Pseudo-Boolean Blasting for the SMT Theory of Bit-Vectors

Pedro Saccomani, Haniel Barbosa

Universidade Federal de Minas Gerais

July 19, 2026

Outline

1. Introduction
2. Reasoning with Arithmetic
3. Pseudo-Boolean blasting
4. Proof production
5. Contributions and Challenges

Outline

- Work in Progress, in a novel decision procedure for the SMT-theory of bit-vectors, with two main components: Encoding of operators into pseudo-boolean constraints and proof logging.
- Work in collaboration with Jakob Nordström and Wietze Koops and current and past students from our research group:
  - **Alan Prado**
  - Bernardo Borges
  - Vinícius dos Santos Daher

Introduction: Motivation

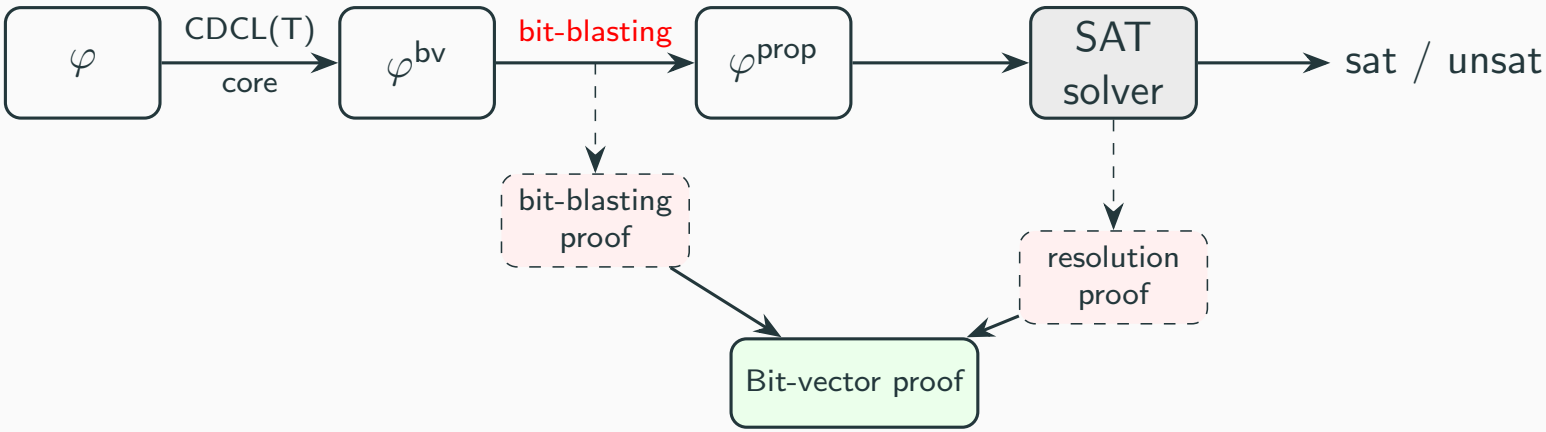
Bit-vectors can capture the semantics of machine integers: N-bit integers can be modeled as bit-vectors of width n:

$$a = a_{m-1}a_{m-2}\cdots a_1 a_0 \quad \longleftrightarrow \quad a' = \sum_{i=0}^{m-1} a_i \cdot 2^i$$

Where arithmetic operations over bit-vectors can be viewed as arithmetic modulo 2^m over the integers.

Introduction: Deciding bit-vector constraints

How does an SMT solver decide a bit-vector formula?



- φ: input formula in the bit-vector theory.
- φ^{bv}: the conjunction of bit-vector *literals* whose satisfiability the theory solver must decide.
- φ^{prop}: an equisatisfiable propositional (CNF) formula.

Introduction: Bit-blasting

The step $\varphi^{\text{bv}} \rightarrow \varphi^{\text{prop}}$ is called bit-blasting:

- each k-width term becomes k Boolean variables;
- each predicate is encoded as an equisatisfiable set of clauses;
- each operator has its semantics given as the output of a boolean circuit.

SMT-LIB's fixed-size bit-vector theory has **many** operators to encode:

```
Bitwise:  bvand, bvor, bvxor, bvnot, bvnand, bvnor, bvxnor
Arithmetic:  bvadd, bvsub, bvneg, bvmul, bvudiv, bvurem, bvsdiv, bvsrem,
bvsmod
Shifts:  bvshl, bvlshr, bvashr
Structural:  concat, extract, zero_extend, sign_extend, repeat,
rotate_left, rotate_right
Relational:  bvult, bvule, bvugt, bvuge, bvslt, bvsle, bvsgt, bvsge, =,
distinct
```

Introduction: Arithmetic as a bottleneck

$\mathcal{O}(n^2)$ clauses

- Boolean circuit for multiplication is quadratic in the width: CNF encoding of multiplication is **quadratic in** n.
- Worse: **resolution refutations of multiplication identities can be exponential**.

A. Biere, *Weaknesses of CDCL Solvers*. A. Niemetz, *Challenges in State-of-the-Art Bit-Precise Reasoning*.

Approach 1: word-level computer algebra

Interpret bit-vectors through their word-level specification:

- Encode as *polynomials* over the bits and employ techniques based on **Gröbner-basis** engines.
  [**Ritirc et al.**, **2017**, **Kaufmann et al.**, **2019**].
- Reasoning can be expressed by the Polynomial Calculus (PC) proof system.

Strength

Excels at *arithmetic*: verify multiplication circuits where SAT/resolution blows up.

Weakness

Does not handle *general*, *bit-level* reasoning well. Bit-vector formula may contain mixed constraints.[**Liew et al.**, **2020**].

Approach 2: pseudo-Boolean reasoning

Pseudo-Boolean (PB) solvers reason with cutting planes (CP).

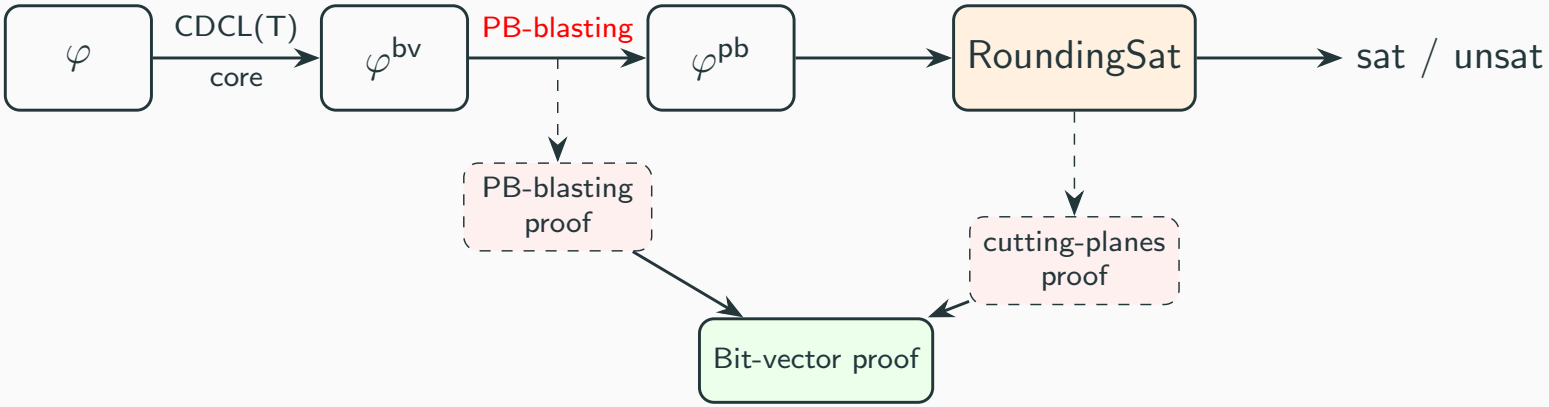
- CP is able to *simulate* resolution, and is **exponentially stronger** on some of the arithmetic structure of bit-vectors[**Liew et al.**, **2020**].
- Avoids PC's difficulty with bit-level reasoning using more natural encodings.

Resolution	**Cutting Planes**	**Polynomial Calculus**
SAT / CDCL	PB solvers	
bit-level ✓ arithmetic ✗	bit-level ✓ arithmetic ✓	bit-level ✗ arithmetic ✓

Our goal: Develop a **proof-producing** decision procedure that blasts bit-vectors to PB constraints and discharges them to an external PB solver.

Pseudo-Boolean blasting

We replace the step $\varphi^{\text{bv}} \to \varphi^{\text{prop}}$ by $\varphi^{\text{bv}} \to \varphi^{\text{pb}}$ with *pseudo-Boolean constraints*: 0/1 variables and linear inequalities of the form: $\sum_i a_i \ell_i \geq A$.



Two proof formats: Alethe & CPC

cvc5 logs proofs in two formats:

- **Alethe** [**Schurr et al.**, **2021**]: A generic, SMT-LIB-style proof format, checked by **Carcara** [**Andreotti et al.**, **2023**].
- **CPC** (Cooperating Proof Calculus): Checked by **Ethos** [**Reynolds et al.**, **2024**].

Both can give *fine-grained*, independently checkable certificates.

Goal

Produce a checkable proof for the **whole** PB-blasting decision procedure: The translation *and* cutting-planes reasoning.

Lifting cutting-planes proofs

The proof splits into two parts.

(1) Cutting-planes "core". VERIPB can be mapped cutting-planes rules, represented as an **inference rule** (considering the reverse polish notation rule):

$$\frac{\sum_i a_i\,\ell_i \geq A,\quad \sum_i b_i\,\ell_i \geq B \quad\mid\quad -}{\sum_i (a_i + b_i)\,\ell_i \geq A + B}\ \texttt{cp\_addition}\ (\text{VERIPB}\ \texttt{+})$$

$$\frac{\sum_i a_i\,\ell_i \geq A \quad\mid\quad c \in \mathbb{N}^+}{\sum_i c\,a_i\,\ell_i \geq c\,A}\ \texttt{cp\_multiplication}\ (\text{VERIPB}\ c\ \texttt{*})$$

$$\frac{\sum_i a_i\,\ell_i \geq A \quad\mid\quad c \in \mathbb{N}^+}{\sum_i \lceil a_i/c \rceil\,\ell_i \geq \lceil A/c \rceil}\ \texttt{cp\_division}\ (\text{VERIPB}\ c\ \texttt{d})$$

$$\frac{\sum_i a_i\,\ell_i \geq A \quad\mid\quad -}{\sum_i \min(a_i, A)\,\ell_i \geq A}\ \texttt{cp\_saturation}\ (\text{VERIPB}\ \texttt{s})$$

Lifting cutting-planes proofs

(2) Translation justification. There are two ways to capture this part:

- A single proof rule: The proof checker will infer **and** replay the translation (or prove it externally).
- A separate proof rule for **each** blasting step `pbblast_bvmult`, `pbblast_bvand`, `pbblast_bvult`, ...

Translation justification: proof structure

Each PB-blasting step is a defining equality $t \approx \mathrm{pbblast}(t)$, turning a term into a `pbbterm` of 0/1 integers $\big($writing x_i for $(\mathsf{int\_of}_i\, x)\big)$.

Alethe: one rule (without premises and arguments); e.g. `bvand`:

$$\frac{- \quad | \quad -}{(\mathsf{bvand}\ x\ y) \approx \mathsf{pbbterm}\big(r_0,\ \ldots,\ r_{n-1}\big)}\ \texttt{pbblast\_bvand}$$

each bit r_i encodes $x_i \wedge y_i$: $\quad x_i \geq r_i, \quad y_i \geq r_i, \quad r_i + 1 \geq x_i + y_i$

CPC: One *generic* rule with a program that replays the blast

$$\frac{- \quad | \quad \texttt{\$pb\_\_bitblast\_step}\,(\mathsf{bvand}\ x\ y)\ = b}{(\mathsf{bvand}\ x\ y) \approx b}\ \texttt{pb\_bitblast\_step}$$

Example: blasting `bvmul`

Input (2-bit, **unsatisfiable**: commutativity):

$$(\texttt{bvmul}\ x\ y)\ \neq\ (\texttt{bvmul}\ y\ x)$$

Encoding of $z = x \cdot y$ (Liew et al. tableau [**Liew et al.**, **2020**]):

partial products $t_{ij} \Leftrightarrow x_i \wedge y_j$:

$$x_i + y_j - 2\,t_{ij} \geq 0 \qquad -x_i - y_j + t_{ij} \geq -1$$

weight-balance equation:

$$\sum_{i,j} 2^{i+j}\, t_{ij}\ -\ \sum_{i} 2^{i}\, r_i\ =\ 0$$

Disequality ($r_P \neq r_Q$): with $z = \mathrm{XOR}(r_P, r_Q)$, assert $\sum_i z_i \geq 1$.

Example: the expected Alethe proof

For $(\texttt{bvmul}\ x\ y) \neq (\texttt{bvmul}\ y\ x)$, the expected proof has the shape $\left(P = \sum_i 2^i p_i,\ Q = \sum_i 2^i q_i\right)$:

```
(assume a0 (not (= (bvmul x y) (bvmul y x))))
(step t1 (cl (= (bvmul x y) (pbbterm p0 p1))) :rule pbblast_bvmult)
(step t2 (cl (= (bvmul y x) (pbbterm q0 q1))) :rule pbblast_bvmult)
(step t3 (cl (= (= (bvmul x y) (bvmul y x)) (= (- P Q) 0))) :rule
pbblast_bveq)
```

⋮ introduce tableau + weight-balance constraints

```
(step t20 ...:rule cp_multiplication ...)
```

⋮ ≈70 cutting-planes steps (add / mult / divide / saturate)

```
(step tN (cl (>= 0 1)) :rule cp_addition :premises (...))
(step tM (cl false) :rule la_generic :premises (tN))
```

Translation (`pbblast_*`) + a large cutting-planes core.

Contributions

- Translation of bit-vector formulas into pseudo-boolean constraints
  - Extended the encodings described in [**Liew et al.**, **2020**] to cover a **substantial** number of operators of the SMT-LIB of bit-vectors;
  - Implemented this translation in the **cvc5** SMT solver.
- Proof Production and Checking
  - Proposal of a specification for the alethe rules on our calculus;
  - Checkers for the core cutting planes rules implemented on carcara.
  - **cvc5** can emit VERIPB proofs in a coarse-grained single rule.

Challenges (I)

- Translation of bit-vector formulas into pseudo-boolean constraints
  - Shifts and division: Implementation **almost** complete.
  - Evaluating different encodings: Can improve performance.
- Proof production
  - Add support for proof production of the translation step. **Idea:** register each translation as generic step, and add necessary information on post-processing.
  - Translation of VERIPB proofs into the calculi that **cvc5** produces proofs.

Challenges (II)

- Proof Checking
  - Extend carcara support to blasting steps and to completely cover veriPB translated proofs.
  - Incorporate the veriPB proof checker on carcara pipeline, in order to support proofs on the coarse-grained format.

References i

[**Andreotti et al.**, **2023**] Carcara: An Efficient Proof Checker and Elaborator for SMT Proofs in the Alethe Format. (TACAS)

[**Barbosa et al.**, **2022**] cvc5: A Versatile and Industrial-Strength SMT Solver. (TACAS)

[**Elffers & Nordström**, **2018**] Divide and Conquer: Towards Faster Pseudo-Boolean Solving. (IJCAI)

[**Kaufmann et al.**, **2019**] Verifying Large Multipliers by Combining SAT and Computer Algebra. (FMCAD)

[**Liew et al.**, **2020**] Verifying Properties of Bit-Vector Multiplication Using Cutting Planes Reasoning. (FMCAD)

[**Reynolds et al.**, **2024**] Ethos: A Proof Checker for the Cooperating Proof Calculus (CPC). (cvc5/ethos, tool)

References ii

[Ritirc et al., 2017] Column-Wise Verification of Multipliers Using Computer Algebra. (FMCAD)

[Schurr et al., 2021] Alethe: Towards a Generic SMT Proof Format. (PxTP)