

Formally Verified Certification of Constraint Programming Proofs

Maarten Flippo¹ Konstantin Sidorov¹ Tip ten Brink¹
Clément Pit-Claudel² Emir Demirović¹

July 21st, 2026 - CP 2026

¹Delft University of Technology

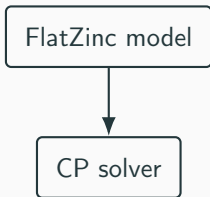
²EPFL

We formalise the semantics of constraint programming to **verify** claims of unsatisfiability.

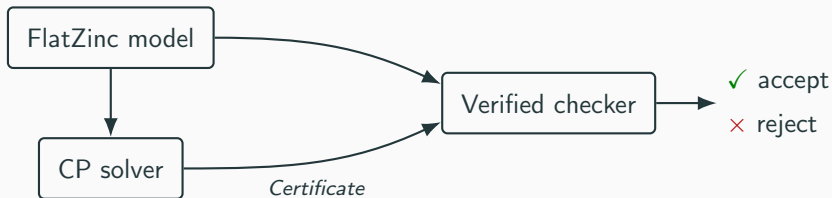
We formalise the semantics of constraint programming to **verify** claims of unsatisfiability.

FlatZinc model

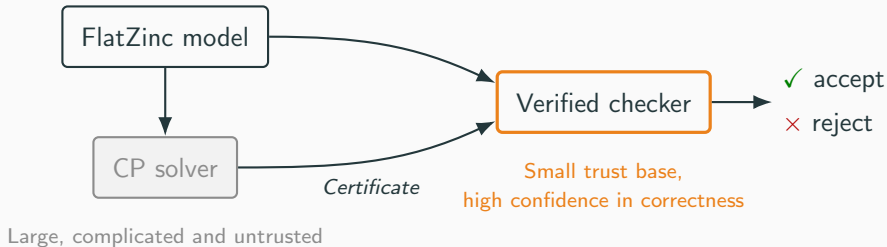
We formalise the semantics of constraint programming to **verify** claims of unsatisfiability.



We formalise the semantics of constraint programming to **verify** claims of unsatisfiability.



We formalise the semantics of constraint programming to **verify** claims of unsatisfiability.



Why is certification needed?

Why do we need a CP checker?

Why is certification needed?

- CP reasoning is diverse

Why do we need a CP checker?

Why is certification needed?

- CP reasoning is diverse
- Theoretically complex filtering rules

Why do we need a CP checker?

Why is certification needed?

- CP reasoning is diverse
- Theoretically complex filtering rules
- Non-trivial implementations

Why do we need a CP checker?

Why is certification needed?

- CP reasoning is diverse
- Theoretically complex filtering rules
- Non-trivial implementations

Why do we need a CP checker?

Bugs can, and do, occur.

Why is certification needed?

- CP reasoning is diverse
- Theoretically complex filtering rules
- Non-trivial implementations

Why do we need a CP checker?

- Formalizes filtering rules for CP

Bugs can, and do, occur.

Why is certification needed?

- CP reasoning is diverse
- Theoretically complex filtering rules
- Non-trivial implementations

Why do we need a CP checker?

- Formalizes filtering rules for CP
- Proves they are sound *in general*

Bugs can, and do, occur.

Why is certification needed?

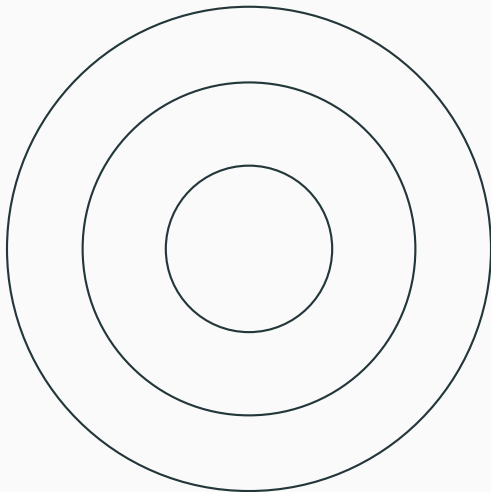
- CP reasoning is diverse
- Theoretically complex filtering rules
- Non-trivial implementations

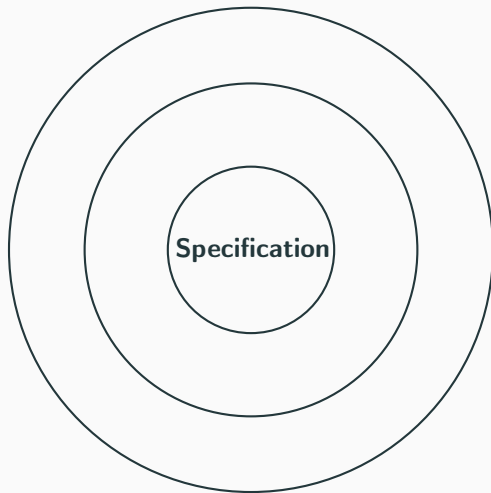
Why do we need a CP checker?

- Formalizes filtering rules for CP
- Proves they are sound *in general*
- Checks theory and implementation

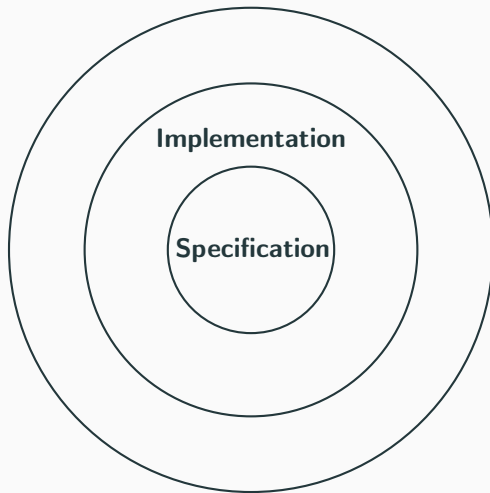
Bugs can, and do, occur.

FznDrcpCheck

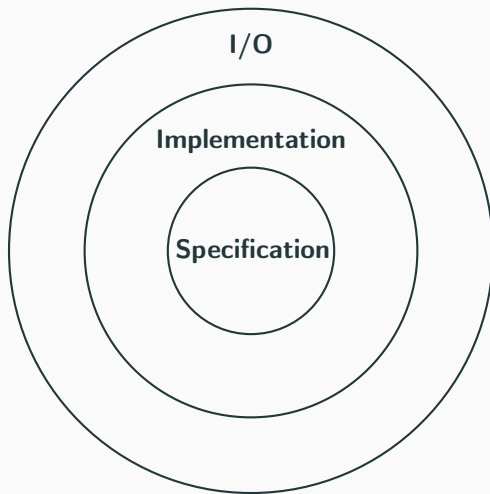




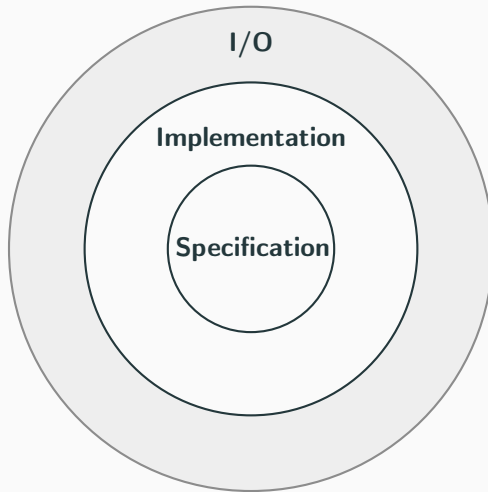
Project Overview

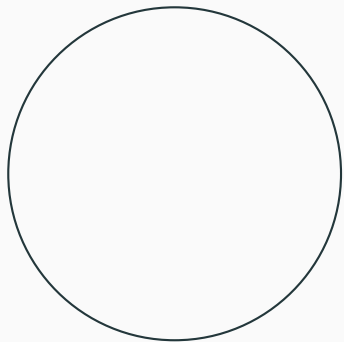


Project Overview



Project Overview







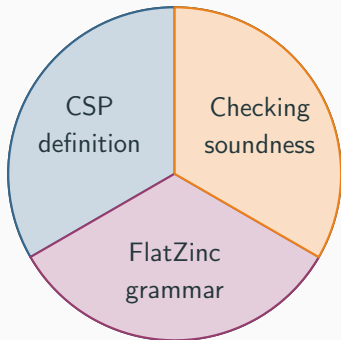
CSP definition:

- A variable: just an ID
- A domain: map from variables to sets of integers
- A constraint: a predicate



FlatZinc grammar:

- Base for generated parser

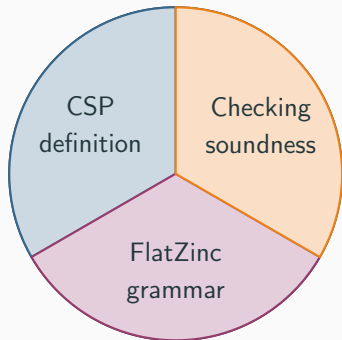


Checking soundness:

"If a checker accepts a conclusion given a CSP and some support, then the conclusion follows from the model."

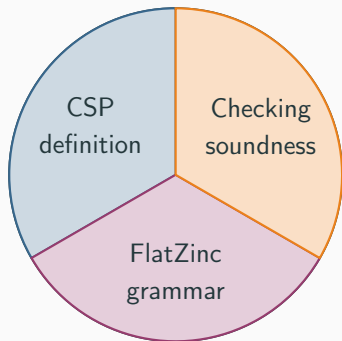
- The proof shape is irrelevant
- Bugs may cause wrongful rejection but never wrong accept!

Specification



Mirrors math in papers.
Not executable.

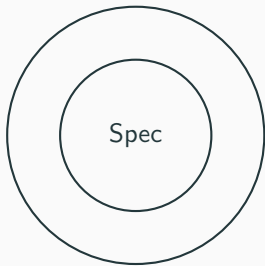
Specification



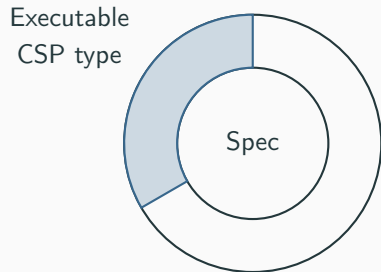
Mirrors math in papers.
Not executable.

The trusted base.

Formalisation of Filtering

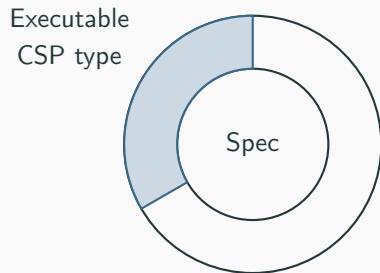


Formalisation of Filtering



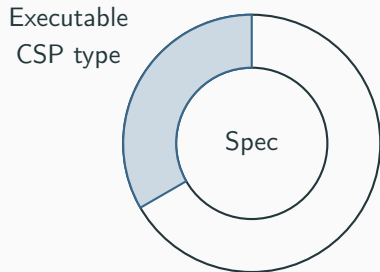
Executable CSP type:

Formalisation of Filtering



Executable CSP type:

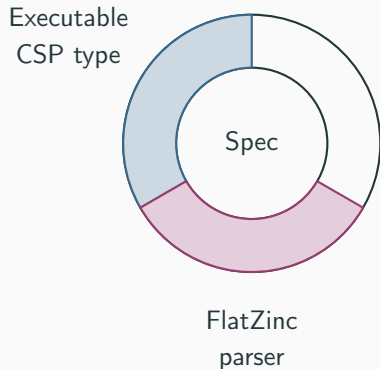
- Functions become maps



Executable CSP type:

- Functions become maps
- Dedicated domain representation

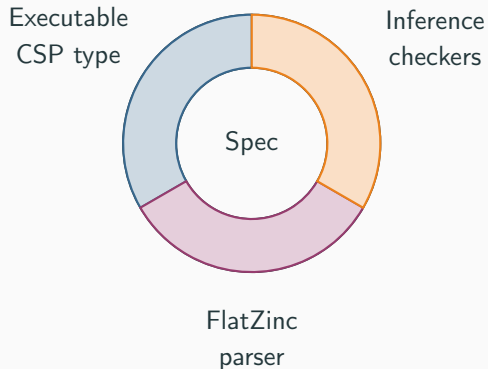
Formalisation of Filtering



FlatZinc parser:

- Generated based on the specification

Formalisation of Filtering



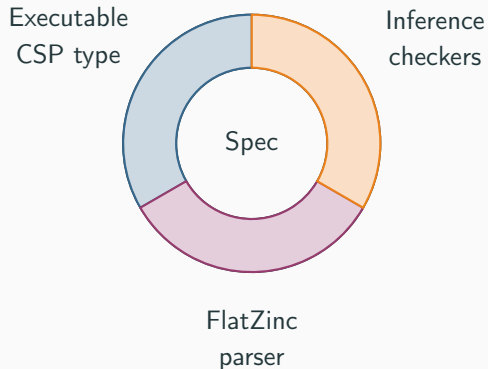
Inference Checkers:

$f(\text{model constraint}, \text{implied constraint})$

$\in$

$\{\text{sound}, \text{Don't know}\}$

Formalisation of Filtering



Inference Checkers:

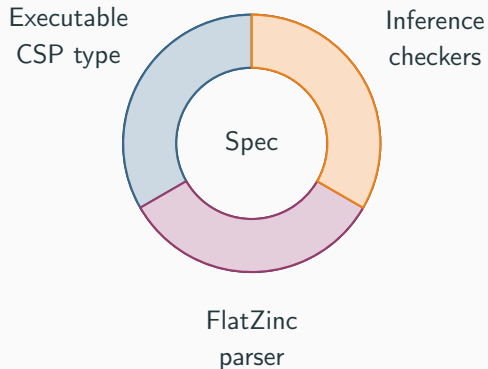
$f(\text{model constraint}, \text{implied constraint})$

$\in$

$\{\text{sound}, \text{Don't know}\}$

- Formalizes filtering

Formalisation of Filtering



Inference Checkers:

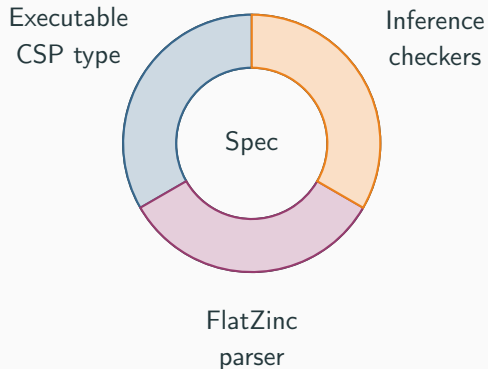
$f(\text{model constraint}, \text{implied constraint})$

$\in$

$\{\text{sound}, \text{Don't know}\}$

- Formalizes filtering
- Must be proven sound *in general*

Formalisation of Filtering



Inference Checkers:

$f(\text{model constraint}, \text{implied constraint})$

$\in$

$\{\text{sound}, \text{Don't know}\}$

- Formalizes filtering
- Must be proven sound *in general*
- Incomplete

Implemented inference checkers

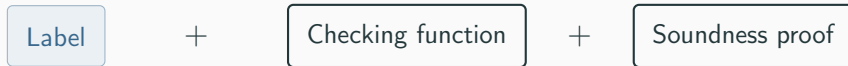
Linear less-or-equal	bound-consistent
All-different	domain-consistent
Cumulative	with time-table filtering
Unit propagation	of clauses

Implemented inference checkers

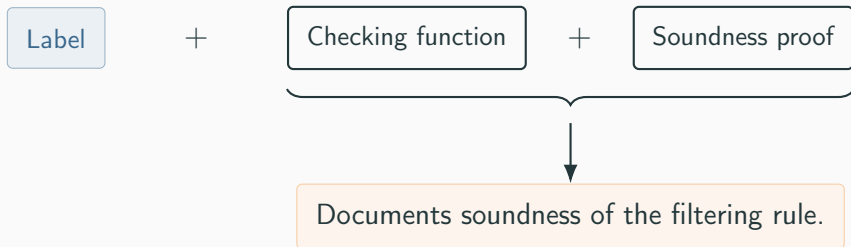
Linear less-or-equal	bound-consistent
All-different	domain-consistent
Cumulative	with time-table filtering
Unit propagation	of clauses

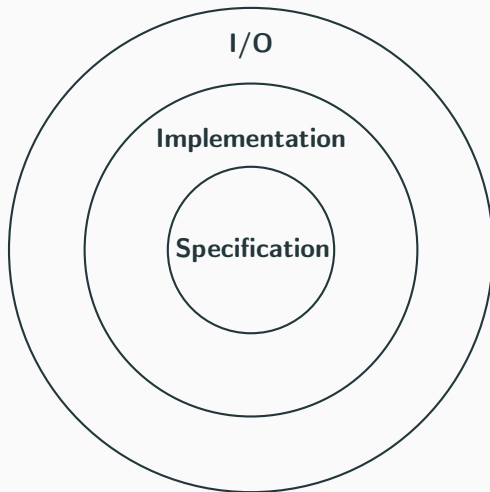
Easy to extend. We hope you will help us add more.

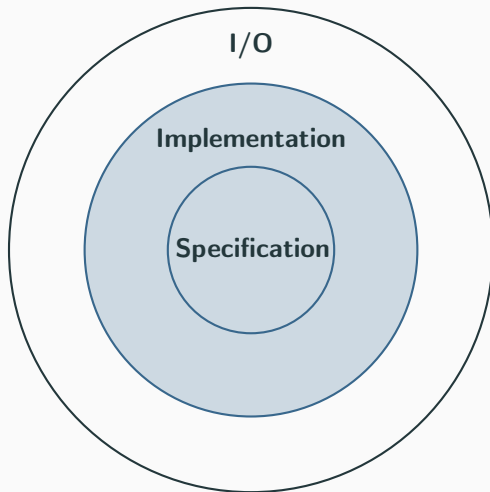
What is needed for an inference checker?



What is needed for an inference checker?







The proof system

Requirements:

Nice to have:

Requirements:

- Easy to produce

Nice to have:

Requirements:

- Easy to produce
- Easy to verify

Nice to have:

Requirements:

- Easy to produce
- Easy to verify

Nice to have:

- Easy to interpret

Is this problem feasible?

assign $x \in \{0, 1, 2\}$

$y, z \in \{0, 1\}$

s.t. $c_1: 5x + 4y \leq 6$

$c_2: x + y + z \geq 3$

Is this problem feasible?

assign $x \in \{0, 1, 2\}$

$y, z \in \{0, 1\}$

s.t. $c_1: 5x + 4y \leq 6$

$c_2: x + y + z \geq 3$

Proof



Derived constraints

$\vdots$

$c_n : \top \implies \perp$

Is this problem feasible?

assign $x \in \{0, 1, 2\}$

$y, z \in \{0, 1\}$

s.t. $c_1: 5x + 4y \leq 6$

$c_2: x + y + z \geq 3$

Proof



Derived constraints

$\vdots$

$c_n : \top \implies \perp$

Contradiction

assign $x \in \{0, 1, 2\}$

$y, z \in \{0, 1\}$

s.t. $c_1: 5x + 4y \leq 6$

$c_2: x + y + z \geq 3$

Proof

$c_3 : c_1 \models \langle y \geq 0 \rangle \implies \langle x \leq 1 \rangle$

Linear bounds

$\vdots$

$c_n : \top \implies \perp$

Inference rules

assign $x \in \{0, 1, 2\}$

$y, z \in \{0, 1\}$

s.t. $c_1: 5x + 4y \leq 6$

$c_2: x + y + z \geq 3$

Proof

$c_3 : c_1 \models \langle y \geq 0 \rangle \implies \langle x \leq 1 \rangle$

Linear bounds

$\vdots$

$c_n : \top \implies \perp$

Assume $\langle y \geq 0 \rangle$:

$5x \leq 6 - 4y \Rightarrow x \leq \frac{6}{5} \Rightarrow \langle x \leq 1 \rangle$

Inference rules

assign $x \in \{0, 1, 2\}$

$y, z \in \{0, 1\}$

s.t. $c_1: 5x + 4y \leq 6$

$c_2: x + y + z \geq 3$

Proof

$c_3: c_1 \models \langle y \geq 0 \rangle \implies \langle x \leq 1 \rangle$

Linear bounds

$c_4: c_1 \models \langle x \geq 1 \rangle \implies \langle y \leq 0 \rangle$

Linear bounds

$c_5: c_2 \models \langle x \leq 1 \rangle \wedge \langle z \leq 1 \rangle \implies \langle y \geq 1 \rangle$

Linear bounds

$\vdots$

$c_n: \top \implies \perp$

Combining facts: a deduction

$c_3: c_1 \models \langle y \geq 0 \rangle \rightarrow \langle x \leq 1 \rangle$

Linear bounds

$c_4: c_1 \models \langle x \geq 1 \rangle \rightarrow \langle y \leq 0 \rangle$

Linear bounds

$c_5: c_2 \models \langle x \leq 1 \rangle \wedge \langle z \leq 1 \rangle \rightarrow \langle y \geq 1 \rangle$

Linear bounds

$c_6: c_3 \wedge c_4 \wedge c_5 \models \langle x \geq 1 \rangle \wedge \langle y \geq 0 \rangle \wedge \langle z \leq 1 \rangle \rightarrow \perp$

Deduction

Combining facts: a deduction

$c_3: c_1 \models \langle y \geq 0 \rangle \rightarrow \langle x \leq 1 \rangle$

Linear bounds

$c_4: c_1 \models \langle x \geq 1 \rangle \rightarrow \langle y \leq 0 \rangle$

Linear bounds

$c_5: c_2 \models \langle x \leq 1 \rangle \wedge \langle z \leq 1 \rangle \rightarrow \langle y \geq 1 \rangle$

Linear bounds

$c_6: c_3 \wedge c_4 \wedge c_5 \models \langle x \geq 1 \rangle \wedge \langle y \geq 0 \rangle \wedge \langle z \leq 1 \rangle \rightarrow \perp$

Deduction

Assume $\langle x \geq 1 \rangle \wedge \langle y \geq 0 \rangle \wedge \langle z \leq 1 \rangle$:

$$\begin{array}{ccc} x & y & z \\ - & [1, \infty) & [0, \infty) & (-\infty, 1] \end{array}$$

Combining facts: a deduction

$c_3: c_1 \models \langle y \geq 0 \rangle \rightarrow \langle x \leq 1 \rangle$

Linear bounds

$c_4: c_1 \models \langle x \geq 1 \rangle \rightarrow \langle y \leq 0 \rangle$

Linear bounds

$c_5: c_2 \models \langle x \leq 1 \rangle \wedge \langle z \leq 1 \rangle \rightarrow \langle y \geq 1 \rangle$

Linear bounds

$c_6: c_3 \wedge c_4 \wedge c_5 \models \langle x \geq 1 \rangle \wedge \langle y \geq 0 \rangle \wedge \langle z \leq 1 \rangle \rightarrow \perp$ Deduction

Assume $\langle x \geq 1 \rangle \wedge \langle y \geq 0 \rangle \wedge \langle z \leq 1 \rangle$:

	x	y	z
–	$[1, \infty)$	$[0, \infty)$	$(-\infty, 1]$
c_3	$[1, 1]$	$[0, \infty)$	$(-\infty, 1]$

Combining facts: a deduction

$$c_3: c_1 \models \langle y \geq 0 \rangle \rightarrow \langle x \leq 1 \rangle$$

Linear bounds

$$c_4: c_1 \models \langle x \geq 1 \rangle \rightarrow \langle y \leq 0 \rangle$$

Linear bounds

$$c_5: c_2 \models \langle x \leq 1 \rangle \wedge \langle z \leq 1 \rangle \rightarrow \langle y \geq 1 \rangle$$

Linear bounds

$$c_6: c_3 \wedge c_4 \wedge c_5 \models \langle x \geq 1 \rangle \wedge \langle y \geq 0 \rangle \wedge \langle z \leq 1 \rangle \rightarrow \perp$$

Deduction



Assume $\langle x \geq 1 \rangle \wedge \langle y \geq 0 \rangle \wedge \langle z \leq 1 \rangle$:

	x	y	z
–	$[1, \infty)$	$[0, \infty)$	$(-\infty, 1]$
c_3	$[1, 1]$	$[0, \infty)$	$(-\infty, 1]$
c_4	$[1, 1]$	$[0, 0]$	$(-\infty, 1]$
c_5	$[1, 1]$	$[1, 0]$	$(-\infty, 1]$

Inconsistency!

- Built on CP concepts
- Mirrors solver operations
 - Propagations $\rightarrow$ inference steps
 - Learned nogoods $\rightarrow$ deductions

- Built on CP concepts
- Mirrors solver operations
 - Propagations $\rightarrow$ inference steps
 - Learned nogoods $\rightarrow$ deductions

Formalized in FznDrcpCheck

Experimental evaluation

Goal of the evaluation

Is verification practical?

Can we certify realistic instances?

Goal of the evaluation

Is verification practical?

✓ Checking is typically a fraction of proof production time.

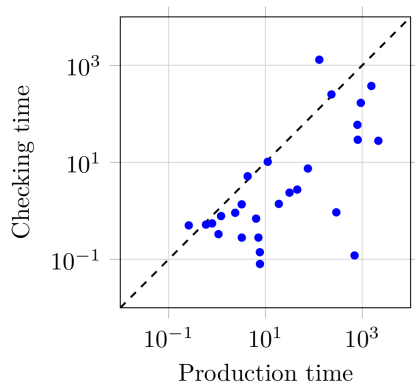
Can we certify realistic instances?

✓ Formally certified MiniZinc Challenge results.

Cost of certification

Setup

- MiniZinc Challenge* instances solved in 20 minutes
- Proof production = solve time + process time



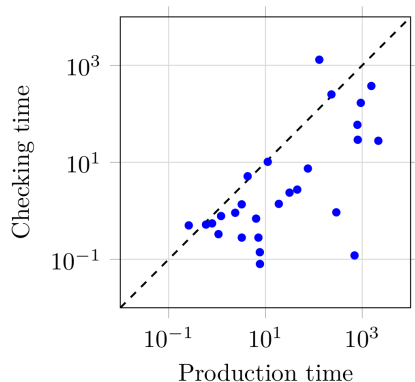
Cost of certification

Setup

- MiniZinc Challenge* instances solved in 20 minutes
- Proof production = solve time + process time

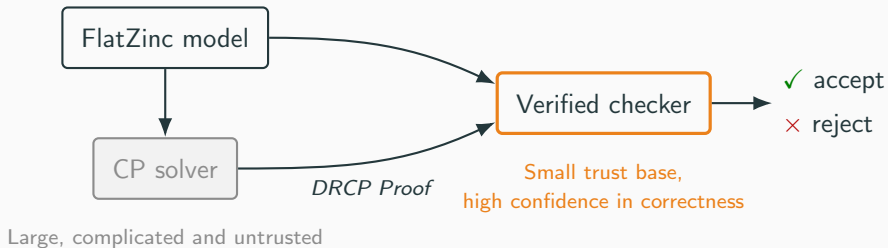
Conclusion

- FznDrcpCheck can verify practical instances.

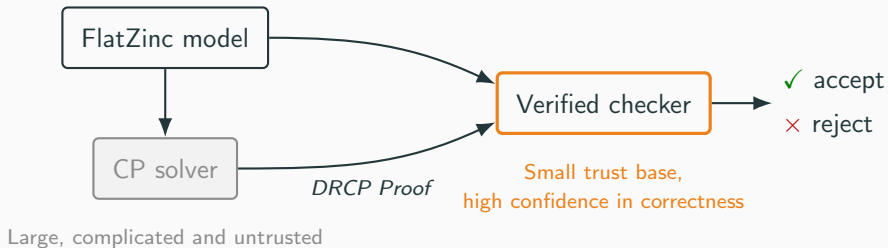


Closing

Conclusions



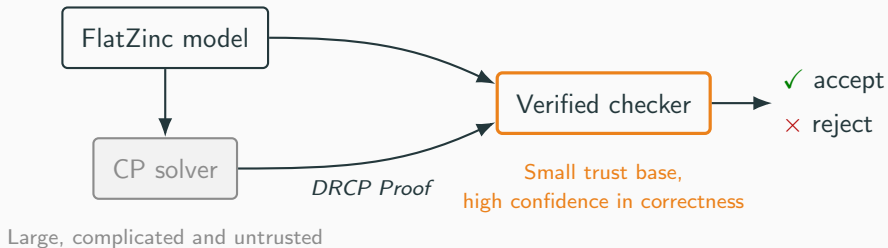
Conclusions



We presented

- **FznDrcpCheck**: Formalization of CP with checkers

Conclusions



We presented

- **FznDrcpCheck**: Formalization of CP with checkers
- **DRCP**: the proof system

Certified Preprocessing

Certified Preprocessing

Hypercube linear resolution¹

¹Flippo, M., Stuckey, P.J., Demirović, E. “Resolution Meets Cutting Planes: Introducing Hypercube Linear Resolution.” CPAIOR 2026.

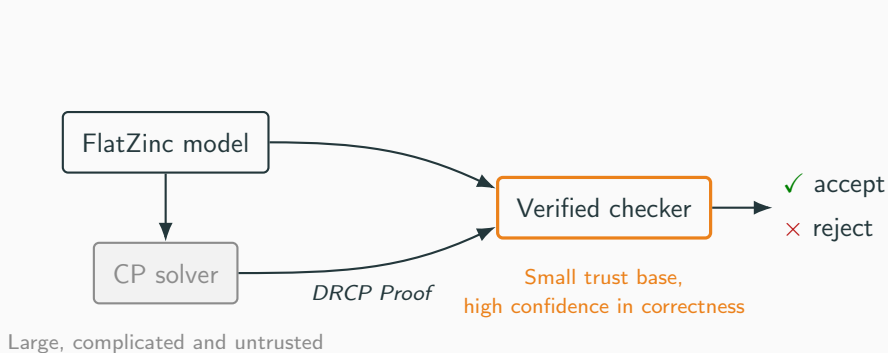
Certified Preprocessing

Hypercube linear resolution¹

More inference rules!

¹Flippo, M., Stuckey, P.J., Demirović, E. “Resolution Meets Cutting Planes: Introducing Hypercube Linear Resolution.” CPAIOR 2026.

Thank you



Checker



Paper



Appendix

	Translation to PB (VeriPB)	DRCP with FznDrcpCheck
<i>How</i>	CP $\rightarrow$ PB. Use VeriPB checker.	Formalise CP directly.
<i>Why</i>	Solver updates do not change checker.	CP-style proofs. Formalisation of CP.