

Bit-precise Reasoning with Parametric Bit-vectors

Zvika Berger¹ Yoni Zohar¹ Aina Niemetz²
Mathias Preiner² Andrew Reynolds³ Clark Barrett²
Cesare Tinelli³

¹Bar Ilan University

²Stanford University

³The University of Iowa

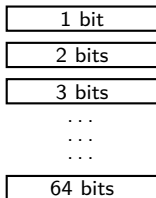
IVD 2026
Presented at SAT'25

- 1 Introduction
- 2 Theory
- 3 Solver
- 4 Evaluation

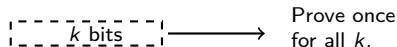
Motivation: Parametric Bit-Vectors

- ◆ Bit-precise reasoning is important for verification.
- ◆ Example 1: LLVM optimizations.
- ◆ Alive proves for specific, fixed bit-widths.
- ◆ Example 2: Rewriting rules for bit-vectors.
- ◆ Our goal: prove all bit-widths.

Fixed Bit-Width

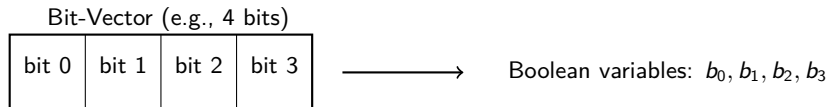


Parametric Bit-Width



Fixed-Width Bit-Vectors

- ◆ The standard approach to bit-vectors is SAT reduction.
- ◆ Each value maps to Boolean variables, one per bit.
- ◆ This reduction works only when bit-widths are known.
- ◆ A different approach is needed for parametric bit-vectors!



Parametric Bit-Vectors - Preliminary Work

- ◆ Preliminary work was introduced in [CADE'19]¹.
- ◆ Based on int-blasting.
- ◆ Theory Issues:
 - ◆ Eager translation with quantifiers.
 - ◆ Unable to reason about bit-widths.
- ◆ Implementation Issues:
 - ◆ Supports only a single parametric bit-width.
 - ◆ Ad hoc evaluation (no standalone tool).

¹Niemetz Aina, et al. Towards bit-width-independent proofs in SMT solvers. CADE'19.

Contributions

- ◆ A new theory of parametric bit-vectors.
 - ◆ Reasoning about bit-widths is supported.
- ◆ Lazy algorithm without quantified axioms.
- ◆ Implementation and evaluation.
 - ◆ Support for multiple parametric bit-widths.

Outline

- 1 Introduction
- 2 Theory
- 3 Solver
- 4 Evaluation

Problem Instance

Consider the following formula φ in PBV (Parametric Bit-Vector) theory:

$$\varphi = x \approx x +^B (x \circ x)$$

Theory Contributions

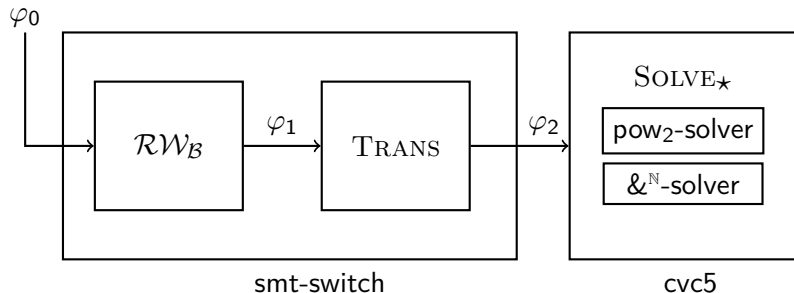
- ◆ T_{BV} cannot handle parametric bit-widths.
- ◆ New theory with single sort PBV.
- ◆ We add a function to handle type constraints.
- ◆ Improved theory over [CADE'19]:
 - ◆ More expressive: supports bit-width reasoning.
 - ◆ Simpler and clearer.

Outline

- 1 Introduction
- 2 Theory
- 3 Solver**
- 4 Evaluation

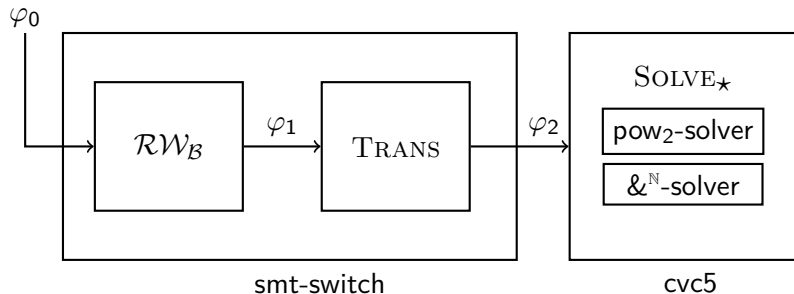
PBV Solver - Architecture

- ◆ A solver for T_{PBV} -satisfiability:



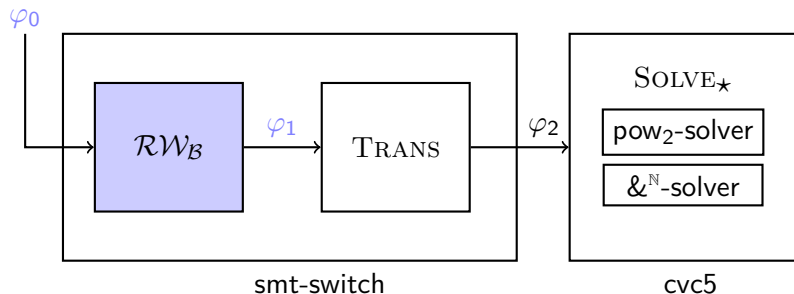
PBV Solver - Architecture

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

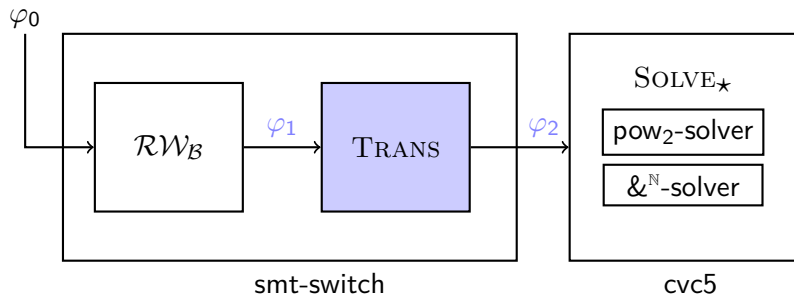


smt-switch is an open-source API for SMT solving.

PBV Solver - Architecture

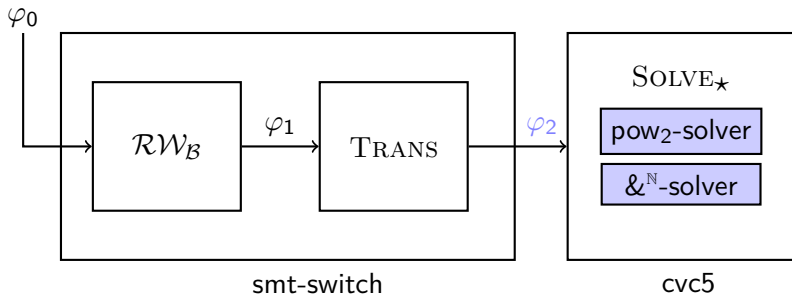


PBV Solver - Architecture



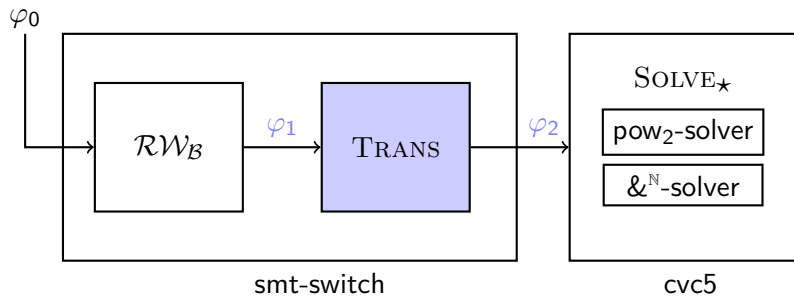
PBV Solver - Architecture

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



cvc5 is an open-source SMT solver.

PBV Solver - TRANS



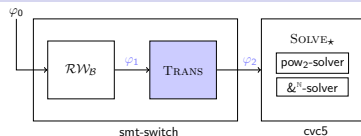
Translation Function CONV

function CONV(e)

match e :

z $\rightarrow z$
 $|t|$ $\rightarrow \kappa(t)$
 $\text{to-pbv}(k, t)$ $\rightarrow \text{CONV}(t) \bmod \text{pow}_2(k)$
 x $\rightarrow \chi(x)$
 $t_1 \approx t_2$ $\rightarrow \text{CONV}(t_1) \approx \text{CONV}(t_2)$
 $\bowtie_u(t_1, t_2)$ $\rightarrow \text{CONV}(t_1) \bowtie_u \text{CONV}(t_2)$
 $\bowtie_s(t_1, t_2)$ $\rightarrow \text{uts}(\kappa(t_1), \text{CONV}(t_1)) \bowtie_s \text{uts}(\kappa(t_2), \text{CONV}(t_2))$
 $t_1 +^B t_2$ $\rightarrow (\text{CONV}(t_1) + \text{CONV}(t_2)) \bmod \text{pow}_2(\kappa(t_1))$
 $t_1 -^B t_2$ $\rightarrow (\text{CONV}(t_1) - \text{CONV}(t_2)) \bmod \text{pow}_2(\kappa(t_1))$
 $t_1 \cdot^B t_2$ $\rightarrow (\text{CONV}(t_1) \cdot \text{CONV}(t_2)) \bmod \text{pow}_2(\kappa(t_1))$
 $t_1 \text{div}^B t_2$ $\rightarrow \text{ite}(\text{CONV}(t_2) \approx 0, \text{pow}_2(\kappa(t_1)) - 1, \text{CONV}(t_1) \text{div} \text{CONV}(t_2))$
 $t_1 \bmod^B t_2$ $\rightarrow \text{ite}(\text{CONV}(t_2) \approx 0, \text{CONV}(t_1), \text{CONV}(t_1) \bmod \text{CONV}(t_2))$
 $\sim t$ $\rightarrow \text{pow}_2(\kappa(t)) - (\text{CONV}(t) + 1)$
 $-^B t$ $\rightarrow (\text{pow}_2(\kappa(t)) - \text{CONV}(t)) \bmod \text{pow}_2(\kappa(t))$
 $t_1 \ll t_2$ $\rightarrow (\text{CONV}(t_1) \cdot \text{pow}_2(\text{CONV}(t_2))) \bmod \text{pow}_2(\kappa(t_1))$
 $t_1 \gg t_2$ $\rightarrow \text{CONV}(t_1) \text{div} \text{pow}_2(\text{CONV}(t_2))$
 $t_1 \& t_2$ $\rightarrow \&^N(\kappa(t_1), \text{CONV}(t_1), \text{CONV}(t_2))$
 $t_1 | t_2$ $\rightarrow \text{CONV}(t_1) + \text{CONV}(t_2) - \&^N(\kappa(t_1), \text{CONV}(t_1), \text{CONV}(t_2))$
 $t_1 \oplus t_2$ $\rightarrow \text{CONV}(t_1) + \text{CONV}(t_2) - 2 \cdot \&^N(\kappa(t_1), \text{CONV}(t_1), \text{CONV}(t_2))$
 $t_1 \circ t_2$ $\rightarrow \text{CONV}(t_1) \cdot \text{pow}_2(\kappa(t_2)) + \text{CONV}(t_2)$
 $t[i : j]$ $\rightarrow (\text{CONV}(t) \text{div} \text{pow}_2(j)) \bmod \text{pow}_2(i - j + 1)$
 $\text{ext}_z(n, t)$ $\rightarrow \text{CONV}(t)$
 $\text{ext}_s(n, t)$ $\rightarrow \text{ite}(\text{CONV}(t[\kappa(t) - 1]) \approx 1, (\text{pow}_2(n) - 1) \cdot \text{pow}_2(\kappa(t)) + \text{CONV}(t), \text{CONV}(t))$
 $\bullet(t_1, \dots, t_n)$ $\rightarrow \bullet(\text{CONV}(t_1), \dots, \text{CONV}(t_n))$

end function



Translation Function CONV

function CONV(*e*)

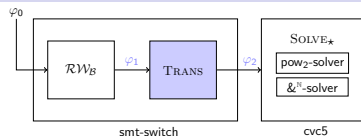
match *e*:

z → *z*
 |*t*| → $\kappa(t)$
 to-pbv(*k*, *t*) → $\text{CONV}(t) \bmod \text{pow}_2(k)$
x → $\chi(x)$
 $t_1 \approx t_2$ → $\text{CONV}(t_1) \approx \text{CONV}(t_2)$
 $\bowtie_u(t_1, t_2)$ → $\text{CONV}(t_1) \bowtie \text{CONV}(t_2)$
 $\bowtie_s(t_1, t_2)$ → $\text{uts}(\kappa(t_1), \text{CONV}(t_1)) \bowtie \text{uts}(\kappa(t_2), \text{CONV}(t_2))$
 $t_1 +^B t_2$ → $(\text{CONV}(t_1) + \text{CONV}(t_2)) \bmod \text{pow}_2(\kappa(t_1))$


$t_1 +^B t_2$ → $(\text{CONV}(t_1) + \text{CONV}(t_2)) \bmod \text{pow}_2(\kappa(t_1))$

$t_1 \text{ div } t_2$ → $\text{ite}(\text{CONV}(t_2) \approx 0, \text{pow}_2(\kappa(t_1)) - 1, \text{CONV}(t_1) \text{ div } \text{CONV}(t_2))$
 $t_1 \bmod^B t_2$ → $\text{ite}(\text{CONV}(t_2) \approx 0, \text{CONV}(t_1), \text{CONV}(t_1) \bmod \text{CONV}(t_2))$
 $\sim t$ → $\text{pow}_2(\kappa(t)) - (\text{CONV}(t) + 1)$
 $-^B t$ → $(\text{pow}_2(\kappa(t)) - \text{CONV}(t)) \bmod \text{pow}_2(\kappa(t))$
 $t_1 \ll t_2$ → $(\text{CONV}(t_1) \cdot \text{pow}_2(\text{CONV}(t_2))) \bmod \text{pow}_2(\kappa(t_1))$
 $t_1 \gg t_2$ → $\text{CONV}(t_1) \text{ div } \text{pow}_2(\text{CONV}(t_2))$
 $t_1 \& t_2$ → $\&^N(\kappa(t_1), \text{CONV}(t_1), \text{CONV}(t_2))$
 $t_1 | t_2$ → $\text{CONV}(t_1) + \text{CONV}(t_2) - \&^N(\kappa(t_1), \text{CONV}(t_1), \text{CONV}(t_2))$
 $t_1 \oplus t_2$ → $\text{CONV}(t_1) + \text{CONV}(t_2) - 2 \cdot \&^N(\kappa(t_1), \text{CONV}(t_1), \text{CONV}(t_2))$
 $t_1 \circ t_2$ → $\text{CONV}(t_1) \cdot \text{pow}_2(\kappa(t_2)) + \text{CONV}(t_2)$
 $t[i : j]$ → $(\text{CONV}(t) \text{ div } \text{pow}_2(j)) \bmod \text{pow}_2(i - j + 1)$
 $\text{ext}_z(n, t)$ → $\text{CONV}(t)$
 $\text{ext}_s(n, t)$ → $\text{ite}(\text{CONV}(t[\kappa(t) - 1]) \approx 1, (\text{pow}_2(n) - 1) \cdot \text{pow}_2(\kappa(t)) + \text{CONV}(t), \text{CONV}(t))$
 $\bullet(t_1, \dots, t_n)$ → $\bullet(\text{CONV}(t_1), \dots, \text{CONV}(t_n))$

end function



Improved Bitwise-Or Translation

$2^{31} + 1 = 641 \cdot 6700417$ $\text{avg}(x, y) = (x \& y) + ((x \oplus y) \gg 1)$ 
 $2^{30} + 1 = 274177 \cdot 67280421310721$ $x - y = x + \bar{y} + 1$
 $[a] + [b] \leq [a + b] \leq [a] + [b] + 1$ $\text{pop}(x) = -\sum_{i=0}^{31} (x \text{ rot } i)$
 George Boole 1815 - 1864 $\lfloor \sqrt{11111111} \rfloor = 1111$
 $(x \oplus 0) = (x | -x) \gg 31$ $\text{mux}(x, y, m) = ((x \oplus y) \& m) \oplus y$
 $A(n, d) = A(n - 1, d - 1), d \text{ even}$ $-\bar{x} = x + 1$

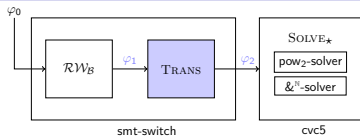
Hacker's Delight

SECOND EDITION _{=0.01010101...}

$2^{31} - 1 = 11100001$
 $n = -2^{31} b_{31} + 2^{30} b_{30} + 2^{29} b_{29} + \dots + 2^0 b_0$
 $x \lceil \rceil = - \lfloor -x \rfloor$ $f(x, y, z) = g(x, y) \oplus zh(x, y)$
 Num factors of 2 in $x = \log_2(x \& (-x)), x \neq 0$ $\text{rjust}(x) = x \gg (x \& -x), x \neq 0$

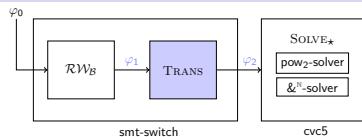
$$\begin{aligned} x \oplus y &= (x | y) - (x \& y) \\ x + y &= (x | y) + (x \& y) \end{aligned}$$

HENRY S. WARREN, JR.



$$\begin{aligned} x \oplus y &= (x | y) - (x \& y) \\ x + y &= (x | y) + (x \& y) \end{aligned}$$

Improved Bitwise-Or Translation



Hacker's Delight:

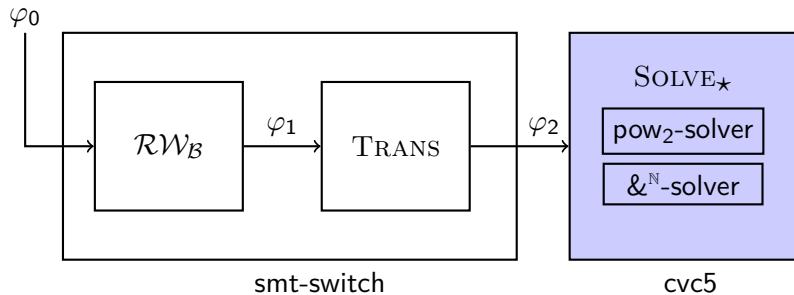
$$t_1 | t_2 \mapsto (t_1 + t_2) - (t_1 \& t_2)$$

Translation to integers:

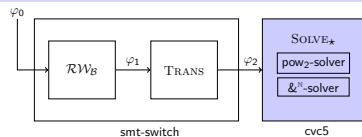
$$t_1 | t_2 \mapsto (((\text{CONV}(t_1) + \text{CONV}(t_2)) \bmod \text{pow}_2(\kappa(t_1))) - \&^{\mathbb{N}}(\kappa(t_1), \text{CONV}(t_1), \text{CONV}(t_2))) \bmod \text{pow}_2(\kappa(t_1)))$$

Improvement:

$$t_1 | t_2 \mapsto \text{CONV}(t_1) + \text{CONV}(t_2) - \&^{\mathbb{N}}(\kappa(t_1), \text{CONV}(t_1), \text{CONV}(t_2))$$

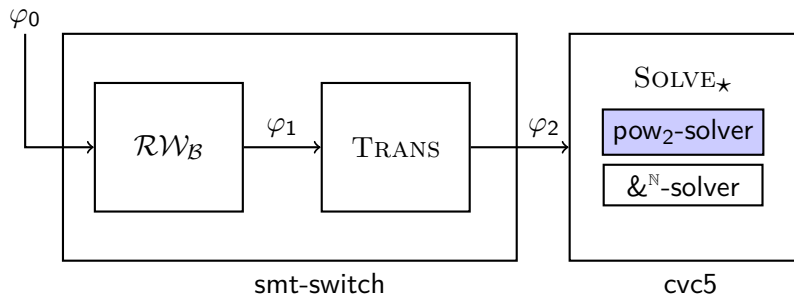


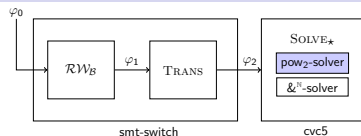
Lazy vs. Eager Algorithm



- ◆ Eager: adds all axioms upfront.
- ◆ Lazy: adds lemmas only when needed.
- ◆ Lazy algorithm uses CEGAR² to add lemmas.
- ◆ The key is selecting the right lemmas.

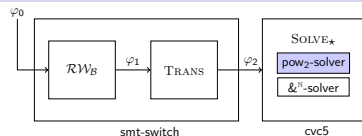
²Clarke et al., "Counterexample-Guided Abstraction Refinement", CAV 2000.





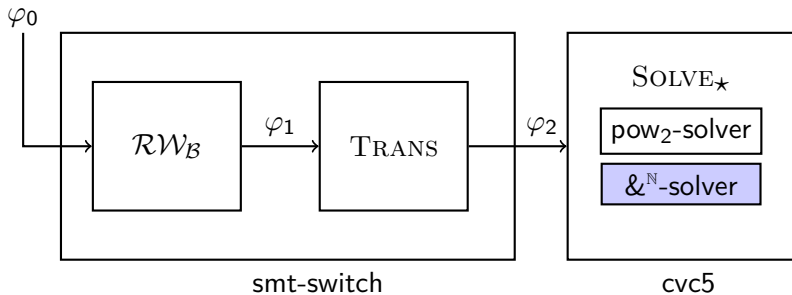
Name	Lemma	Source
$\text{pow}_2(x) \in S$		
positive	$0 \leq x \Rightarrow \text{pow}_2(x) > 0$	[CADE'19]
even	$x \geq 1 \Rightarrow \text{pow}_2(x) \bmod 2 \approx 0$	[CADE'19]
div	$x \geq 0 \Rightarrow x \text{ div } \text{pow}_2(x) \approx 0$	[CADE'19]
neg	$x < 0 \Rightarrow \text{pow}_2(x) \approx 0$	new
bound	$(x \geq v \wedge v \geq 7) \Rightarrow v \cdot x + v^2 < \text{pow}_2(x)$	new
value	$(0 \leq x \wedge x \approx v) \Rightarrow \text{pow}_2(x) \approx 2^v$	new
$\text{pow}_2(x), \text{pow}_2(y) \in S$		
monotonicity	$(0 \leq x \wedge x < y) \Rightarrow \text{pow}_2(x) < \text{pow}_2(y)$	[CADE'19]

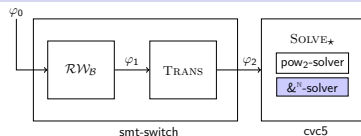
- ◆ v is the value chosen in the model returned by SOLVE_* .



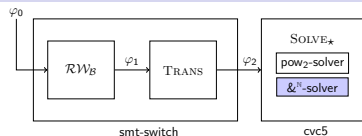
Name	Lemma	Source
$\text{pow}_2(x) \in S$		
positive	$0 \leq x \Rightarrow \text{pow}_2(x) > 0$	[CADE'19]
	even: $x \geq 1 \Rightarrow \text{pow}_2(x) \bmod 2 \approx 0$	
neg	$x < 0 \Rightarrow \text{pow}_2(x) \approx 0$	new
bound	$(x \geq v \wedge v \geq 7) \Rightarrow v \cdot x + v^2 < \text{pow}_2(x)$	new
value	$(0 \leq x \wedge x \approx v) \Rightarrow \text{pow}_2(x) \approx 2^v$	new
$\text{pow}_2(x), \text{pow}_2(y) \in S$		
monotonicity	$(0 \leq x \wedge x < y) \Rightarrow \text{pow}_2(x) < \text{pow}_2(y)$	[CADE'19]

- ◆ v is the value chosen in the model returned by SOLVE_\star .





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



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

Outline

- 1 Introduction
- 2 Theory
- 3 Solver
- 4 Evaluation**

Benchmark Sets

Compiler Optimizations

- ◆ *alive* (200 benchmarks)³

Crafted benchmarks

- ◆ *rewrite* (2006 benchmarks)⁴
- ◆ *syrew* (1500 benchmarks)⁴
- ◇ enumerated by SyGuS (cvc5)
for $w = 4$

SMT solvers internals

- ◆ *lemmas* (70 benchmarks)⁵
- ◆ *ic* (180 benchmarks)⁵
- ◆ *icfb* (46 benchmarks)⁶
- ◇ quantified benchmarks

Mutated Benchmarks

- ◆ *mut* (9442 benchmarks):
- ◇ mutated benchmarks derived
from the other sets.

⁴Niemetz Aina, et al. Towards bit-width-independent proofs in SMT solvers. CADE'19.

⁵Niemetz Aina, et al. Scalable bit-blasting with abstractions. CAV'24.

⁶Niemetz Aina, et al. On solving quantified bit-vector constraints using invertibility conditions. CAV'18

⁷Niemetz Aina, et al. Ternary Propagation-Based Local Search for More Bit-Precise Reasoning, FMCAD'20.

Evaluation Configuration

Attribute	BASELINE	EAGER	PBV
<i>Multiple Bit-widths</i>	×	✓	✓
<i>Lazy pow₂</i>	×	×	✓
<i>Lazy &^N</i>	×	×	✓
<i> -elimination</i>	×	✓	✓
<i>⊕ -elimination</i>	×	✓	✓
<i>>> without mod</i>	×	✓	✓
<i>New lemmas for &^N</i>	×	✓	✓
<i>New lemmas for pow₂</i>	×	✓	✓
<i>RW_B</i>	×	✓	✓

Evaluation Results

Benchmarks	#	BASELINE	EAGER	PBV
<i>alive</i>	200	71	93	107
<i>ic</i>	180	43	58	77
<i>rewrite</i>	2006	658	1221	1331
<i>syrew</i>	1500	558	720	912
<i>lemmas</i>	70	12	14	23
<i>icfb</i>	46	1	9	12
<i>mut</i>	9441	669	1084	4863
<i>total</i>	13443	2012	3199	7325

- ◆ Both EAGER and PBV improve performance on all benchmark sets.

Evaluation Results

Benchmarks	#	BASELINE	EAGER	PBV
<i>alive</i>	200	71	93	107
<i>ic</i>	180	43	58	77
<i>rewrite</i>	2006	658	1221	1331
<i>syrew</i>	1500	558	720	912
<i>lemmas</i>	70	12	14	23
<i>icfb</i>	46	1	9	12
<i>mut</i>	9441	669	1084	4863
<i>total</i>	13443	2012	3199	7325
<i>sat</i>		0	0	3641
<i>unsat</i>		2012	3199	3684

- ◆ Both EAGER and PBV improve performance on all benchmark sets.
- ◆ BASELINE and EAGER did not solve any satisfiable benchmarks.

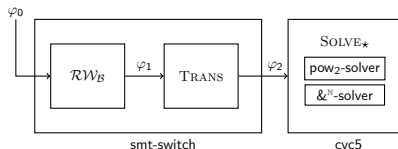
Summary and Future Work

◆ Summary:

- ◆ New theory for parametric bit-vectors
- ◆ Ability to reason about bit-widths
- ◆ Lazy algorithm for parametric bit-vectors
- ◆ Lazy-based tool shows major improvement

◆ Future Work:

- ◆ Integrate the implementation into cvc5.
- ◆ Explore proof production for parametric bit-vectors.



Thank you for listening!

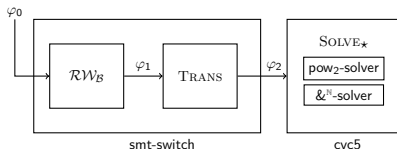
Summary and Future Work

◆ Summary:

- ◆ New theory for parametric bit-vectors
- ◆ Ability to reason about bit-widths
- ◆ Lazy algorithm for parametric bit-vectors
- ◆ Lazy-based tool shows major improvement

◆ Future Work:

- ◆ Integrate the implementation into cvc5.
- ◆ Explore proof production for parametric bit-vectors.



Thank you for listening!

Admissible Satisfiability – Satisfiable Example

- ◆ φ is **admissibly T_{PBV} -satisfiable** if there exists a T_{PBV} -interpretation that satisfies $\varphi \wedge \text{ADM}(\varphi)$.

Example

$$\varphi = y \approx z \circ w$$

$\text{ADM}(\varphi)$:

$$|y| \approx |z| + |w| \wedge |y| > 0 \wedge |z| > 0 \wedge |w| > 0$$

An admissible T_{PBV} -interpretation \mathcal{I} :

$$y^{\mathcal{I}} = 00, \quad z^{\mathcal{I}} = 0, \quad w^{\mathcal{I}} = 0$$

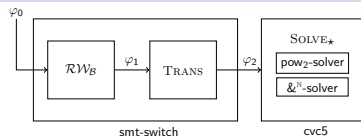
Thus, φ is admissibly T_{PBV} -satisfiable.

Parametric Bit Vector Rewriter – \mathcal{RW}_B (Part 1)

Rule Name	Term	Simplify
bv-concat-extract-merge	$(\text{concat } (\text{extract } k \ (+ \ j \ 1) \ s) \ (\text{extract } j \ i \ s))$	$(\text{extract } k \ i \ s)$
bv-extract-extract	$(\text{extract } l \ k \ (\text{extract } j \ i \ x))$	$(\text{extract } (+ \ i \ l) \ (+ \ i \ k) \ x)$
bv-extract-whole	$(\text{extract } k \ 0 \ x)$	x
bv-add-zero	$(\text{bvadd } x \ 0)$	x
bv-reverse-extract-and	$(\text{bvand } (\text{extract } j \ i \ x) \ (\text{extract } j \ i \ y))$	$(\text{extract } j \ i \ (\text{bvand } x \ y))$
bv-xor-simplify-2	$(\text{bvxor } x \ (\text{bvnot } x))$	$(\text{bvnot } 0)$
bv-or-zero	$(\text{bvor } x \ (\text{int_to_pbv } k \ 0))$	x
bv-mul-one	$(\text{bvmul } x \ (\text{int_to_pbv } k \ 1))$	x
bv-mul-zero	$(\text{bvmul } x \ (\text{int_to_pbv } k \ 0))$	$(\text{int_to_pbv } k \ 0)$
bv-zero-extend-eliminate	$(\text{zero-extend } 0 \ x)$	x
bv-sign-extend-eliminate	$(\text{sign-extend } 0 \ x)$	x
bv-not-neq	$(= \ x \ (\text{bvnot } x))$	false
bv-neg-sub	$(\text{bvneg } (\text{bvsub } x \ y))$	$(\text{bvsub } y \ x)$
bv-neg-idemp	$(\text{bvneg } (\text{bvneg } x))$	x
bv-ugt-eliminate	$(\text{bvugt } x \ y)$	$(\text{bvult } y \ x)$
bv-uge-eliminate	$(\text{bvuge } x \ y)$	$(\text{bvule } y \ x)$
bv-sgt-eliminate	$(\text{bvsgt } x \ y)$	$(\text{bvslt } y \ x)$
bv-sge-eliminate	$(\text{bvsgt } x \ y)$	$(\text{bvslsle } y \ x)$
bv-shl-by-const-0	$(\text{bvshl } x \ (\text{int_to_pbv } k \ 0))$	x
bv-shl-by-const-2	$(\text{bvshl } x \ (\text{int_to_pbv } k \ k))$	$(\text{int_to_pbv } k \ 0)$
bv-lshr-by-const-0	$(\text{bvlsht } x \ (\text{int_to_pbv } k \ 0))$	x
bv-lshr-by-const-2	$(\text{bvlsht } x \ (\text{int_to_pbv } k \ k))$	$(\text{int_to_pbv } k \ 0)$

Parametric Bit Vector Rewriter – \mathcal{RW}_B (Part 2)

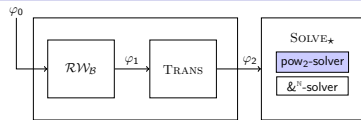
Rule Name	Term	Simplify
bv-ashr-by-const-0	(bvashr x (int_to_pbv k 0))	x
bv-bitwise-idemp-2	(bvor x x)	x
bv-or-one	(bvor x (bvnot x))	(bvnot (int_to_pbv k 0))
bv-xor-duplicate	(bvxor x x)	(int_to_pbv k 0)
bv-xor-zero	(bvxor x (int_to_pbv k 0))	x
bv-bitwise-not-or	(bvor x (bvnot x))	(bvnot (int_to_pbv k 0))
bv-xor-not	(bvxor (bvnot x) (bvnot y))	(bvxor x y)
bv-not-idemp	(bvnot (bvnot x))	x
bv-ult-zero-1	(bvult (int_to_pbv k 0) x)	(not (= x (int_to_pbv k 0)))
bv-ult-zero-2	(bvult x (int_to_pbv k 0))	false
bv-ult-self	(bvult x x)	false
bv-lt-self	(bvslt x x)	false
bv-ule-self	(bvule x x)	true
bv-ule-zero	(bvule x (int_to_pbv k 0))	(= x (int_to_pbv k 0))
bv-zero-ule	(bvule (int_to_pbv k 0) x)	true
bv-sle-self	(bvslsle x x)	true
bv-ule-max	(bvule x (bvnot x))	true
bv-udiv-zero	(bvudiv x (int_to_pbv k 0))	(bvnot (int_to_pbv k 0))
bv-udiv-one	(bvudiv x (int_to_pbv k 1))	x
bv-urem-one	(bvurem x (int_to_pbv k 1))	(int_to_pbv k 0)
bv-urem-self	(bvurem x x)	(int_to_pbv k 0)
bv-shl-zero	(bvshl (int_to_pbv k 0) x)	(int_to_pbv k 0)
bv-lshr-zero	(bvlsht (int_to_pbv k 0) x)	(int_to_pbv k 0)
bv-ashr-zero	(bvashr (int_to_pbv k 0) x)	(int_to_pbv k 0)
bv-ult-one	(bvult x (int_to_pbv k 1))	(= x (int_to_pbv k 0))



- ◆ Rewrite rules to eliminate modulo operations:

$+ \frac{1}{rr}$	$((x \bmod \text{pow}_2(k)) + (y \bmod \text{pow}_2(k))) \bmod \text{pow}_2(k)$	\rightarrow	$(x + y) \bmod \text{pow}_2(k)$
$+ \frac{2}{rr}$	$((x \bmod \text{pow}_2(k)) + y) \bmod \text{pow}_2(k)$	\rightarrow	$(x + y) \bmod \text{pow}_2(k)$
$+ \frac{3}{rr}$	$(x + (y \bmod \text{pow}_2(k))) \bmod \text{pow}_2(k)$	\rightarrow	$(x + y) \bmod \text{pow}_2(k)$
$- \frac{1}{rr}$	$((x \bmod \text{pow}_2(k)) - (y \bmod \text{pow}_2(k))) \bmod \text{pow}_2(k)$	\rightarrow	$(x - y) \bmod \text{pow}_2(k)$
$- \frac{2}{rr}$	$((x \bmod \text{pow}_2(k)) - y) \bmod \text{pow}_2(k)$	\rightarrow	$(x - y) \bmod \text{pow}_2(k)$
$- \frac{3}{rr}$	$(x - (y \bmod \text{pow}_2(k))) \bmod \text{pow}_2(k)$	\rightarrow	$(x - y) \bmod \text{pow}_2(k)$
$\cdot \frac{1}{rr}$	$((x \bmod \text{pow}_2(k)) \cdot (y \bmod \text{pow}_2(k))) \bmod \text{pow}_2(k)$	\rightarrow	$(x \cdot y) \bmod \text{pow}_2(k)$
$\cdot \frac{2}{rr}$	$((x \bmod \text{pow}_2(k)) \cdot y) \bmod \text{pow}_2(k)$	\rightarrow	$(x \cdot y) \bmod \text{pow}_2(k)$
$\cdot \frac{3}{rr}$	$(x \cdot (y \bmod \text{pow}_2(k))) \bmod \text{pow}_2(k)$	\rightarrow	$(x \cdot y) \bmod \text{pow}_2(k)$

- ◆ cvc5 does not implement them: modulo assumed positive.



Example

$\varphi = (x + {}^B x) + {}^B x$, where $\kappa(x) = k$.

$\text{CONV}(\varphi)$:

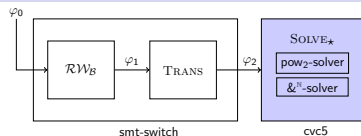
$$(((x' + x') \bmod 2^k) + x') \bmod 2^k$$

$\mathcal{RW}_A(\text{CONV}(\varphi))$:

$$((x' + x') + x') \bmod 2^k$$

where $x' = \chi(x)$.

CEGAR Procedure



function SOLVE_★(φ)

$\Gamma \leftarrow \{\varphi\}$

loop

$result, \mathcal{I} \leftarrow \text{SOLVE}(\Gamma)$

if $result$ is “unsat” **return** “unsat”

$\mathcal{L} \leftarrow \mathcal{L}_{\&^N}(\{\&^N(k, t_1, t_2) \mid \&^N(k, t_1, t_2) \in \text{Terms}(\Gamma)\}, \mathcal{I})$

$\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{\text{pow}_2}(\{\text{pow}_2(t) \mid \text{pow}_2(t) \in \text{Terms}(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$

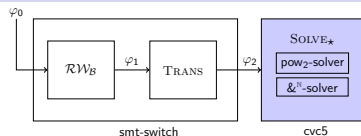
if $\mathcal{I} \models \mathcal{L}$ **return** “sat”

$\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L} \text{ and } \mathcal{I} \not\models \psi\}$

end loop

end function

CEGAR Procedure



function $SOLVE_{\star}(\varphi)$

$\Gamma \leftarrow \{\varphi\}$

loop

$result, \mathcal{I} \leftarrow SOLVE(\Gamma)$

if $result$ is “unsat” **return** “unsat”

$\mathcal{L} \leftarrow \mathcal{L}_{\&^N}(\{\&^N(k, t_1, t_2) \mid \&^N(k, t_1, t_2) \in Terms(\Gamma)\}, \mathcal{I})$

$\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{pow_2}(\{pow_2(t) \mid pow_2(t) \in Terms(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$

if $\mathcal{I} \models \mathcal{L}$ **return** “sat”

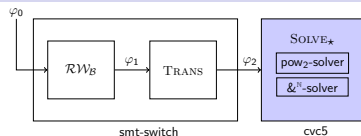
$\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L} \text{ and } \mathcal{I} \not\models \psi\}$

end loop

end function

SOLVE: SMT solver for integer arithmetic with uninterpreted functions.

CEGAR Procedure



function $SOLVE_{\star}(\varphi)$

$\Gamma \leftarrow \{\varphi\}$

loop

$result, \mathcal{I} \leftarrow SOLVE(\Gamma)$

if $result$ is "unsat" **return** "unsat"

$\mathcal{L} \leftarrow \mathcal{L}_{\&N}(\{\&N(k, t_1, t_2) \mid \&N(k, t_1, t_2) \in Terms(\Gamma)\}, \mathcal{I})$

$\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{pow_2}(\{pow_2(t) \mid pow_2(t) \in Terms(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$

if $\mathcal{I} \models \mathcal{L}$ **return** "sat"

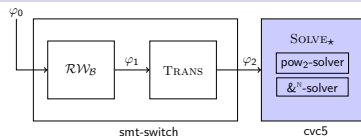
$\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L} \text{ and } \mathcal{I} \not\models \psi\}$

end loop

end function

$SOLVE$ returns "unsat" \Rightarrow
unsat under all function interpretations.

CEGAR Procedure



function $\text{SOLVE}_\star(\varphi)$

$\Gamma \leftarrow \{\varphi\}$

loop

$result, \mathcal{I} \leftarrow \text{SOLVE}(\Gamma)$

if $result$ is “unsat” **return** “unsat”

$\mathcal{L} \leftarrow \mathcal{L}_{\&^N}(\{\&^N(k, t_1, t_2) \mid \&^N(k, t_1, t_2) \in \text{Terms}(\Gamma)\}, \mathcal{I})$

$\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{\text{pow}_2}(\{\text{pow}_2(t) \mid \text{pow}_2(t) \in \text{Terms}(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$

if $\mathcal{I} \models \mathcal{L}$ **return** “sat”

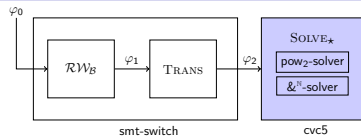
$\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L} \text{ and } \mathcal{I} \not\models \psi\}$

end loop

end function

Interpretation satisfies all lemmas \Rightarrow
formula is satisfiable.

CEGAR Procedure



function $\text{SOLVE}_*(\varphi)$

$\Gamma \leftarrow \{\varphi\}$

loop

$result, \mathcal{I} \leftarrow \text{SOLVE}(\Gamma)$

if $result$ is "unsat" **return** "unsat"

$\mathcal{L} \leftarrow \mathcal{L}_{\&^N}(\{\&^N(k, t_1, t_2) \mid \&^N(k, t_1, t_2) \in \text{Terms}(\Gamma)\}, \mathcal{I})$

$\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{\text{pow}_2}(\{\text{pow}_2(t) \mid \text{pow}_2(t) \in \text{Terms}(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$

if $\mathcal{I} \models \mathcal{L}$ **return** "sat"

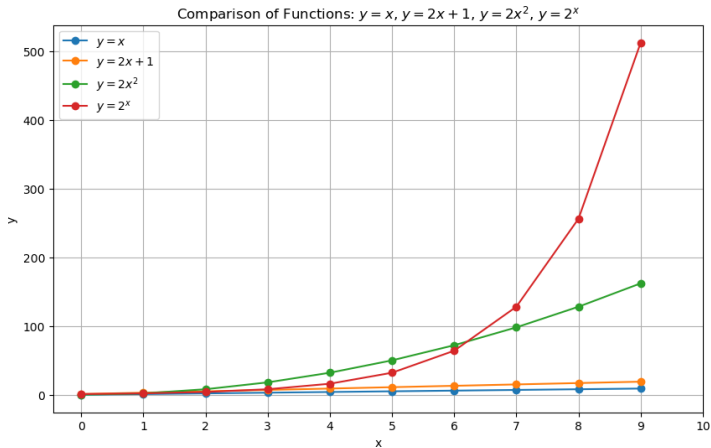
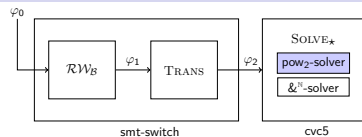
$\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L} \text{ and } \mathcal{I} \not\models \psi\}$

end loop

end function

← Add to the formula all lemmas not satisfied by the model.

bound Lemma



Evaluation on *alive*

Family	Considered	Proved			
		⁷	EAGER	PBV	VBS
AddSub (52)	19	9	8	12	16
MulDivRem (29)	13	3	8	7	10
AndOrXor (162)	130	60	61	77	88
Select (51)	27	16	13	8	16
Shifts (17)	11	0	3	3	3
LoadStoreAlloca (9)	0	0	0	0	0
Total (320)	200	88	93	107	133

⁷Niemetz, Aina, Towards bit-width-independent proofs in SMT solvers. CADE'19.

Evaluation on *ic*

$\ell[x]$	\approx	$\not\approx$	$<_u$	$>_u$	\leq_u	\geq_u	$<_s$	$>_s$	\leq_s	\geq_s
$-^B x \boxtimes t$	✓✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓✓	✓
$\sim x \boxtimes t$	✓✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓✓	✓✓
$x \& s \boxtimes t$	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✗	✗	✗	✗
$x s \boxtimes t$	✓	✓✓	✓✓	✓	✓✓	✓✓	✗	✗	✗	✗
$x \ll s \boxtimes t$	✓	✓	✓✓	✓	✓✓	✓	✗	✗	✗	✗
$s \ll x \boxtimes t$	✓✓	✓	✓	✓	✓✓	✓	✗	✓	✗	✓
$x \gg s \boxtimes t$	✓✓	✓	✓✓	✓	✓✓	✓✓	✓	✗	✓	✗
$s \gg x \boxtimes t$	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓	✓
$x \gg_a s \boxtimes t$	✓	✓	✓✓	✓✓	✓✓	✓	✗	✓	✗	✓
$s \gg_a x \boxtimes t$	✓✓	✓	✓	✓	✓✓	✓	✗	✗	✗	✓
$x +^B s \boxtimes t$	✓✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓✓	✓✓
$x \cdot^B s \boxtimes t$	✗	✓	✓✓	✗	✓✓	✗	✗	✗	✗	✗
$x \text{div}^B s \boxtimes t$	✓	✓✓	✓✓	✓✓	✓✓	✗	✓	✓	✓	✓
$s \text{div}^B x \boxtimes t$	✓	✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓✓	✗
$x \text{mod}^B s \boxtimes t$	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✗	✓	✗	✓
$s \text{mod}^B x \boxtimes t$	✗	✓✓	✓✓	✓✓	✓✓	✓	✓	✗	✓	✗
$x \circ s \boxtimes t$	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗
$s \circ x \boxtimes t$	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗

Mutated Benchmarks

- ◆ For each benchmark, we generate several mutations.
- ◆ A mutation of φ replaces the first occurrence of:
 - ◇ $\&\leftrightarrow|$
 - ◇ $+^B \leftrightarrow -^B$
 - ◇ $-^B \leftrightarrow \sim$
 - ◇ $\ll \leftrightarrow \gg$
 - ◇ $\text{to-pbv}(k, 1) \leftrightarrow \text{to-pbv}(k, 0)$
 - ◇ $\text{to-pbv}(k, k) \leftrightarrow \text{to-pbv}(k, 0)$
- ◆ A mutated version demonstrated a bug in the benchmark.

Evaluation With Multiple Bit-Widths

Benchmarks	#	BASELINE	EAGER	PBV	VBS
<i>alive</i>	17	0	9	10	10
<i>ic</i>	20	0	5	8	8
<i>rewrite</i>	0	0	0	0	0
<i>syrew</i>	0	0	0	0	0
<i>lemmas</i>	2	0	0	0	0
<i>icfb</i>	16	0	8	10	10
<i>mut</i>	316	0	88	150	151
<i>total</i>	371	0	110	178	179
sat		0	0	20	20
unsat		0	110	158	159
time-solved		0	277	30	26
mem-solved		0	796	666	659

- ◆ BASELINE does not support multiple bit-widths.
- ◆ PBVsolves more cases and reduces solving time.

icfb Incorrect Condition

- ◆ Incorrect condition from [FMCAD'20]⁸:

$$(t \Rightarrow s \neq 0 \wedge x_{lo} <_u s) \wedge (\sim t \Rightarrow (s \geq_u x) = t)$$

- ◆ The corrected condition is:

$$(t \Rightarrow s \neq 0 \wedge x_{lo} <_u s) \wedge (\sim t \Rightarrow x_{hi} \geq_u s)$$

- ◆ We were able to detect the error in the original benchmark.
- ◆ However, we were unable to prove the corrected condition.

⁸Niemetz, Preiner, Ternary propagation-based local search for more bit-precise reasoning, FMCAD'20.

Evaluation Attributes

Attribute	OR-E	XOR-E	SH-M-E	ALL-E	POW2++	PIAND++	POW2-L	PIAND-L
Target Theory	$T_{IA}(\text{pow}_2, \&^{\mathbb{N}}, ^{\mathbb{N}}, \oplus^{\mathbb{N}})$						T_3	T_4
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	×	×	×	×	✓	×	✓	×
No redundant axioms	×	×	×	×	×	×	×	×
\mathcal{RW}_B	×	×	×	×	×	×	×	×
\mathcal{RW}_A	×	×	×	×	×	×	×	×

$T_3 = T_{IA}(\text{pow}_{2^*}, \&^{\mathbb{N}})$: pow_2 has a fixed interpretation, $\&^{\mathbb{N}}$ is freely interpreted.

$T_4 = T_{IA}(\text{pow}_2, \&^{\mathbb{N}}_*)$: $\&^{\mathbb{N}}$ has a fixed interpretation, pow_2 is freely interpreted.

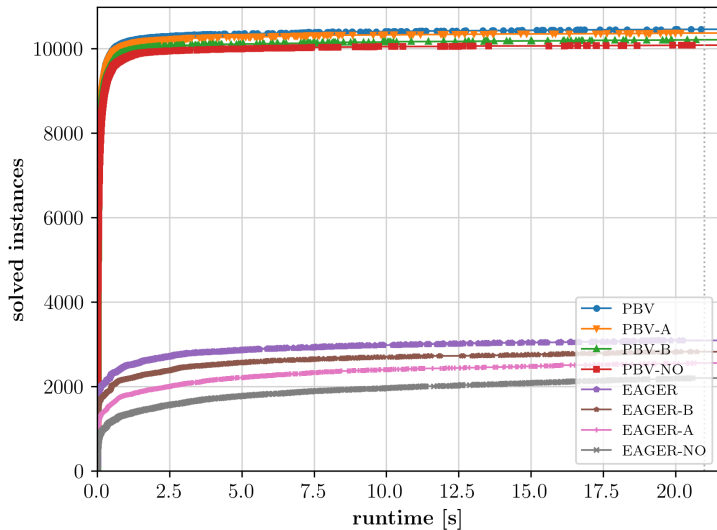
Evaluation Attributes Results

Benchmarks	#	OR-E	XOR-E	SH-M-E	ALL-E	POW2++	PIAND++	POW2-L	PIAND-L
<i>alive</i>	200	71	71	71	71	71	71	63	74
<i>ic</i>	180	44	44	48	49	43	43	55	45
<i>rewrite</i>	2006	692	653	724	758	669	633	681	797
<i>syrew</i>	1500	561	556	604	603	570	551	580	593
<i>lemmas</i>	70	12	12	13	13	12	12	17	13
<i>icfb</i>	46	1	1	1	1	1	1	1	1
<i>mut</i>	9441	692	663	774	794	669	634	732	733
<i>total</i>	13443	2071	2000	2235	2289	2033	1945	2129	2256
time-solved		15771	14863	13154	13181	15680	13947	1366	15765
mem-solved		57550	54182	51684	52451	50870	51346	12391	58938

Evaluation Lazy and Eager

Attribute	EAGER-NO	EAGER-A	EAGER-B	PBV-A	PBV-NO	PBV-B
<i>Multiple Bit-widths</i>	✓	✓	✓	✓	✓	✓
<i>Lazy pow₂</i>	✗	✗	✗	✓	✓	✓
<i>Lazy &^N</i>	✗	✗	✗	✓	✓	✓
<i> -elimination</i>	✓	✓	✓	✓	✓	✓
<i>⊕ -elimination</i>	✓	✓	✓	✓	✓	✓
<i>>> without mod</i>	✓	✓	✓	✓	✓	✓
<i>New lemmas for &^N</i>	✓	✓	✓	✓	✓	✓
<i>New lemmas for pow₂</i>	✓	✓	✓	✓	✓	✓
<i>No redundant axioms</i>	✓	✓	✓	✓	✓	✓
<i>RW_B</i>	✗	✗	✓	✗	✗	✓
<i>RW_A</i>	✗	✓	✗	✓	✗	✗

Evaluation of Rewriters



Evaluation Lazy and Eager

Benchmarks	#	EAGER-NO	EAGER-A	EAGER-B	PBV-A	PBV-NO	PBV-B	VBS
<i>alive</i>	200	87	95	85	105	97	100	125
<i>ic</i>	180	58	59	59	75	75	77	83
<i>rewrite</i>	2006	732	925	1068	1296	1148	1189	1381
<i>syrew</i>	1500	608	721	606	911	796	798	956
<i>lemmas</i>	70	14	14	14	23	23	23	25
<i>icfb</i>	46	9	9	9	11	11	12	12
<i>mut</i>	9441	885	902	1088	4832	4840	4881	5138
<i>total</i>	13443	2393	2725	2929	7253	6990	7080	7720
<i>sat</i>		0	0	0	3652	3657	3651	3841
<i>unsat</i>		2393	2725	2929	3601	3333	3429	3879
time-solved		14829	12738	10352	9805	3512	3511	5161
mem-solved		48953	44266	36934	36417	26244	25108	32677