Bit-precise Reasoning with Parametric Bit-vectors

Zvika Berger[1], Yoni Zohar[1], Mathias Preiner[2], Aina Niemetz[2], Andrew Reynolds[3], Cesare Tinelli[3], Clark Barrett[2]

[1]Bar-Ilan University [2]Stanford University [3]The University of Iowa

Introduction

- SMT extends SAT to include first-order logic and various theories (integers, reals, arrays, strings, bit-vectors, etc.).
- SMT-solvers determine if a formula with first-order logic and theories is satisfiable.
- Standard SMT verification is limited to fixed bit-widths (e.g., verifying 32-bit code does not guarantee 64-bit safety).
- We introduce a framework to prove properties for *parametric* bit-widths k once and for all.

The Theory of Parametric Bit-vectors

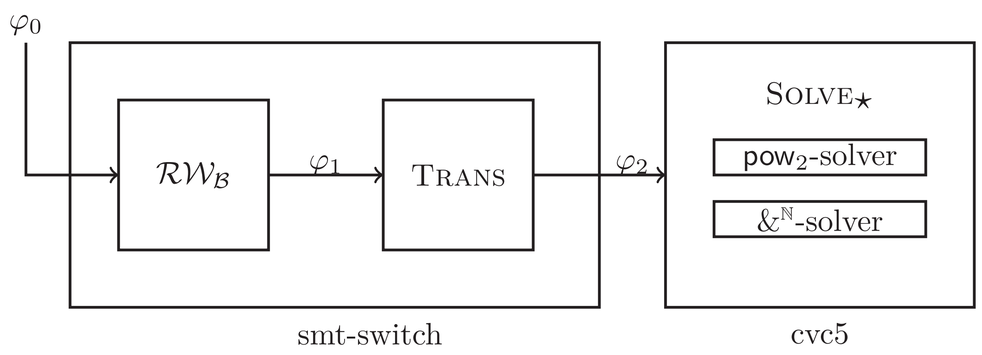
New Theory: T_{PBV}, characterized by:

- A single sort, PBV.
- An **admissibility function** (ADM) that ensures bit-width consistency (e.g., $x +_{\mathcal{B}} y$ is well-formed only if $|x| = |y|$).
- A cleaner and simpler formulation.
- Enables reasoning about bit-widths within the theory.

Satisfiability Definition: φ is admissibly T_{PBV}-satisfiable if there exists a T_{PBV}-interpretation that satisfies $\varphi \wedge \mathrm{ADM}(\varphi)$.

Solver Architecture

A solver architecture for admissible T_{PBV}-satisfiability:



$\mathcal{RW}_{\mathbf{B}}$ Dedicated rewriter for PBV formulas.

Trans Translates PBV to Integers with overflow behaviour.

Solve$_\star$ Abstraction-refinement loop for pow$_2$ and $\&^{\mathbb{N}}$.

Bitwise Elimination

Bitwise OR ($|$) and XOR ($\oplus$) are eliminated using *Hacker's Delight* identities:

$$x \mid y \approx (x +_{\mathcal{B}} y) -_{\mathcal{B}} (x \& y)$$
$$x \oplus y \approx (x \mid y) -_{\mathcal{B}} (x \& y)$$

Remaining bitwise AND ($\&$) operations are mapped to a new integer operator $\&^{\mathbb{N}}$ called piand.

We prove that this translation into the integer theory does not require modular arithmetic.

Piand Lemmas

The abstraction-refinement lemmas for $\&^{\mathbb{N}}$:

Name	Lemma	Source
base	$(k > 0 \wedge x = 1 \wedge y = 1) \Rightarrow \&^{\mathbb{N}}(k, x, y) = 1$	[CADE'19]
max	$(k > 0 \wedge \langle x \rangle_k \wedge y = \mathsf{pow}_2(k) - 1) \Rightarrow \&^{\mathbb{N}}(k, x, y) = x$	[CADE'19]
min	$y = 0 \Rightarrow \&^{\mathbb{N}}(k, x, y) = 0$	[CADE'19]
idem	$(k > 0 \wedge \langle x \rangle_k \wedge x = y) \Rightarrow \&^{\mathbb{N}}(k, x, y) = x$	[CADE'19]
contra	$(x + y) \mod \mathsf{pow}_2(k) = \mathsf{pow}_2(k) - 1 \Rightarrow \&^{\mathbb{N}}(k, x, y) = 0$	[CADE'19]
range	$0 \leq x \wedge 0 \leq y \Rightarrow 0 \leq \&^{\mathbb{N}}(k, x, y) \leq \min(x, y)$	[CADE'19]
empty	$k \leq 0 \Rightarrow \&^{\mathbb{N}}(k, x, y) = 0$	New
lsb	$x \mod 2 = 0 \Rightarrow \&^{\mathbb{N}}(k, x, y) \mod 2 = 0$	New
one	$(k > 0 \wedge y = 1) \Rightarrow \&^{\mathbb{N}}(k, x, y) = x \mod 2$	New
sum$_\geq$	$(k \geq v \wedge v > 0 \wedge \langle x \rangle_v \wedge \langle y \rangle_v) \Rightarrow \&^{\mathbb{N}}(k, x, y) = \sum_{i=0}^{v-1} \mathsf{ex}_i(x) \cdot \mathsf{ex}_i(y) \cdot 2^i$	New
sym	$(x = w \wedge y = z) \Rightarrow \&^{\mathbb{N}}(k, x, y) = \&^{\mathbb{N}}(k, z, w)$	[CADE'19]
diff	$(k > 0 \wedge z = w \wedge x \not\approx y \wedge \langle x \rangle_k \wedge \langle y \rangle_k \wedge \langle z \rangle_k) \Rightarrow (\&^{\mathbb{N}}(k, x, z) \not\approx y \vee \&^{\mathbb{N}}(k, y, w) \not\approx x)$	[CADE'19]

Benchmarks

1. **Compiler Optimization**: `alive`
   - Benchmarks from LLVM optimization.
2. **Handcrafted**: `rewrite`, `syrew`
   - benchmarks for SMT solver evaluation.
3. **SMT Internals**: `ic`, `icfb`, `lemmas`
   - `ic` and `icfb`: local search approaches.
   - `lemmas`: Designed for bit-blasting scalability.
4. **Mutated**: `mutant`
   - Mutated benchmarks derived from the other sets.

Evaluation

Configurations

Attribute	BASELINE	EAGER	PBV
Multiple Bit-widths	×	✓	✓
Lazy pow$_2$	×	×	✓
Lazy $\&^{\mathbb{N}}$	×	×	✓
$\|$*-elimination*	×	✓	✓
$\oplus$*-elimination*	×	✓	✓
$\gg$ *without* mod	×	✓	✓
New lemmas for $\&^{\mathbb{N}}$	×	✓	✓
New lemmas for pow$_2$	×	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	×	✓	✓

Results

Set	#	BASELINE	EAGER	PBV
`alive`	200	71	93	**107**
`ic`	180	43	58	**77**
`rewrite`	2006	658	1221	**1331**
`syrew`	1500	558	720	**912**
`lemmas`	70	12	14	**23**
`icfb`	46	1	9	**12**
`mutant`	9441	669	1084	**4863**
Total	13443	2012	3199	**7325**
sat		0	0	**3641**
unsat		2012	3199	**3684**

- Both **EAGER** and **PBV** improve performance on all benchmark sets.
- **PBV** is the **first to solve satisfiable** parametric formulas.

Summary

- We introduce a **new theory** for parametric bit-vectors (T_{PBV}).
- We propose a **lazy abstraction-refinement algorithm** for solving T_{PBV} formulas.
- Our lazy-based tool demonstrates **significant performance improvements** over existing approaches.
- **Future Work:** Integrate the implementation into the cvc5 production branch and explore proof production capabilities.