# Wishes for the VeriPB proof format: an update

Daniel Le Berre

WHOOPS '25 - September 14, 2025

Université d'Artois and CNRS

# SAT4J is 21 years old ...

- SAT, MAXSAT, PBO solvers
- Main development between 2004 and 2011
- Specific work by Emmanuel Lonca (Multi-Objective Optimization in 2015) and Romain Wallon (PB proof systems in 2020)
- Contains PB solvers with either Resolution-based or Cutting-Planes-based proof systems

# Proof logging in Sat4j

- 2013: DRUP UNSAT proof (Daniel)
- 2021: VeriPB 1 UNSAT proof (Antony Blomme and Romain Wallon)
- 2024: iDRUP incremental proof and VeriPB 2 optimal and UNSAT first trial (Daniel)
- 2025: VeriPB 2 optimal and UNSAT second trial (Daniel with the help of Marc and Wietze)

# VeriPB 2 proof logging difficulty in Sat4j, simplifications

From toy/crafted benchmarks to competition benchmarks

- PB competition benchmarks contain "unit clauses"
- PB competition benchmarks contain PB constraints that can be simplified as cardinality constraints or clauses

Sat4j Cutting Planes VeriPB 2.0 certificates are incorrect in 2024 on those benchmarks

# VeriPB 2 proof logging difficulty in Sat4j, simplifications

From toy/crafted benchmarks to competition benchmarks

- PB competition benchmarks contain "unit clauses"
- PB competition benchmarks contain PB constraints that can be simplified as cardinality constraints or clauses

Sat4j Cutting Planes VeriPB 2.0 certificates are incorrect in 2024 on those benchmarks

Sat4j Resolution VeriPB 2.0 certificates are correct on those benchmarks

# Why is it a problem in Sat4j?

- "Unit clauses" are propagated directly when parsing the benchmark
- The simplification is performed to represent the constraint in the most appropriate way in the solver, again while parsing the benchmark
- This is true for both Sat4j Resolution and Sat4j Cutting Planes

There is no "event" in that case, especially because the simplifications can be done in many places

# Why is it a problem in Sat4j?

- "Unit clauses" are propagated directly when parsing the benchmark
- The simplification is performed to represent the constraint in the most appropriate way in the solver, again while parsing the benchmark
- This is true for both Sat4j Resolution and Sat4j Cutting Planes

There is no "event" in that case, especially because the simplifications can be done in many places

Claim at SLOPPY'24: *VeriPB 2.0 is friendly to Resolution proof system, unfriendly with Cutting Planes one!*

# How to fix this?

On Sat4j side:

- create new events for all simplifications occurring before the search?
- not so easy on 21 years old code (47k LOC)
- API fuzz testing can help (not available for PB yet in Sat4j)

On VeriPB side:

- Could VeriPB be more friendly with equivalent transformations?
- Could VeriPB focus on what we derive, not how we derive it?

# How did we fix this this year?

Suppose Sat4j reads the constraint $6x_1 + 2x_2 + x_3 + x_4 \geqslant 5$ and that $x_4 = 0$

We use labels to replace the original constraint by the simplified constraint:

`@x4 rup 1 ~x4 >= 1` $\overline{x}_4 \geqslant 1$ (1)

`@M pol M @x4 + s` $5x_1 + 2x_2 + x_3 \geqslant 5$ (2)

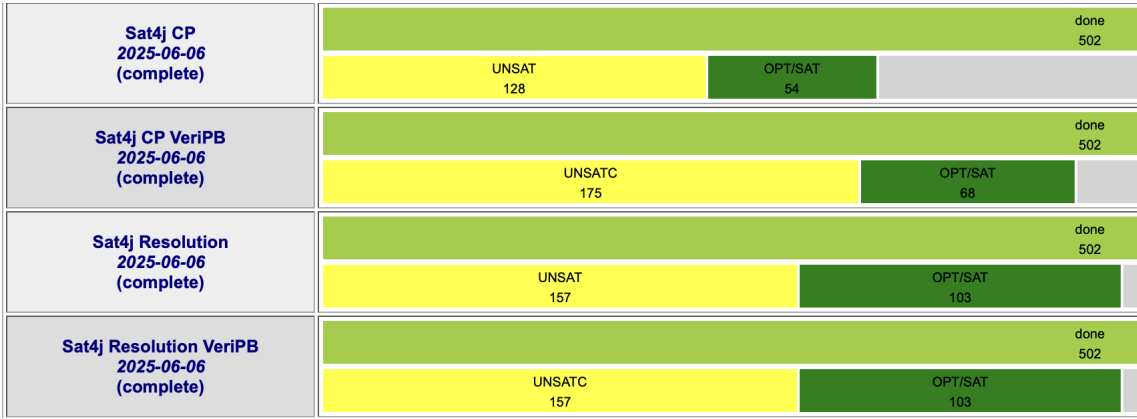
The propagation of $x_1$ is logged only if needed.

Suppose Sat4j reads later $3\overline{x}_1 + 2x_2 + 2x_3 + \overline{x}_4 + 2x_5 \geqslant 3$.

`@x1 rup 1 x1 >= 1` $x_1 \geqslant 1$ (3)

`@N pol N @x1 3 * + x4 w` $2x_2 + 2x_3 + 2x_5 \geqslant 2$ (4)

# VeriPB 2.0 in Sat4j 2025: the achievement



# VeriPB 2.0 in Sat4j 2025: the numbers

Codebase (before the changes):

- Lines of Code 48,339
- Classes 525
- Files 548

```
% git diff --stat main veripb2
149 files changed, 2961 insertions(+), 1299 deletions(-)
```

# VeriPB 2.0 in Sat4j 2025: the numbers

Codebase (before the changes):

- Lines of Code 48,339
- Classes 525
- Files 548

```
% git diff --stat main veripb2
149 files changed, 2961 insertions(+), 1299 deletions(-)
```

It only works on specific solvers (ResolutionPB24 and CuttingPlanesPB24)!

# VeriPB 2.0 in Sat4j 2025: the numbers

Codebase (before the changes):

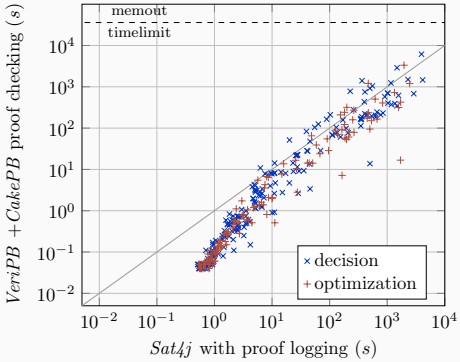
- Lines of Code 48,339
- Classes 525
- Files 548

```
% git diff --stat main veripb2
149 files changed, 2961 insertions(+), 1299 deletions(-)
```

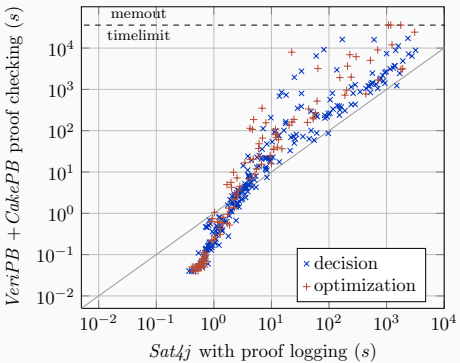
It only works on specific solvers (ResolutionPB24 and CuttingPlanesPB24)!

It introduced bugs in Sat4j (to compute hints for rup statements)!

# The issue with rup proofs




Cutting Planes




Resolution

# Can VeriPB proof logging help in fixing PB solvers?

- VeriPB checks that rules are correctly applied
- We know that it does not prevent deriving constraints with irrelevant literals
- Can VeriPB proofs help fixing this?

# Irrelevant Literals

Cutting planes rules may introduce **irrelevant literals**

$$\frac{3d + a + b + c \geqslant 3 \qquad 3\bar{d} + 2a + 2b \geqslant 3}{3a + 3b + c \geqslant 3}$$

# Irrelevant Literals

Cutting planes rules may introduce **irrelevant literals**

$$\frac{3d + a + b + c \geqslant 3 \qquad 3\bar{d} + 2a + 2b \geqslant 3}{3a + 3b + c \geqslant 3}$$

# Irrelevant Literals

Cutting planes rules may introduce **irrelevant literals**

$$\frac{3d + a + b + c \geqslant 3 \qquad 3\bar{d} + 2a + 2b \geqslant 3}{3a + 3b + c \geqslant 3}$$

# Irrelevant Literals

Cutting planes rules may introduce **irrelevant literals**

$$\frac{3d + a + b + c \geqslant 3 \qquad 3\bar{d} + 2a + 2b \geqslant 3}{3a + 3b + c \geqslant 3}$$

*A literal is said to be* **irrelevant** *in a PB constraint when its truth value does not impact the truth value of the constraint: irrelevant literals can thus be* **removed**

# Irrelevant Literals

Cutting planes rules may introduce **irrelevant literals**

$$\frac{3d + a + b + c \geqslant 3 \qquad 3\bar{d} + 2a + 2b \geqslant 3}{3a + 3b + \not{c} \geqslant 3}$$

*A literal is said to be* **irrelevant** *in a PB constraint when its truth value does not impact the truth value of the constraint: irrelevant literals can thus be* **removed**

# Production of Irrelevant Literals

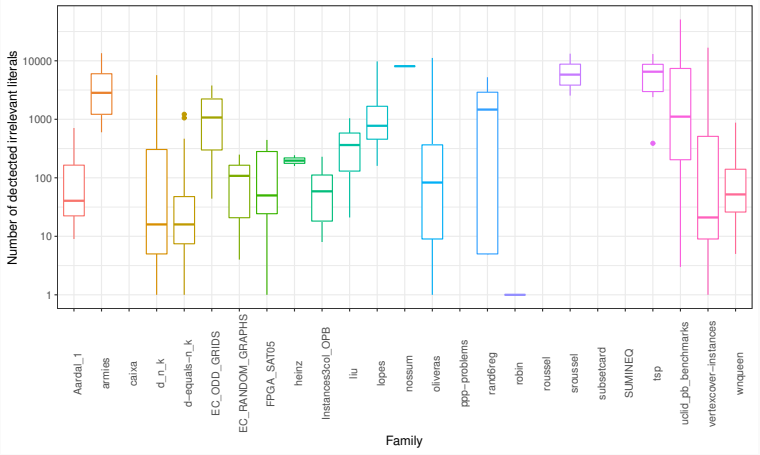



**Figure 1:** Statistics about the production of irrelevant literals in *Sat4j CurringPlanes* for each family of benchmarks (logarithmic scale)

# Artificially Relevant Literals

Irrelevant literals may become **artificially relevant**, in which case they may impact the strength of the derived constraints

$$\dfrac{\dfrac{3a + 3b + c \geqslant 3 \qquad 3\bar{a} + 3d + 2c \geqslant 3}{3b + 3c + 3d \geqslant 3}}{b + c + d \geqslant 1}$$

# Artificially Relevant Literals

Irrelevant literals may become **artificially relevant**, in which case they may impact the strength of the derived constraints

$$\dfrac{\dfrac{3a + 3b + c \geqslant 3 \qquad 3\bar{a} + 3d + 2c \geqslant 3}{3b + 3c + 3d \geqslant 3}}{b + c + d \geqslant 1}$$

# Artificially Relevant Literals

Irrelevant literals may become **artificially relevant**, in which case they may impact the strength of the derived constraints

$$\dfrac{\dfrac{3a + 3b + c \geqslant 3 \qquad 3\bar{a} + 3d + 2c \geqslant 3}{3b + 3c + 3d \geqslant 3}}{b + c + d \geqslant 1}$$

# Artificially Relevant Literals

Irrelevant literals may become **artificially relevant**, in which case they may impact the strength of the derived constraints

$$\dfrac{\dfrac{3a + 3b + \cancel{c} \geqslant 3 \qquad 3\bar{a} + 3d + 2c \geqslant 3}{3b + 3c + 3d \geqslant 3}}{b + c + d \geqslant 1}$$

# Artificially Relevant Literals

Irrelevant literals may become **artificially relevant**, in which case they may impact the strength of the derived constraints

$$\dfrac{\dfrac{3a + 3b + \cancel{c} \geqslant 3 \qquad 3\bar{a} + 3d + \cancel{2c} \geqslant 3}{3b + 3c + 3d \geqslant 3}}{b + c + d \geqslant 1}$$

# Artificially Relevant Literals

Irrelevant literals may become **artificially relevant**, in which case they may impact the strength of the derived constraints

$$\dfrac{\dfrac{3a + 3b + \cancel{c} \geqslant 3 \qquad 3\bar{a} + 3d + \cancel{2c} \geqslant 3}{3b + \cancel{3c} + 3d \geqslant 3}}{b + c + d \geqslant 1}$$

# Artificially Relevant Literals

Irrelevant literals may become **artificially relevant**, in which case they may impact the strength of the derived constraints

$$\dfrac{\dfrac{3a + 3b + \cancel{c} \geqslant 3 \qquad 3\bar{a} + 3d + \cancel{2c} \geqslant 3}{3b + \cancel{3c} + 3d \geqslant 3}}{b + \cancel{c} + d \geqslant 1}$$

# Artificially Relevant Literals

Irrelevant literals may become **artificially relevant**, in which case they may impact the strength of the derived constraints

$$\dfrac{\dfrac{3a + 3b + \cancel{c} \geqslant 3 \qquad 3\bar{a} + 3d + \cancel{2c} \geqslant 3}{3b + \cancel{3c} + 3d \geqslant 3}}{b + \cancel{c} + d \geqslant 1}$$

*Detecting irrelevant literals is* **NP-hard**

# One more thing ...

# Why we really care about SAT/UNSAT/OPTIMAL proofs!



## Autonomous Code Evolution Meets *NP-Completeness*

Cunxi Yu[1,2*], Rongjian Liang[3], Chia-Tung Ho[4], Haoxing Ren[3]

[1*] NVIDIA Research, College Park, 20740, MD, USA.
[2*] University of Maryland, College Park, 20742, MD, USA.
[3] NVIDIA Research, Austin, 78717, TX, USA.
[4] NVIDIA Research, Santa Clara, 95051, CA, USA.

*Corresponding author(s). E-mail(s): cunxiyu@umd.edu;

**Abstract**

Large language models (LLMs) have recently shown strong coding abilities, enabling not only static code generation but also iterative code self-evolving through agentic frameworks. Recently, AlphaEvolve [1] demonstrated that LLM-based coding agents can autonomously improve algorithms and surpass human experts, with scopes limited to isolated kernels spanning hundreds of lines of code. Inspired by AlphaEvolve, we present SATLUTION, the first framework to extend LLM-based code evolution to the full repository scale, encompassing hundreds of files and tens of thousands of lines of C/C++ code. Targeting Boolean Satisfiability (SAT), the canonical NP-complete problem and a cornerstone of both theory and applications. SATLUTION orchestrates LLM agents to directly evolve solver repositories under strict correctness guarantees and distributed runtime feedback, while simultaneously self-evolving its own evolution policies and rules. Starting from SAT Competition 2024 codebases and benchmark, SATLUTION evolved solvers that decisively outperformed the human-designed winners of the SAT Competition 2025, and also surpassed both 2024 and 2025 champions on the 2024 benchmarks.

**Keywords:** Large Language Models (LLMs), Boolean Satisfiability (SAT), Combinatorial Optimization, Coding Agent

**Tribute:** We are deeply grateful to the SAT solving community for nearly three decades of foundational work , which has produced modern SAT solvers capable of handling industrial-scale instances. In particular , the SAT Competition series , founded in 2002 , has provided a rigorous benchmarking arena that continues to motivate and accelerate solver innovation , setting high standards for performance and reproducibility in the field. We further acknowledge the landmark contributions of numerous solvers , such as *zChaff*