $\overline{x}_i$ to mean $1 - x_i$.
- Can rewrite CNF to pseudo-Boolean directly,

$$x_1 \vee \overline{x}_2 \vee x_3 \qquad \leftrightarrow \qquad x_1 + \overline{x}_2 + x_3 \geq 1$$

Cutting Planes Proofs

Model axioms From the input

Literal axioms

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

Addition

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

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

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

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

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

Extension Variables

Suppose we want new, fresh variable a encoding

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

Introduce constraints

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

Should be fine, so long as a hasn't been used before.

Interleaving RUP and Extended Cutting Planes

- Can define RUP similarly for pseudo-Boolean constraints.
  - It does the same thing on clauses.
  - Should probably be called “reverse integer bounds consistency”.
- Idea: use RUP for backtracking, and include explicit extended cutting planes steps to justify reasoning.

Proof Logs for Extended Cutting Planes

For satisfiable instances: just specify a satisfying assignment.

For unsatisfiability: a sequence of pseudo-Boolean constraints.

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

Enumeration and Optimisation Problems

Enumeration:

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

Enumeration and Optimisation Problems

Enumeration:

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

For optimisation:

- Define an objective $f = \sum_i w_i \ell_i$, $w_i \in \mathbb{Z}$, to minimise in the pseudo-Boolean model.
- To maximise, negate objective.
- Log a solution α, get a solution-improving constraint $\sum_i w_i \ell_i \leq -1 + \sum_i w_i \alpha(\ell_i)$.

The VeriPB System

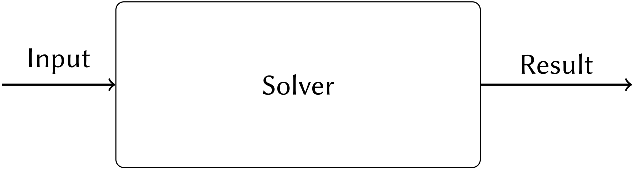
`https://gitlab.com/MIAOresearch/software/VeriPB`

- MIT licence, written in Python with parsing in C++.
- Useful features like tracing and proof debugging.

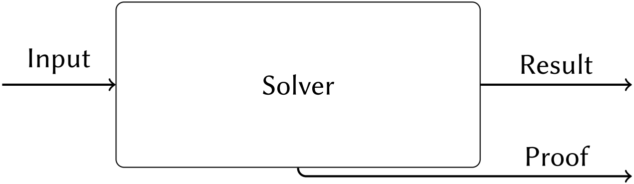
Making a Proof-Logging Solver

1. Output a pseudo-Boolean encoding of the problem.
2. Make the solver log its search tree.
   - Output a small header.
   - Output something on every backtrack.
   - Output something every time a solution is found.
   - Output a small footer.
3. Figure out how to log propagations.

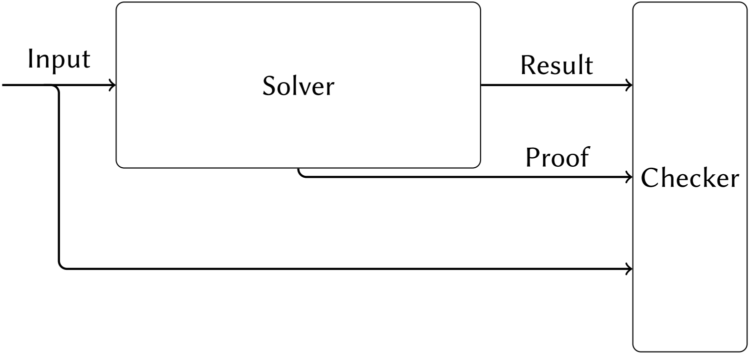
A Slightly Different Workflow



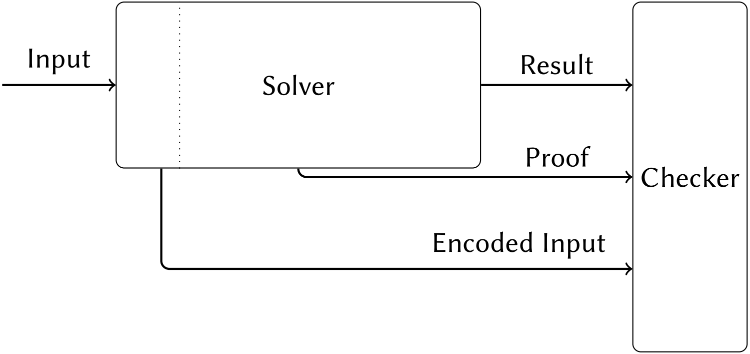
A Slightly Different Workflow



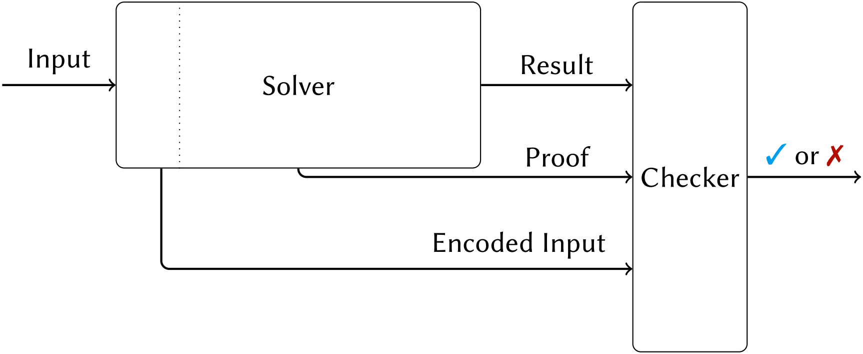
A Slightly Different Workflow



A Slightly Different Workflow



A Slightly Different Workflow



Extremely Critical Point That is Easily Misunderstood

- We're working with a normal constraint programming solver here.
- The Pseudo-Boolean encoding is only for the proof, and does not affect how the solver works.

Compiling CP Variables

Given $A \in \{-3 \ldots 9\}$:

$$a_{=-3} + a_{=-2} + a_{=-1} + a_{=0} + a_{=1} + a_{=2} + a_{=3} + a_{=4} + a_{=5} + a_{=6} + a_{=7} + a_{=8} + a_{=9} = 1$$

Compiling CP Variables

Given $A \in \{-3 \ldots 9\}$:

$$-32a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} + 16a_{\text{b4}} \geq -3 \text{ and}$$
$$32a_{\text{neg}} + -1a_{\text{b0}} + -2a_{\text{b1}} + -4a_{\text{b2}} + -8a_{\text{b3}} + -16a_{\text{b4}} \geq -9$$

Compiling CP Variables

Given $A \in \{-3 \dots 9\}$:

$$-32a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} + 16a_{\text{b4}} \geq -3 \text{ and}$$
$$32a_{\text{neg}} + -1a_{\text{b0}} + -2a_{\text{b1}} + -4a_{\text{b2}} + -8a_{\text{b3}} + -16a_{\text{b4}} \geq -9$$

Then where needed, define:

$$a_{\geq 4} \leftrightarrow -32a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} + 16a_{\text{b4}} \geq 4$$
$$a_{\geq 5} \leftrightarrow -32a_{\text{neg}} + 1a_{\text{b0}} + 2a_{\text{b1}} + 4a_{\text{b2}} + 8a_{\text{b3}} + 16a_{\text{b4}} \geq 5$$
$$a_{=4} \leftrightarrow a_{\geq 4} \wedge \overline{a}_{\geq 5}$$

We can do this in the pseudo-Boolean model, where needed, or lazily inside the proof using extension variables.

Introducing Useful Facts About Variables

When creating $x_{=i}$, also introduce

$$x_{\geq i} \rightarrow x_{\geq j} \quad \text{and} \quad x_{\geq h} \rightarrow x_{\geq i}$$

for the closest two values h and j that already have equality variables.

Introducing Useful Facts About Variables

When creating $x_{=i}$, also introduce

$$x_{\geq i} \rightarrow x_{\geq j} \quad \text{and} \quad x_{\geq h} \rightarrow x_{\geq i}$$

for the closest two values h and j that already have equality variables.

All-different is easier if we introduce

$$\sum_{i=\ell}^{u} x_{=i} \geq 1$$

which is also easy to do in a proof.

Compiling Constraints

- Also need to compile every constraint to pseudo-Boolean form.
- Doesn't need to be a propagating encoding.
- Can use additional variables.

Compiling Constraints

Given $2A + 3B + 4C \geq 42$, where $A, B, C \in \{-3 \ldots 9\}$,

$$\begin{aligned} &-64a_{\text{neg}} + 2a_{\text{b0}} + 4a_{\text{b1}} + 8a_{\text{b2}} + 16a_{\text{b3}} + 32a_{\text{b4}} \\ &+ -96b_{\text{neg}} + 3b_{\text{b0}} + 6b_{\text{b1}} + 12b_{\text{b2}} + 24b_{\text{b3}} + 48b_{\text{b4}} \\ &+ -128c_{\text{neg}} + 4c_{\text{b0}} + 8c_{\text{b1}} + 16c_{\text{b2}} + 32c_{\text{b3}} + 64c_{\text{b4}} \geq 42. \end{aligned}$$

Compiling Constraints

Given $(A, B, C) \in [(1,2,3), (1,3,4), (2,2,5)]$, define

$$
\begin{aligned}
3\bar{t}_0 + a_{=1} + b_{=2} + c_{=3} &\geq 3 \quad \text{i.e.} \quad & t_0 &\longrightarrow (a_{=1} \wedge b_{=2} \wedge c_{=3}) \\
3\bar{t}_1 + a_{=1} + b_{=4} + c_{=4} &\geq 3 \quad \text{i.e.} \quad & t_1 &\longrightarrow (a_{=1} \wedge b_{=4} \wedge c_{=4}) \\
3\bar{t}_2 + a_{=2} + b_{=2} + c_{=5} &\geq 3 \quad \text{i.e.} \quad & t_2 &\longrightarrow (a_{=2} \wedge b_{=2} \wedge c_{=5})
\end{aligned}
$$

using a tuple selector variable

$$t_0 + t_1 + t_2 = 1$$

Proof Logging Search Trees

Want to just output a reverse unit propagation step on every backtrack.

This works for forward-checking / DPLL, but not with strong propagators.

The key invariant: any propagation visible to the CP solver must be reflected either

- By "unit propagation" on the pseudo-Boolean model,
- Or by reverse unit propagation on the backtrack clause.

Proof Logging Inference: The Easy Cases

If it follows from bounds consistency on the pseudo-Boolean model, no further proof logging needed.

For example, a tuple in a table constraint becoming infeasible.

Intuition: some facts are so obvious they don't need stated.

Proof Logging Inference: Using RUP

Some facts are “obvious” once we tell the proof verifier they are true, but not otherwise.

For example, a variable losing a value due to a table constraint.

We log these propagations using RUP.

Intuition: like singleton arc consistency.

Proof Logging Inference: Explicit Justifications

Some facts aren't "obvious" but can be justified explicitly.

All-different: sum up the "variable takes at least one value" and "value is used at most once" constraints for a Hall set or Hall violator.

Integer linear inequalities: the slack algorithm gives an easy proof.

Justifying All-Different Failures

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

Justifying All-Different Failures

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

Justifying All-Different Failures

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

Justifying All-Different Failures

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

Justifying All-Different Failures

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

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

Justifying All-Different Failures

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

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

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

Justifying All-Different Failures

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

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

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

Justifying All-Different Failures

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

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

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

$$v_{=1} \geq 0$$

$$0 \geq 1$$

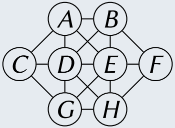
Proof Logging Reformulations

Some reformulations can be done inside the proof log:

- Turning not-equals from sums into binary constraints.
- 2D element constraints.
- Autotabulation.

Symmetry Elimination

The Crystal Maze Puzzle



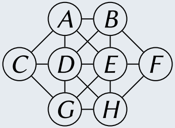
Place numbers 1 to 8 without repetition, adjacent circles cannot have consecutive numbers.

Symmetry Elimination

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.

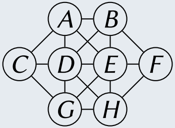
Symmetry Elimination

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.

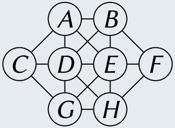
Symmetry Elimination

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.

We can introduce these constraints inside the proof, rather than as part of the pseudo-Boolean model. Based upon a *dominance* rule, no group theory required!

The Glasgow Constraint Solver

`https://github.com/ciaranm/glasgow-constraint-solver`

- MIT licence, written in fancy modern C++.
- A growing collection of global constraints:
  - Absolute value.
  - All-different.
  - Circuit (check and prevent).
  - Element.
  - Integer linear (in)equalities (with large domains, and GAC reformulation).
  - Minumum and Maximum.
  - Regular (and hence Stretch, Geost, DiffN).
  - Smart Table (and hence Lex, At Most One, Not All Equal).
- I couldn't think of a name.

A VeriPB Proof for a CP Problem

$A \in \{1 \dots 5\}$

$B \in \{1 \dots 2\}$

$C \in \{2 \dots 3\}$

$D \in \{2 \dots 3\}$

$\text{AllDiff}(A, B, C, D)$

$A + B + C \leq 9$

$\text{minimise } 2A + 3D$

```
Problem p;
auto va = p.create_integer_variable(1_i, 5_i, "a");
auto vb = p.create_integer_variable(1_i, 2_i, "b");
auto vc = p.create_integer_variable(2_i, 3_i, "c");
auto vd = p.create_integer_variable(2_i, 3_i, "d");
p.post(AllDifferent({va, vb, vc, vd}));
p.post(LinearLessEqual{Linear{{1_i, va}, {1_i, vb}, {1_i, vc}}, 9_i});

auto obj = p.create_integer_variable(0_i, 10000_i, "obj");
p.post(LinearEquality{Linear{{2_i, va}, {3_i, vd}, {-1_i, obj}}, 0_i});
p.minimise(obj);

cout << solve_with(p,
    SolveCallbacks{
        .solution = [&](const CurrentState & s) -> bool {
            cout << "a = " << s(va) << " b = " << s(vb) << " c = " << s(vc)
                << " d = " << s(vd) << " obj = " << s(obj) << endl;
            return true;
        },
    },
    ProofOptions{"tutorial.opb", "tutorial.veripb"});
```

A VeriPB Proof for a CP Problem

$A \in \{1 \dots 5\}$

$B \in \{1 \dots 2\}$

$C \in \{2 \dots 3\}$

$D \in \{2 \dots 3\}$

$\text{AllDiff}(A, B, C, D)$

$A + B + C \leq 9$

$\text{minimise } 2A + 3D$

```
$ ./build/tutorial_proof
a = 4 b = 1 c = 2 d = 3 obj = 17
a = 4 b = 1 c = 3 d = 2 obj = 14
propagators: 3
recursions: 5
failures: 1
propagations: 20 7 0
max depth:  2
solutions: 2
solve time: 0.001696s
```

A VeriPB Proof for a CP Problem

$A \in \{1 \dots 5\}$

$B \in \{1 \dots 2\}$

$C \in \{2 \dots 3\}$

$D \in \{2 \dots 3\}$

$\text{AllDiff}(A, B, C, D)$

$A + B + C \le 9$

$\text{minimise } 2A + 3D$

```
* #variable= 38 #constraint= 48
min: 1 xObj_b_0 2 xObj_b_1 4 xObj_b_2 8 xObj_b_3 16 xObj_b_4 32 xObj_b_5
    64 xObj_b_6 128 xObj_b_7 256 xObj_b_8 512 xObj_b_9 1024 xObj_b_10
    2048 xObj_b_11 4096 xObj_b_12 8192 xObj_b_13 ;
* variable xA_a 1 .. 5 bits encoding
1 xA_b_0 2 xA_b_1 4 xA_b_2 >= 1 ;
-1 xA_b_0 -2 xA_b_1 -4 xA_b_2 >= -5 ;
* variable xB_b 1 .. 2 bits encoding
1 xB_b_0 2 xB_b_1 >= 1 ;
-1 xB_b_0 -2 xB_b_1 >= -2 ;
* variable xC_c 2 .. 3 bits encoding
1 xC_b_0 2 xC_b_1 >= 2 ;
-1 xC_b_0 -2 xC_b_1 >= -3 ;
* variable xD_d 2 .. 3 bits encoding
1 xD_b_0 2 xD_b_1 >= 2 ;
-1 xD_b_0 -2 xD_b_1 >= -3 ;
* variable xObj_obj 0 .. 10000 bits encoding
1 xObj_b_0 2 xObj_b_1 4 xObj_b_2 8 xObj_b_3 16 xObj_b_4 32 xObj_b_5
    64 xObj_b_6 128 xObj_b_7 256 xObj_b_8 512 xObj_b_9 1024 xObj_b_10
    2048 xObj_b_11 4096 xObj_b_12 8192 xObj_b_13 >= 0 ;
-1 xObj_b_0 -2 xObj_b_1 -4 xObj_b_2 -8 xObj_b_3 -16 xObj_b_4 -32 xObj_b_5
    -64 xObj_b_6 -128 xObj_b_7 -256 xObj_b_8 -512 xObj_b_9 -1024 xObj_b_10
    -2048 xObj_b_11 -4096 xObj_b_12 -8192 xObj_b_13 >= -10000 ;
```

A VeriPB Proof for a CP Problem

$$A \in \{1 \ldots 5\}$$
$$B \in \{1 \ldots 2\}$$
$$C \in \{2 \ldots 3\}$$
$$D \in \{2 \ldots 3\}$$
$$\text{AllDiff}(A, B, C, D)$$
$$A + B + C \leq 9$$
$$\text{minimise } 2A + 3D$$

```
* constraint all different on A, B, C, D
-1 xA_eq_1 -1 xB_eq_1 >= -1 ;
-1 xA_eq_2 -1 xB_eq_2 -1 xC_eq_2 -1 xD_eq_2 >= -1 ;
-1 xA_eq_3 -1 xC_eq_3 -1 xD_eq_3 >= -1 ;

* need xA_ge_2
1 xA_b_0 2 xA_b_1 4 xA_b_2 2 ~xA_ge_2 >= 2 ;
-1 xA_b_0 -2 xA_b_1 -4 xA_b_2 6 xA_ge_2 >= -1 ;
* need lower bound xA_eq_1
1 ~xA_ge_2 1 ~xA_eq_1 >= 1 ;
-1 ~xA_ge_2 1 xA_eq_1 >= 0 ;
* need xB_ge_2
1 xB_b_0 2 xB_b_1 2 ~xB_ge_2 >= 2 ;
-1 xB_b_0 -2 xB_b_1 2 xB_ge_2 >= -1 ;
* need lower bound xB_eq_1
1 ~xB_ge_2 1 ~xB_eq_1 >= 1 ;
-1 ~xB_ge_2 1 xB_eq_1 >= 0 ;
* need xA_ge_3
1 xA_b_0 2 xA_b_1 4 xA_b_2 3 ~xA_ge_3 >= 3 ;
-1 xA_b_0 -2 xA_b_1 -4 xA_b_2 5 xA_ge_3 >= -2 ;
-1 xA_ge_3 1 xA_ge_2 >= 0 ;
* need xA_eq_2
1 xA_ge_2 1 ~xA_ge_3 2 ~xA_eq_2 >= 2 ;
-1 xA_ge_2 -1 ~xA_ge_3 1 xA_eq_2 >= -1 ;
* and so on...
```

A VeriPB Proof for a CP Problem

$A \in \{1 \ldots 5\}$

$B \in \{1 \ldots 2\}$

$C \in \{2 \ldots 3\}$

$D \in \{2 \ldots 3\}$

$\text{AllDiff}(A, B, C, D)$

$A + B + C \leq 9$

minimise $2A + 3D$

```
* constraint linear inequality 1*A 1*B 1*C <= 9
-1 xA_b_0 -2 xA_b_1 -4 xA_b_2 -1 xB_b_0 -2 xB_b_1 -1 xC_b_0 -2 xC_b_1 >= -9 ;

* constraint linear equality 2*A 3*D -1*Obj = 0
2 xA_b_0 4 xA_b_1 8 xA_b_2 3 xD_b_0 6 xD_b_1
    -1 xObj_b_0 -2 xObj_b_1 -4 xObj_b_2 -8 xObj_b_3 -16 xObj_b_4
    -32 xObj_b_5 -64 xObj_b_6 -128 xObj_b_7 -256 xObj_b_8
    -512 xObj_b_9 -1024 xObj_b_10 -2048 xObj_b_11 -4096 xObj_b_12
    -8192 xObj_b_13 >= 0 ;
-2 xA_b_0 -4 xA_b_1 -8 xA_b_2 -3 xD_b_0 -6 xD_b_1 1
    xObj_b_0 2 xObj_b_1 4 xObj_b_2 8 xObj_b_3 16 xObj_b_4
    32 xObj_b_5 64 xObj_b_6 128 xObj_b_7 256 xObj_b_8
    512 xObj_b_9 1024 xObj_b_10 2048 xObj_b_11 4096 xObj_b_12
    8192 xObj_b_13 >= 0 ;
```

A VeriPB Proof for a CP Problem

$A \in \{1 \ldots 5\}$

$B \in \{1 \ldots 2\}$

$C \in \{2 \ldots 3\}$

$D \in \{2 \ldots 3\}$

$\text{AllDiff}(A, B, C, D)$

$A + B + C \leq 9$

$\text{minimise } 2A + 3D$

```
pseudo-Boolean proof version 1.2
f 48 0

* all-different
u 1 xC_eq_2 1 xC_eq_3 >= 1 ;
u 1 xD_eq_2 1 xD_eq_3 >= 1 ;
p 49 50 + 35 + 45 +
u 1 ~xA_eq_1 >= 1 ;
u 1 ~xA_eq_2 >= 1 ;
u 1 ~xA_eq_3 >= 1 ;
u 1 ~xB_eq_2 >= 1 ;
```

A VeriPB Proof for a CP Problem

$A \in \{1 \ldots 5\}$

$B \in \{1 \ldots 2\}$

$C \in \{2 \ldots 3\}$

$D \in \{2 \ldots 3\}$

$\text{AllDiff}(A, B, C, D)$

$A + B + C \leq 9$

$\text{minimise } 2A + 3D$

```
* justifying integer linear inequality Obj >= 14
* need A >= 4
u 1 xA_b_0 2 xA_b_1 4 xA_b_2 >= 4 ;
p 48 56 2 * + 7 3 * + 1 d
* need xObj_ge_14
red 1 xObj_b_0 2 xObj_b_1 4 xObj_b_2 8 xObj_b_3 16 xObj_b_4 32 xObj_b_5
    64 xObj_b_6 128 xObj_b_7 256 xObj_b_8 512 xObj_b_9 1024 xObj_b_10
    2048 xObj_b_11 4096 xObj_b_12 8192 xObj_b_13 14 ~xObj_ge_14 >= 14 ;
    xObj_ge_14 0
red -1 xObj_b_0 -2 xObj_b_1 -4 xObj_b_2 -8 xObj_b_3 -16 xObj_b_4 -32 xObj_b_5
    -64 xObj_b_6 -128 xObj_b_7 -256 xObj_b_8 -512 xObj_b_9 -1024 xObj_b_10
    -2048 xObj_b_11 -4096 xObj_b_12 -8192 xObj_b_13 16370 xObj_ge_14 >= -13 ;
    xObj_ge_14 1
u 1 xObj_ge_14 >= 1 ;

* justifying integer linear inequality Obj < 20
p 47 2 2 * + 8 3 * + 1 d
* need xObj_ge_20 (omitted)
u 1 ~xObj_ge_20 >= 1 ;
```

A VeriPB Proof for a CP Problem

$A \in \{1 \ldots 5\}$

$B \in \{1 \ldots 2\}$

$C \in \{2 \ldots 3\}$

$D \in \{2 \ldots 3\}$

$\text{AllDiff}(A, B, C, D)$

$A + B + C \leq 9$

$\text{minimise } 2A + 3D$

```
* need xA_ge_5
red 1 xA_b_0 2 xA_b_1 4 xA_b_2 5 ~xA_ge_5 >= 5 ; xA_ge_5 0
red -1 xA_b_0 -2 xA_b_1 -4 xA_b_2 3 xA_ge_5 >= -4 ; xA_ge_5 1
u -1 xA_ge_5 1 xA_ge_4 >= 0 ;
* need xA_eq_4
red 1 xA_ge_4 1 ~xA_ge_5 2 ~xA_eq_4 >= 2 ; xA_eq_4 0
red -1 xA_ge_4 -1 ~xA_ge_5 1 xA_eq_4 >= -1 ; xA_eq_4 1
* guessing xA_eq_4, decision stack is [ ]

* justifying integer linear inequality Obj < 18
* need A < 5
u -1 xA_b_0 -2 xA_b_1 -4 xA_b_2 11 ~xA_eq_4 >= -4 ;
p 47 71 2 * + 8 3 * + 1 d
* need xObj_ge_18 (omitted)
u 1 ~xA_eq_4 1 ~xObj_ge_18 >= 1 ;
```

A VeriPB Proof for a CP Problem

$A \in \{1 \ldots 5\}$

$B \in \{1 \ldots 2\}$

$C \in \{2 \ldots 3\}$

$D \in \{2 \ldots 3\}$

$\text{AllDiff}(A, B, C, D)$

$A + B + C \leq 9$

$\text{minimise } 2A + 3D$

```
* guessing xC_eq_2, decision stack is [ xA_eq_4 ]
* all-different
u 1 ~xA_eq_4 1 ~xC_eq_2 1 ~xD_eq_2 >= 1 ;

* justifying integer linear inequality Obj >= 17
* need D >= 3
u 1 xD_b_0 2 xD_b_1 6 ~xA_eq_4 6 ~xC_eq_2 >= 3 ;
p 48 56 2 * + 79 3 * + 1 d
* need xObj_ge_17 (omitted)
u 1 ~xA_eq_4 1 ~xC_eq_2 1 xObj_ge_17 >= 1 ;

* solution
* need xObj_eq_17
red 1 xObj_ge_17 1 ~xObj_ge_18 2 ~xObj_eq_17 >= 2 ; xObj_eq_17 0
red -1 xObj_ge_17 -1 ~xObj_ge_18 1 xObj_eq_17 >= -1 ; xObj_eq_17 1
o xA_eq_4 xB_eq_1 xC_eq_2 xD_eq_3 xObj_eq_17 xObj_b_0 ~xObj_b_1
    ~xObj_b_2 ~xObj_b_3 xObj_b_4 ~xObj_b_5 ~xObj_b_6 ~xObj_b_7
    ~xObj_b_8 ~xObj_b_9 ~xObj_b_10 ~xObj_b_11 ~xObj_b_12 ~xObj_b_13
u 1 ~xObj_ge_17 >= 1 ;

* backtracking
u 1 ~xA_eq_4 1 ~xC_eq_2 >= 1 ;
```

A VeriPB Proof for a CP Problem

$A \in \{1 \ldots 5\}$

$B \in \{1 \ldots 2\}$

$C \in \{2 \ldots 3\}$

$D \in \{2 \ldots 3\}$

$\mathrm{AllDiff}(A, B, C, D)$

$A + B + C \leq 9$

$\text{minimise } 2A + 3D$

```
* then a bit more search happens (omitted), until...
* solution
o xA_eq_4 xB_eq_1 xC_eq_3 xD_eq_2 xObj_eq_14 ~xObj_b_0 xObj_b_1
    xObj_b_2 xObj_b_3 ~xObj_b_4 ~xObj_b_5 ~xObj_b_6 ~xObj_b_7
    ~xObj_b_8 ~xObj_b_9 ~xObj_b_10 ~xObj_b_11 ~xObj_b_12 ~xObj_b_13
u 1 ~xObj_ge_14 >= 1 ;

* backtracking
u 1 ~xA_eq_4 1 ~xC_eq_3 >= 1 ;

* backtracking
u 1 ~xA_eq_4 >= 1 ;

* need upper bound xA_eq_5
red 1 xA_ge_5 1 ~xA_eq_5 >= 1 ; xA_eq_5 0
red -1 xA_ge_5 1 xA_eq_5 >= 0 ; xA_eq_5 1
* guessing xA_eq_5, decision stack is [ ]

* backtracking
u 1 ~xA_eq_5 >= 1 ;

* backtracking
u >= 1 ;

* asserting contradiction
c -1
```

Propagator Bugs!

- Early versions of integer linear inequality propagator had bug with negative values and negative coefficients.
  - Integer division and modulus in C++ don't do what you expect for negative numbers.
  - I had forgotten this.
- Using "trust me" assertions, no wrong answers from many tests.
- Using proof logging: caught instantly.

What Do We Have?

- Don't know that the solver is correct.
- Do know that if a solver ever produces a wrong answer, it can be detected.
  - Even if due to a hardware or compiler error, or faulty maths.
  - We will need to get used to verification being (a constant factor) slower than solving.

What Do We Have?

- Don't know that the solver is correct.
- Do know that if a solver ever produces a wrong answer, it can be detected.
  - Even if due to a hardware or compiler error, or faulty maths.
  - We will need to get used to verification being (a constant factor) slower than solving.
  - Under the assumption that the pseudo-Boolean problem is correct.

What Do We Have?

- Don't know that the solver is correct.
- Do know that if a solver ever produces a wrong answer, it can be detected.
  - Even if due to a hardware or compiler error, or faulty maths.
  - We will need to get used to verification being (a constant factor) slower than solving.
  - Under the assumption that the pseudo-Boolean problem is correct.
- Also helps with testing and solver development: bugs are caught if incorrect reasoning is performed, rather than if a wrong answer is produced.

What Do We Have?

- Don't know that the solver is correct.
- Do know that if a solver ever produces a wrong answer, it can be detected.
  - Even if due to a hardware or compiler error, or faulty maths.
  - We will need to get used to verification being (a constant factor) slower than solving.
  - Under the assumption that the pseudo-Boolean problem is correct.
- Also helps with testing and solver development: bugs are caught if incorrect reasoning is performed, rather than if a wrong answer is produced.
- We get an auditable record of exactly what was actually solved.

What Else Can VeriPB Do?

- SAT with symmetries, cardinality, XOR reasoning, MaxSAT.
  - Uncovered several undetected bugs in state of the art solvers.
  - Can't do MaxSAT hitting set solvers yet, MIP isn't proof logged.
- Certified translations from pseudo-Boolean to CNF.
- Clique, subgraph isomorphism, maximum common (connected) induced subgraph.
- In progress: MIP preprocessing, dynamic programming, …

What Reasoning Can We Justify?

- With extension variables, as strong as Extended Frege.
- So according to theorists, we can simulate pretty much everything.

What Reasoning Can We Justify?

- With extension variables, as strong as Extended Frege.
- So according to theorists, we can simulate pretty much everything.
  - Up to a polynomial factor...

What Reasoning Can We Justify?

- With extension variables, as strong as Extended Frege.
- So according to theorists, we can simulate pretty much everything.
  - Up to a polynomial factor…
- Except dominance is apparently even stronger?

What Reasoning Can We Justify Efficiently?

- Quadratic overheads are unpleasant.
- Cutting planes is very good at justifying combinatorial arguments.
- It's not really clear why.

Verifying the Verifier

- How do we know the encoding is correct?
- How do we know the verifier is correct?
- How do we know the proof system is sound?

Proof Trimming

- Proofs can be really really really big.
- Often many steps end up being redundant for the final proof.
- Could we make a tool that turns a really really really big proof into a really big proof?

Going the Other Way

- Can we use proofs to understand solver behaviour?
  - Why solvers work so well when they shouldn't.
  - Why solvers perform so badly when they shouldn't.
- Explainability?

Where We're At

- Can verify *solutions* from state of the art combinatorial solving algorithms, in a unified proof system.
- Found many undetected bugs in widely used solvers.
  - Including in algorithms that have been "proved" correct.

Where We're At

- Can verify *solutions* from state of the art combinatorial solving algorithms, in a unified proof system.
- Found many undetected bugs in widely used solvers.
  - Including in algorithms that have been "proved" correct.
- Not being either proof logged or formally verified should be considered socially unacceptable.

Where We're At

- Can verify *solutions* from state of the art combinatorial solving algorithms, in a unified proof system.
- Found many undetected bugs in widely used solvers.
  - Including in algorithms that have been "proved" correct.
- Not being either proof logged or formally verified should be considered socially unacceptable.
- Perhaps studying proof logs can help explain why solvers work so well?

Getting Involved

- Glasgow has funding for PhD students starting this October.
- I will be hiring for a three year postdoc position as soon as the paperwork is finished.
- The Glasgow constraint solver:
  `https://github.com/ciaranm/glasgow-constraint-solver`
- Install VeriPB:
  `https://gitlab.com/MIAOresearch/software/VeriPB`
- Documentation:
  `https://satcompetition.github.io/2023/downloads/proposals/veripb.pdf`
- Tutorial:
  `https://www.youtube.com/watch?v=s_5BIi4I22w`

https://ciaranm.github.io/

ciaran.mccreesh@glasgow.ac.uk

University of Glasgow

Royal Academy of Engineering