Bit-precise Reasoning with Parametric Bit-vectors

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

[1]Bar Ilan University

[2]Stanford University

[3]The University of Iowa

SAT 2025

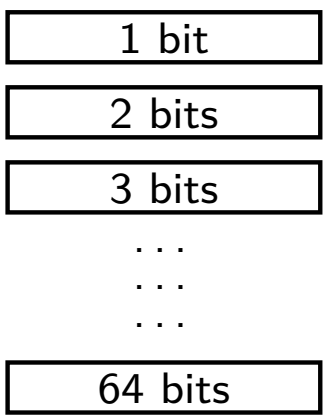
Outline

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

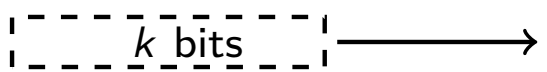
Motivation: Parametric Bit-Vectors

- 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

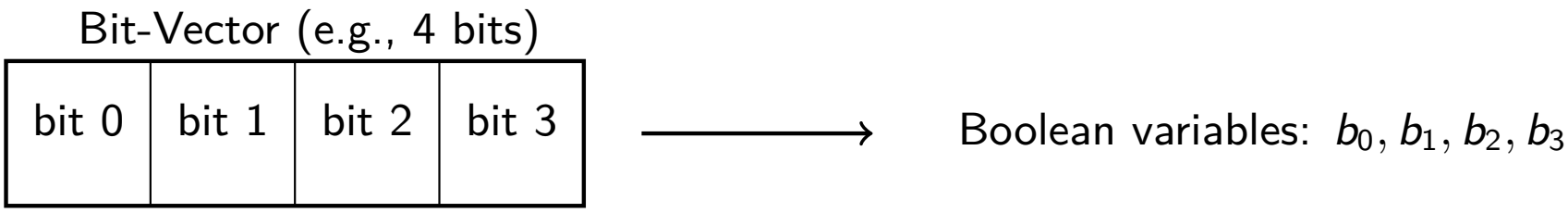


Parametric Bit-Width



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).

[1] Niemetz 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

Parametric Bit-Vectors Theory

- $T_{PBV} = (\Sigma_{PBV}, I_{PBV})$
- The set $\Sigma_{PBV} = \Sigma_{IA} +$ the following operators:

Symbol	**SMT-LIB Syntax**	**Sort**
$\approx_{\text{PBV}}, \not\approx_{\text{PBV}}$	`=, distinct`	PBV $\times$ PBV $\rightarrow$ Bool
$<_{\text{u}}, >_{\text{u}}, <_{\text{s}}, >_{\text{s}}$	`bvult, bvugt, bvslt, bvsgt`	PBV $\times$ PBV $\rightarrow$ Bool
$\leq_{\text{u}}, \geq_{\text{u}}, \leq_{\text{s}}, \geq_{\text{s}}$	`bvule, bvuge, bvsle, bvsge`	PBV $\times$ PBV $\rightarrow$ Bool
$\sim, -^{\mathcal{B}}$	`bvnot, bvneg`	PBV $\rightarrow$ PBV
$\&, \mid, \oplus$	`bvand, bvor, bvxor`	PBV $\times$ PBV $\rightarrow$ PBV
$\ll, \gg, \gg_a$	`bvshl, bvlshr, bvashr`	PBV $\times$ PBV $\rightarrow$ PBV
$+^{\mathcal{B}}, -^{\mathcal{B}}$	`bvadd, bvsub`	PBV $\times$ PBV $\rightarrow$ PBV
$\cdot^{\mathcal{B}}$, mod$^{\mathcal{B}}$, div$^{\mathcal{B}}$	`bvmul, bvurem, bvudiv`	PBV $\times$ PBV $\rightarrow$ PBV
$\llcorner\llcorner : \llcorner\rrcorner$	`pextract`	PBV $\times$ Int $\times$ Int $\rightarrow$ PBV
$\circ$	`concat`	PBV $\times$ PBV $\rightarrow$ PBV
ext_z	`pzero_extend`	Int $\times$ PBV $\rightarrow$ PBV
ext_s	`psign_extend`	Int $\times$ PBV $\rightarrow$ PBV
$\lvert\llcorner\rvert$	`bvsize`	PBV $\rightarrow$ Int
to-pbv	`int_to_pbv`	Int $\times$ Int $\rightarrow$ PBV

Parametric Bit-Vectors Interpretation

- $T_{PBV} = (\Sigma_{PBV}, I_{PBV})$
- I_{PBV}:
  - `Int` domain: all integers.
  - `PBV` domain: all bit-vectors.
  - Integer operators: as usual.
  - Bit-vectors operators: same as in T_{BV} if bit-widths match; arbitrary otherwise.
  - $|010| = 3$.
  - $\text{to-pbv}(3, 2) = 010$.
- $\mathcal{A}$ is a T_{PBV}-interpretation if it satisfies all requirements defined by I_{PBV}.

Satisfiability

- φ is T_{PBV}-satisfiable if there exists a T_{PBV}-interpretation that satisfies φ.
- Not enough!

Example

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

A T_{PBV}-interpretation:

- $x \mapsto 0$
- $x \circ x \mapsto 00$
- $x +^{\mathcal{B}} (x \circ x) \mapsto 0$ (different widths yield arbitrary results)

Therefore, φ is T_{PBV}-satisfiable.

Bw Function

function $\mathrm{Bw}(e)$
  match e:

x	$\rightarrow$	$\lvert x \rvert$
$\text{to-pbv}(k, t)$	$\rightarrow$	k
$t[i : j]$	$\rightarrow$	$i - j + 1$
$ext_z(n, t)$	$\rightarrow$	$\mathrm{Bw}(t) + n$
$ext_s(n, t)$	$\rightarrow$	$\mathrm{Bw}(t) + n$
$t_1 \circ t_2$	$\rightarrow$	$\mathrm{Bw}(t_1) + \mathrm{Bw}(t_2)$
$\diamond(t_1, \ldots, t_n)$	$\rightarrow$	$\mathrm{Bw}(t_1)$

end function

Bw Function

function $\mathrm{Bw}(e)$
  match e:

x	$\to$	$\lvert x \rvert$
$\text{to-pbv}(k, t)$	$\to$	k
$t[i : j]$	$\to$	$i - j + 1$
$ext_z(n, t)$	$\to$	$\mathrm{Bw}(t) + n$
$ext_s(n, t)$	$\to$	$\mathrm{Bw}(t) + n$
$t_1 \circ t_2$	$\to$	$\mathrm{Bw}(t_1) + \mathrm{Bw}(t_2)$
$\diamond(t_1, \ldots, t_n)$	$\to$	$\mathrm{Bw}(t_1)$

end function

Bw Function

function $\mathrm{Bw}(e)$

match e:

x	$\rightarrow$	$\lvert x \rvert$
$\text{to-pbv}(k, t)$	$\rightarrow$	k
$t[i:j]$	$\rightarrow$	$i - j + 1$
$ext_z(n, t)$	$\rightarrow$	$\mathrm{Bw}(t) + n$
$ext_s(n, t)$	$\rightarrow$	$\mathrm{Bw}(t) + n$
$t_1 \circ t_2$	$\rightarrow$	$\mathrm{Bw}(t_1) + \mathrm{Bw}(t_2)$
$\diamond(t_1, \ldots, t_n)$	$\rightarrow$	$\mathrm{Bw}(t_1)$

end function

Adm Function

function $\textsc{Adm}(e)$
  match e:

x	$\rightarrow$	$\textsc{Bw}(e) > 0$
z	$\rightarrow$	$\top$
$\text{to-pbv}(k, t)$	$\rightarrow$	$\textsc{Bw}(e) > 0 \wedge \textsc{Adm}(t)$
$\lvert t \rvert$	$\rightarrow$	$\textsc{Adm}(t)$
$t[i : j]$	$\rightarrow$	$0 \leq j \leq i < \textsc{Bw}(t) \wedge \textsc{Adm}(t)$
$ext_z(n, t)$	$\rightarrow$	$n \geq 0 \wedge \textsc{Adm}(t)$
$ext_s(n, t)$	$\rightarrow$	$n \geq 0 \wedge \textsc{Adm}(t)$
$t_1 \circ t_2$	$\rightarrow$	$\textsc{Adm}(t_1) \wedge \textsc{Adm}(t_2)$
$\bullet(t_1, \ldots, t_n)$	$\rightarrow$	$\bigwedge_{i=1}^{n} \textsc{Adm}(t_i)$
$\diamond(t_1, \ldots, t_n)$	$\rightarrow$	$(\bigwedge_{i=2}^{n} \textsc{Bw}(t_1) \approx \textsc{Bw}(t_i)) \wedge (\bigwedge_{i=1}^{n} \textsc{Adm}(t_i))$

end function

- $\bullet$ Used for Σ_{IA} symbols and Boolean connectives.
- $\diamond$ Used for all other symbols not explicitly handled.

Adm Function

function $\textsc{Adm}(e)$
  match e:

x	$\rightarrow$	$\textsc{Bw}(e) > 0$
z	$\rightarrow$	$\top$
$\text{to-pbv}(k, t)$	$\rightarrow$	$\textsc{Bw}(e) > 0 \wedge \textsc{Adm}(t)$
$\lvert t \rvert$	$\rightarrow$	$\textsc{Adm}(t)$
$t[i : j]$	$\rightarrow$	$0 \leq j \leq i < \textsc{Bw}(t) \wedge \textsc{Adm}(t)$
$ext_z(n, t)$	$\rightarrow$	$n \geq 0 \wedge \textsc{Adm}(t)$
$ext_s(n, t)$	$\rightarrow$	$n \geq 0 \wedge \textsc{Adm}(t)$
$t_1 \circ t_2$	$\rightarrow$	$\textsc{Adm}(t_1) \wedge \textsc{Adm}(t_2)$
$\bullet(t_1, \ldots, t_n)$	$\rightarrow$	$\bigwedge_{i=1}^{n} \textsc{Adm}(t_i)$
$\diamond(t_1, \ldots, t_n)$	$\rightarrow$	$(\bigwedge_{i=2}^{n} \textsc{Bw}(t_1) \approx \textsc{Bw}(t_i)) \wedge (\bigwedge_{i=1}^{n} \textsc{Adm}(t_i))$

end function

- $\bullet$ Used for Σ_{IA} symbols and Boolean connectives.
- $\diamond$ Used for all other symbols not explicitly handled.

ADM Function

function $\textsc{Adm}(e)$
  match e:

x	$\rightarrow$	$\textsc{Bw}(e) > 0$
z	$\rightarrow$	$\top$
$\text{to-pbv}(k, t)$	$\rightarrow$	$\textsc{Bw}(e) > 0 \wedge \textsc{Adm}(t)$
$\lvert t \rvert$	$\rightarrow$	$\textsc{Adm}(t)$
$t[i:j]$	$\rightarrow$	$0 \leq j \leq i < \textsc{Bw}(t) \wedge \textsc{Adm}(t)$
$ext_z(n, t)$	$\rightarrow$	$n \geq 0 \wedge \textsc{Adm}(t)$
$ext_s(n, t)$	$\rightarrow$	$n \geq 0 \wedge \textsc{Adm}(t)$
$t_1 \circ t_2$	$\rightarrow$	$\textsc{Adm}(t_1) \wedge \textsc{Adm}(t_2)$
$\bullet(t_1, \ldots, t_n)$	$\rightarrow$	$\bigwedge_{i=1}^{n} \textsc{Adm}(t_i)$
$\diamond(t_1, \ldots, t_n)$	$\rightarrow$	$(\bigwedge_{i=2}^{n} \textsc{Bw}(t_1) \approx \textsc{Bw}(t_i)) \wedge (\bigwedge_{i=1}^{n} \textsc{Adm}(t_i))$

end function

- $\bullet$ Used for Σ_{IA} symbols and Boolean connectives.
- $\diamond$ Used for all other symbols not explicitly handled.

Satisfiability and Admissible Satisfiability

- φ is T_{PBV}-satisfiable if there exists a T_{PBV}-interpretation that satisfies φ.
- φ is admissibly T_{PBV}-satisfiable if there exists a T_{PBV}-interpretation that satisfies $\varphi \wedge \textsc{Adm}(\varphi)$.

Admissible Satisfiability – Examples

- φ is admissibly T_{PBV}-satisfiable if there exists a T_{PBV}-interpretation that satisfies $\varphi \wedge \textsc{Adm}(\varphi)$.

Example

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

$\textsc{Adm}(\varphi)$ includes the following constraints:

$$|x| > 0 \wedge |x| \approx |x| + |x|$$

Therefore, φ is not admissibly T_{PBV}-satisfiable.

Theory Summary

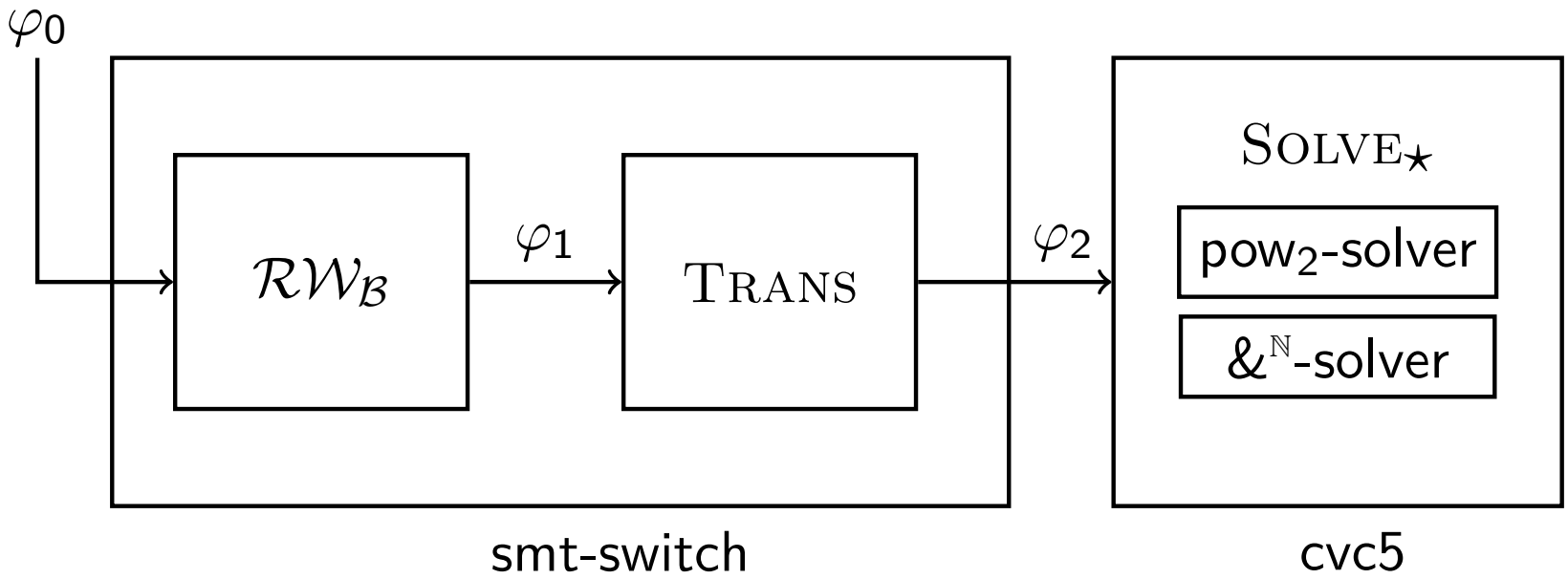
- T_{BV} cannot handle parametric bit-widths.
- T_{PBV} satisfiability alone is not sufficient.
- Admissible T_{PBV}-satisfiability is therefore used.
- Improved theory over [CADE'19]:
  - More expressive: supports bit-width reasoning.
  - Simpler and clearer.

Outline

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

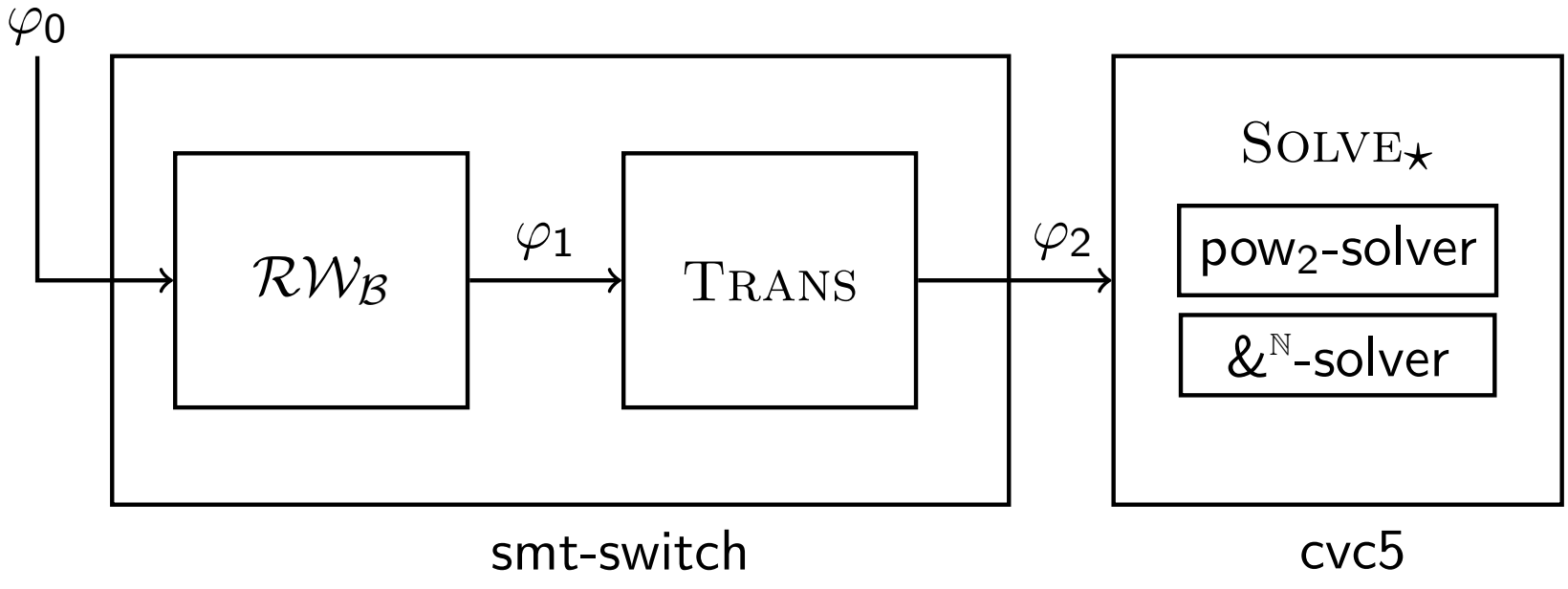
PBV Solver - Architecture

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



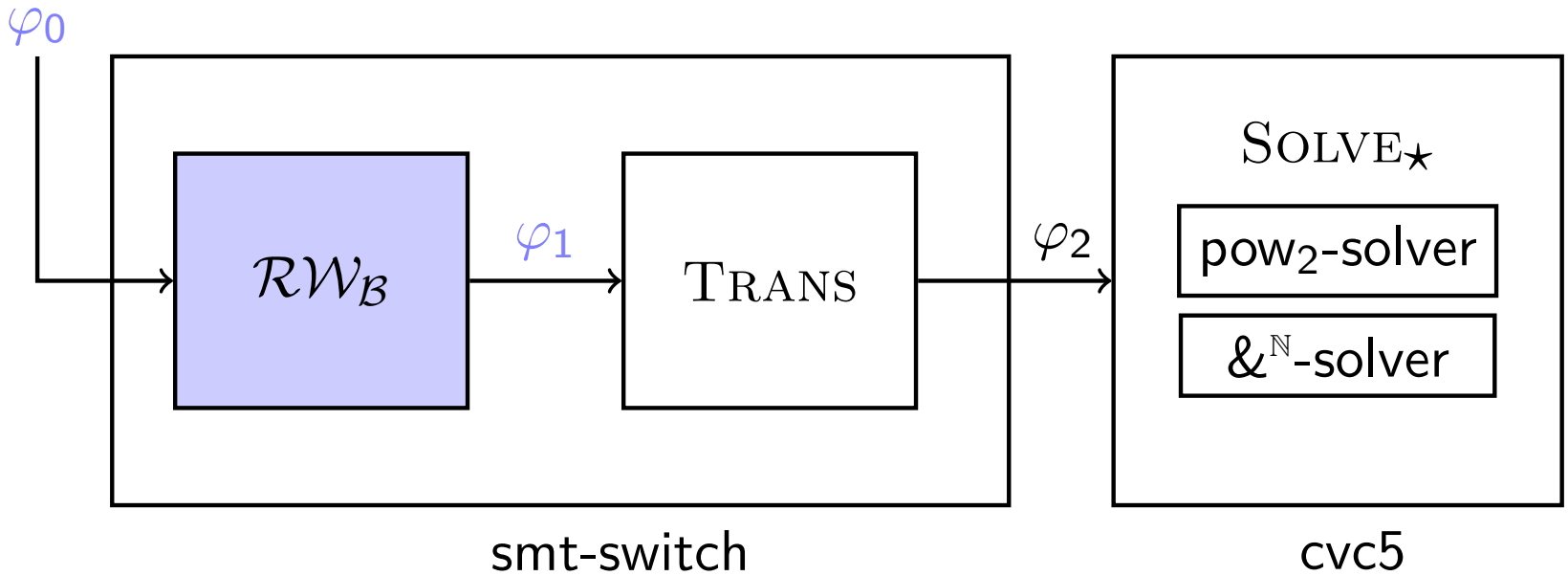
PBV Solver - Architecture

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

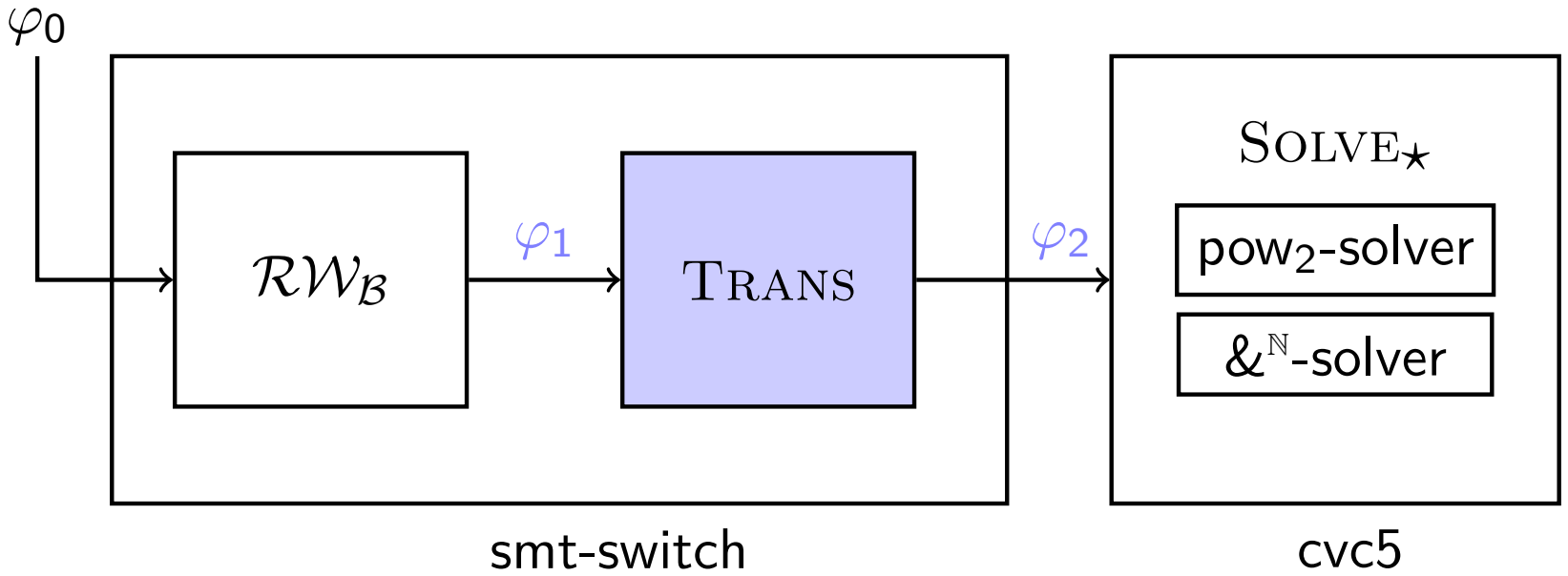


smt-switch is an open-source API for SMT solving.

PBV Solver - Architecture

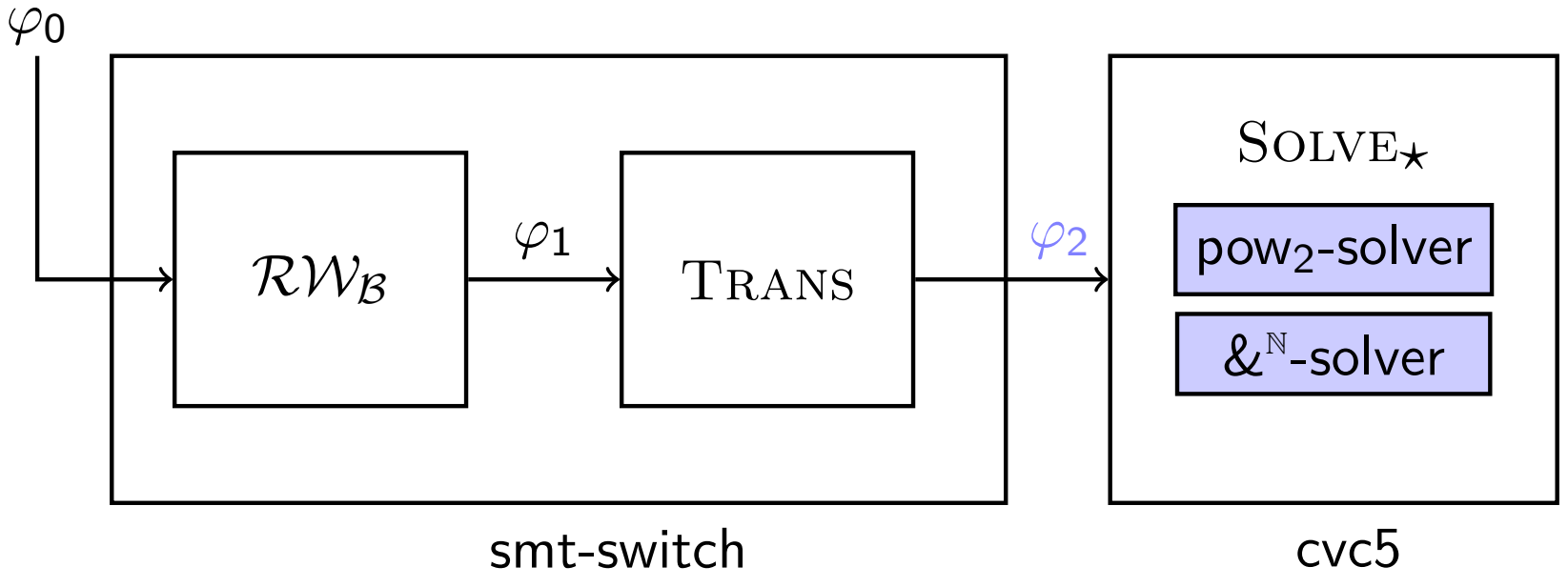


PBV Solver - Architecture



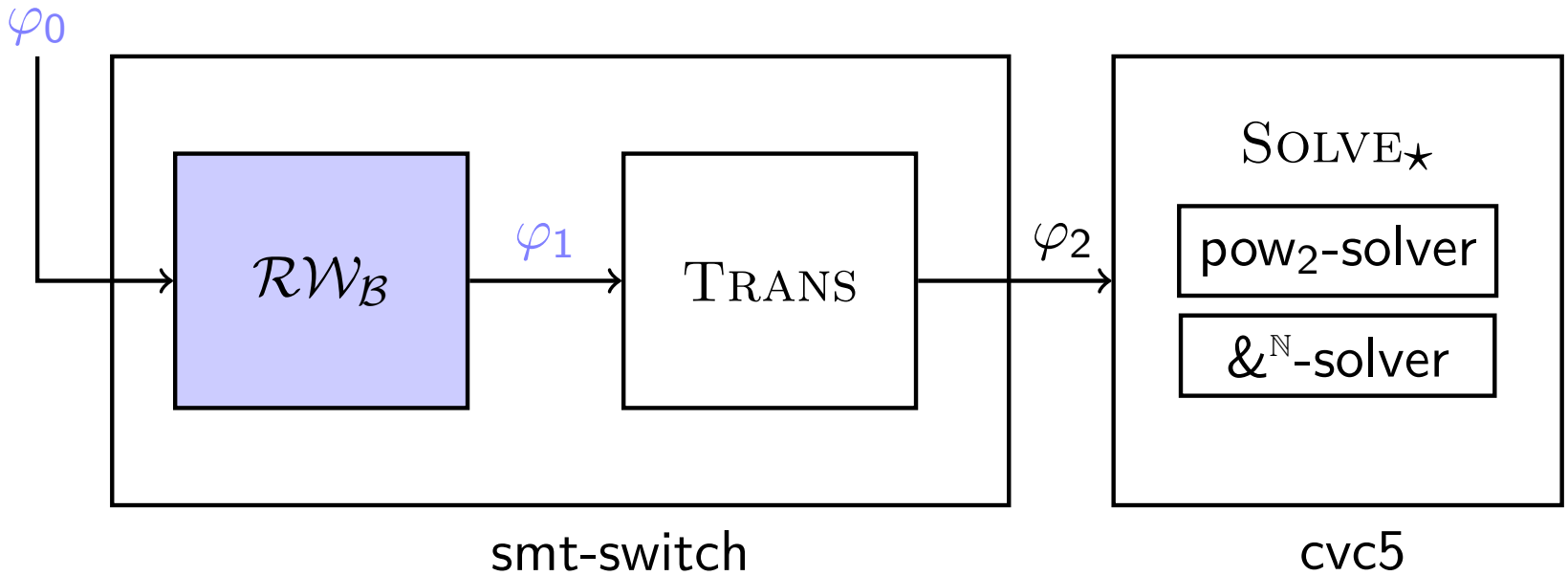
PBV Solver - Architecture

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

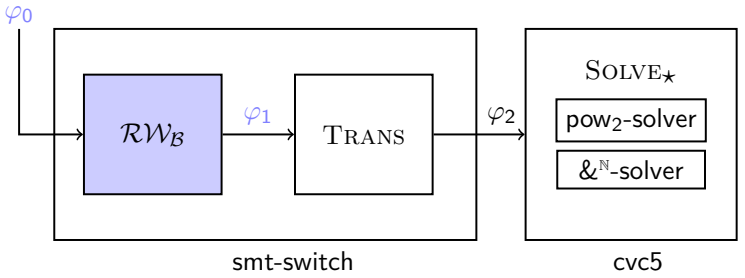


cvc5 is an open-source SMT solver.

PBV Solver - $\mathcal{RW}_{\mathcal{B}}$



Parametric Bit Vector Rewriter – $\mathcal{RW}_{\mathcal{B}}$



- Implemented 47 rewrite rules useful for the T_{PBV} theory.

Example

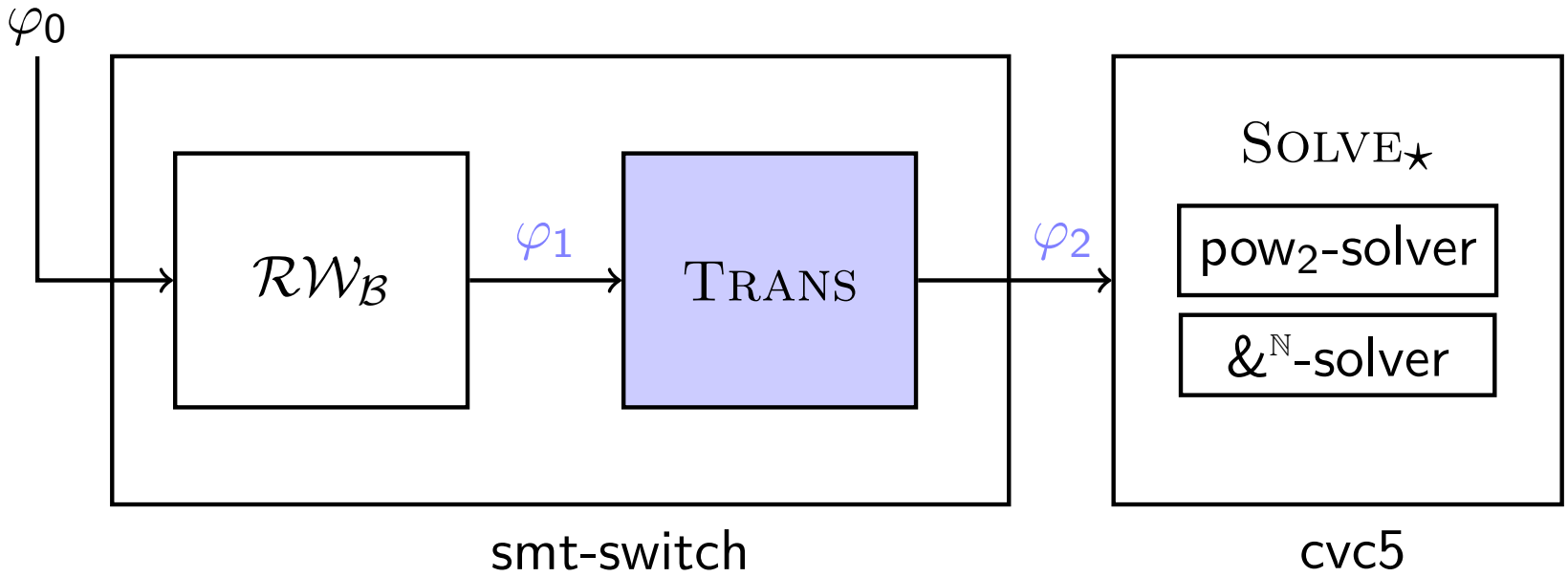
`bv-add-zero:` $(\texttt{bvadd}\ x\ 0) \mapsto x$

- In some cases, we implement their reverse.

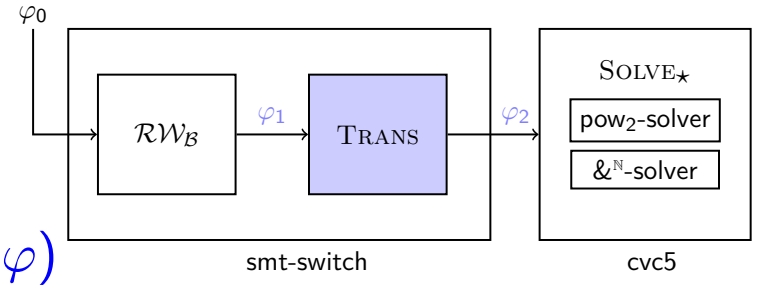
Example

`bv-reverse-extract-and:`
$(\texttt{bvand}\ (\texttt{extract}\ j\ i\ x)\ (\texttt{extract}\ j\ i\ y)) \mapsto (\texttt{extract}\ j\ i\ (\texttt{bvand}\ x\ y))$

PBV Solver - Trans



Translation Function Trans



function Trans(φ)
  return Conv(φ) $\wedge$ Range(φ) $\wedge$ Adm(φ)
end function

function Range(e)
  match e:

x	$\to 0 \leq \chi(x) < \mathsf{pow}_2(\kappa(x))$
$\lvert t \rvert$	$\to e \approx \mathrm{Bw}(t) \wedge \mathrm{Range}(t)$
z	$\to \top$
$\diamond(t_1, \ldots, t_n)$	$\to \bigwedge_{i=1}^{n} \mathrm{Range}(t_i)$

end function

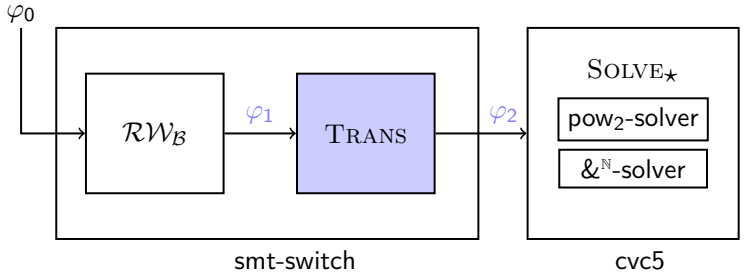
- Based on [CADE'19].
- Our changes are highlighted in blue.

Translation Function Conv

function $\textsc{Conv}(e)$
  match e:

$$
\begin{array}{ll}
z & \to z \\
|t| & \to \kappa(t) \\
\text{to-pbv}(k,t) & \to \textsc{Conv}(t) \bmod \text{pow}_2(k) \\
x & \to \chi(x) \\
t_1 \approx t_2 & \to \textsc{Conv}(t_1) \approx \textsc{Conv}(t_2) \\
\bowtie_u (t_1, t_2) & \to \textsc{Conv}(t_1) \bowtie \textsc{Conv}(t_2) \\
\bowtie_s (t_1, t_2) & \to \text{uts}(\kappa(t_1), \textsc{Conv}(t_1)) \bowtie \text{uts}(\kappa(t_2), \textsc{Conv}(t_2)) \\
t_1 +^{\mathcal{B}} t_2 & \to (\textsc{Conv}(t_1) + \textsc{Conv}(t_2)) \bmod \text{pow}_2(\kappa(t_1)) \\
t_1 -^{\mathcal{B}} t_2 & \to (\textsc{Conv}(t_1) - \textsc{Conv}(t_2)) \bmod \text{pow}_2(\kappa(t_1)) \\
t_1 \cdot^{\mathcal{B}} t_2 & \to (\textsc{Conv}(t_1) \cdot \textsc{Conv}(t_2)) \bmod \text{pow}_2(\kappa(t_1)) \\
t_1 \,\text{div}^{\mathcal{B}}\, t_2 & \to \mathit{ite}(\textsc{Conv}(t_2) \approx 0, \text{pow}_2(\kappa(t_1)) - 1, \textsc{Conv}(t_1) \text{ div } \textsc{Conv}(t_2)) \\
t_1 \,\text{mod}^{\mathcal{B}}\, t_2 & \to \mathit{ite}(\textsc{Conv}(t_2) \approx 0, \textsc{Conv}(t_1), \textsc{Conv}(t_1) \bmod \textsc{Conv}(t_2)) \\
\sim t & \to \text{pow}_2(\kappa(t)) - (\textsc{Conv}(t) + 1) \\
-^{\mathcal{B}} t & \to (\text{pow}_2(\kappa(t)) - \textsc{Conv}(t)) \bmod \text{pow}_2(\kappa(t)) \\
t_1 \ll t_2 & \to (\textsc{Conv}(t_1) \cdot \text{pow}_2(\textsc{Conv}(t_2))) \bmod \text{pow}_2(\kappa(t_1)) \\
t_1 \gg t_2 & \to \textsc{Conv}(t_1) \text{ div } \text{pow}_2(\textsc{Conv}(t_2)) \\
t_1 \,\&\, t_2 & \to \&^{\mathbb{N}}(\kappa(t_1), \textsc{Conv}(t_1), \textsc{Conv}(t_2)) \\
t_1 \mid t_2 & \to \textsc{Conv}(t_1) + \textsc{Conv}(t_2) - \&^{\mathbb{N}}(\kappa(t_1), \textsc{Conv}(t_1), \textsc{Conv}(t_2)) \\
t_1 \oplus t_2 & \to \textsc{Conv}(t_1) + \textsc{Conv}(t_2) - 2 \cdot \&^{\mathbb{N}}(\kappa(t_1), \textsc{Conv}(t_1), \textsc{Conv}(t_2)) \\
t_1 \circ t_2 & \to \textsc{Conv}(t_1) \cdot \text{pow}_2(\kappa(t_2)) + \textsc{Conv}(t_2) \\
t[i:j] & \to (\textsc{Conv}(t) \text{ div } \text{pow}_2(j)) \bmod \text{pow}_2(i - j + 1) \\
\mathit{ext}_z(n,t) & \to \textsc{Conv}(t) \\
\mathit{ext}_s(n,t) & \to \mathit{ite}(\textsc{Conv}(t[\kappa(t) - 1]) \approx 1, (\text{pow}_2(n) - 1) \cdot \text{pow}_2(\kappa(t)) + \textsc{Conv}(t), \textsc{Conv}(t)) \\
\bullet(t_1, \ldots, t_n) & \to \bullet(\textsc{Conv}(t_1), \ldots, \textsc{Conv}(t_n))
\end{array}
$$

end function

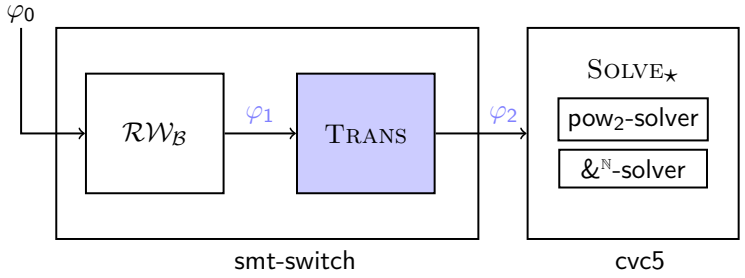


Translation Function Conv

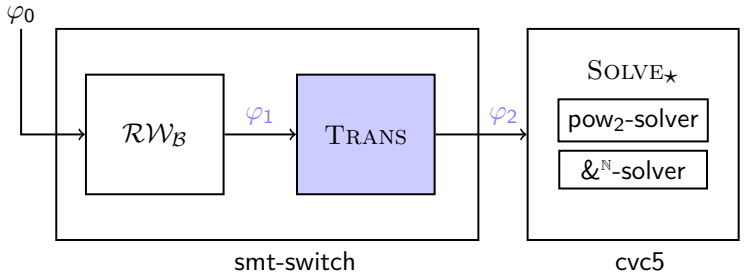
function $\textsc{Conv}(e)$
match e:

z	$\to z$
$\lvert t\rvert$	$\to \kappa(t)$
$\text{to-pbv}(k, t)$	$\to \textsc{Conv}(t) \bmod \text{pow}_2(k)$
x	$\to \chi(x)$
$t_1 \approx t_2$	$\to \textsc{Conv}(t_1) \approx \textsc{Conv}(t_2)$
$\bowtie_u (t_1, t_2)$	$\to \textsc{Conv}(t_1) \bowtie \textsc{Conv}(t_2)$
$\bowtie_s (t_1, t_2)$	$\to \text{uts}(\kappa(t_1), \textsc{Conv}(t_1)) \bowtie \text{uts}(\kappa(t_2), \textsc{Conv}(t_2))$
$t_1 +^{\mathcal{B}} t_2$	$\to (\textsc{Conv}(t_1) + \textsc{Conv}(t_2)) \bmod \text{pow}_2(\kappa(t_1))$
$t_1 \text{div}^{\mathcal{B}} t_2$	$\to ite(\textsc{Conv}(t_2) \approx 0, \text{pow}_2(\kappa(t_1)) - 1, \textsc{Conv}(t_1) \text{ div } \textsc{Conv}(t_2))$
$t_1 \bmod^{\mathcal{B}} t_2$	$\to ite(\textsc{Conv}(t_2) \approx 0, \textsc{Conv}(t_1), \textsc{Conv}(t_1) \bmod \textsc{Conv}(t_2))$
$\sim t$	$\to \text{pow}_2(\kappa(t)) - (\textsc{Conv}(t) + 1)$
$-^{\mathcal{B}} t$	$\to (\text{pow}_2(\kappa(t)) - \textsc{Conv}(t)) \bmod \text{pow}_2(\kappa(t))$
$t_1 \ll t_2$	$\to (\textsc{Conv}(t_1) \cdot \text{pow}_2(\textsc{Conv}(t_2))) \bmod \text{pow}_2(\kappa(t_1))$
$t_1 \gg t_2$	$\to \textsc{Conv}(t_1) \text{ div } \text{pow}_2(\textsc{Conv}(t_2))$
$t_1 \,\&\, t_2$	$\to \&^{\mathbb{N}}(\kappa(t_1), \textsc{Conv}(t_1), \textsc{Conv}(t_2))$
$t_1 \mid t_2$	$\to \textsc{Conv}(t_1) + \textsc{Conv}(t_2) - \&^{\mathbb{N}}(\kappa(t_1), \textsc{Conv}(t_1), \textsc{Conv}(t_2))$
$t_1 \oplus t_2$	$\to \textsc{Conv}(t_1) + \textsc{Conv}(t_2) - 2 \cdot \&^{\mathbb{N}}(\kappa(t_1), \textsc{Conv}(t_1), \textsc{Conv}(t_2))$
$t_1 \circ t_2$	$\to \textsc{Conv}(t_1) \cdot \text{pow}_2(\kappa(t_2)) + \textsc{Conv}(t_2)$
$t[i:j]$	$\to (\textsc{Conv}(t) \text{ div } \text{pow}_2(j)) \bmod \text{pow}_2(i - j + 1)$
$ext_z(n, t)$	$\to \textsc{Conv}(t)$
$ext_s(n, t)$	$\to ite(\textsc{Conv}(t[\kappa(t) - 1]) \approx 1, (\text{pow}_2(n) - 1) \cdot \text{pow}_2(\kappa(t)) + \textsc{Conv}(t), \textsc{Conv}(t))$
$\bullet(t_1, \ldots, t_n)$	$\to \bullet(\textsc{Conv}(t_1), \ldots, \textsc{Conv}(t_n))$

end function

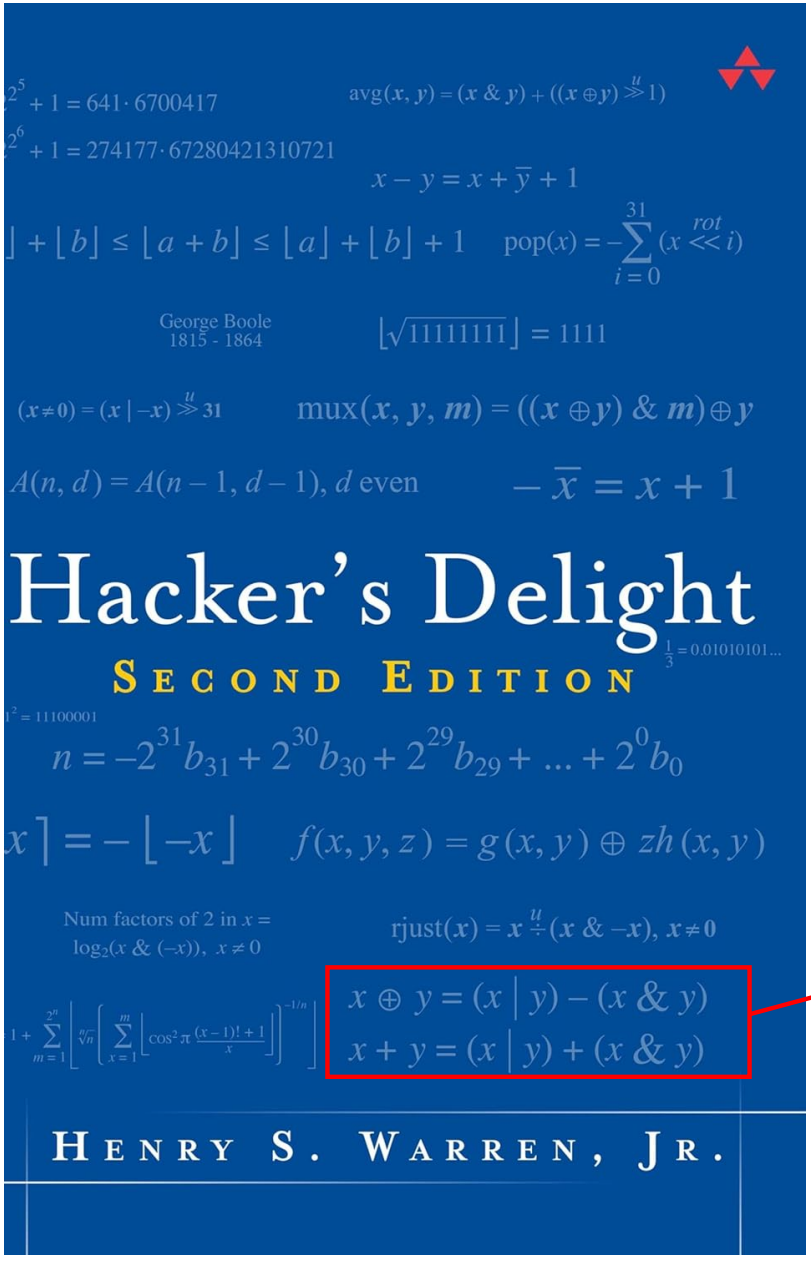


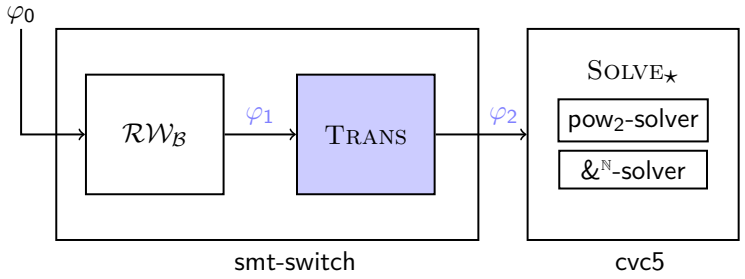
Translation Function Conv- New Operators



- Operators newly supported in our approach:
  - $|t|$
  - $\text{to-pbv}(k, t)$
  - $t[i : j]$
  - $ext_z(n, t)$
  - $ext_s(n, t)$

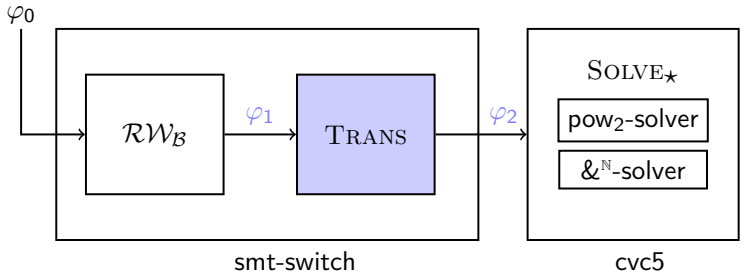
Improved Bitwise-Or Translation





$$x \oplus y = (x \mid y) - (x \,\&\, y)$$
$$x + y = (x \mid y) + (x \,\&\, y)$$

Improved Bitwise-Or Translation



Hacker's Delight:

$$t_1 \mid t_2 \quad \mapsto \quad (t_1 + t_2) - (t_1 \,\&\, t_2)$$

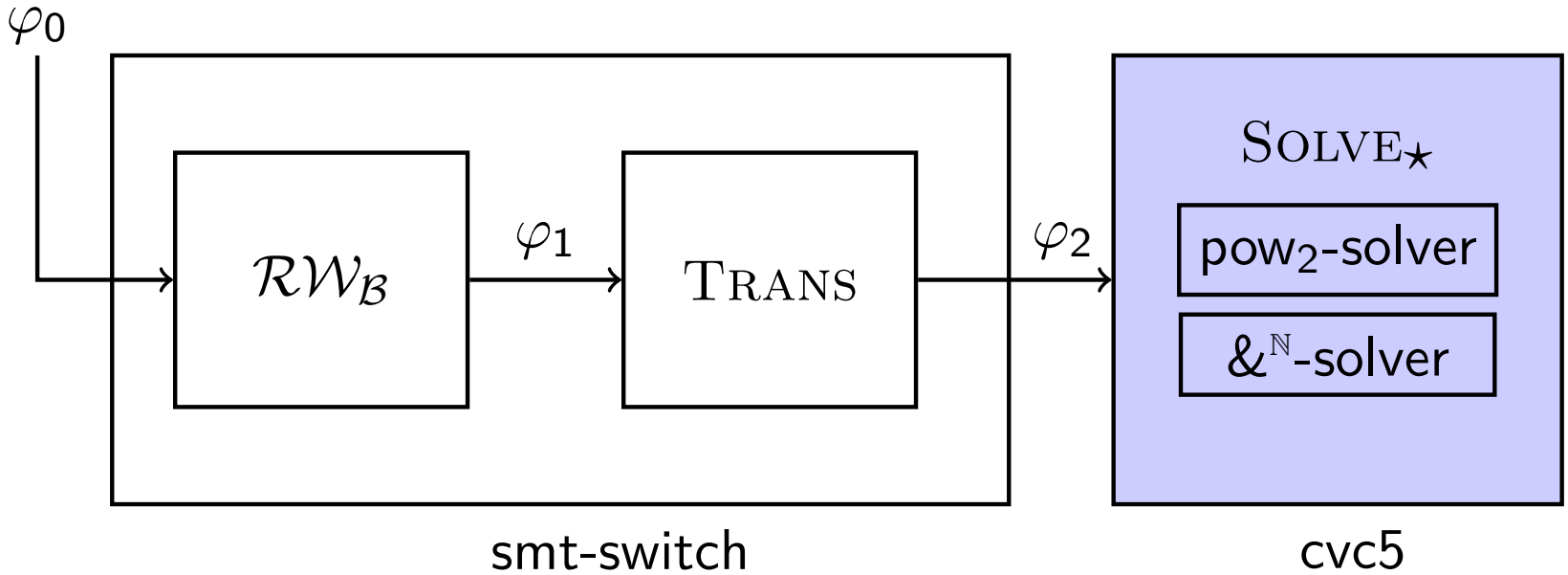
Translation to integers:

$$\begin{aligned} t_1 \mid t_2 \quad \mapsto \quad & (((\textsc{Conv}(t_1) + \textsc{Conv}(t_2)) \bmod \mathsf{pow}_2(\kappa(t_1))) - \\ & \&^{\mathbb{N}}(\kappa(t_1), \textsc{Conv}(t_1), \textsc{Conv}(t_2))) \bmod \mathsf{pow}_2(\kappa(t_1)) \end{aligned}$$

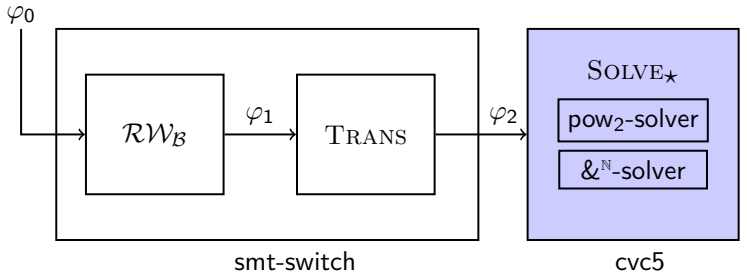
Improvement:

$$t_1 \mid t_2 \quad \mapsto \quad \textsc{Conv}(t_1) + \textsc{Conv}(t_2) - \&^{\mathbb{N}}(\kappa(t_1), \textsc{Conv}(t_1), \textsc{Conv}(t_2))$$

PBV Solver - $\textsc{Solve}_\star$



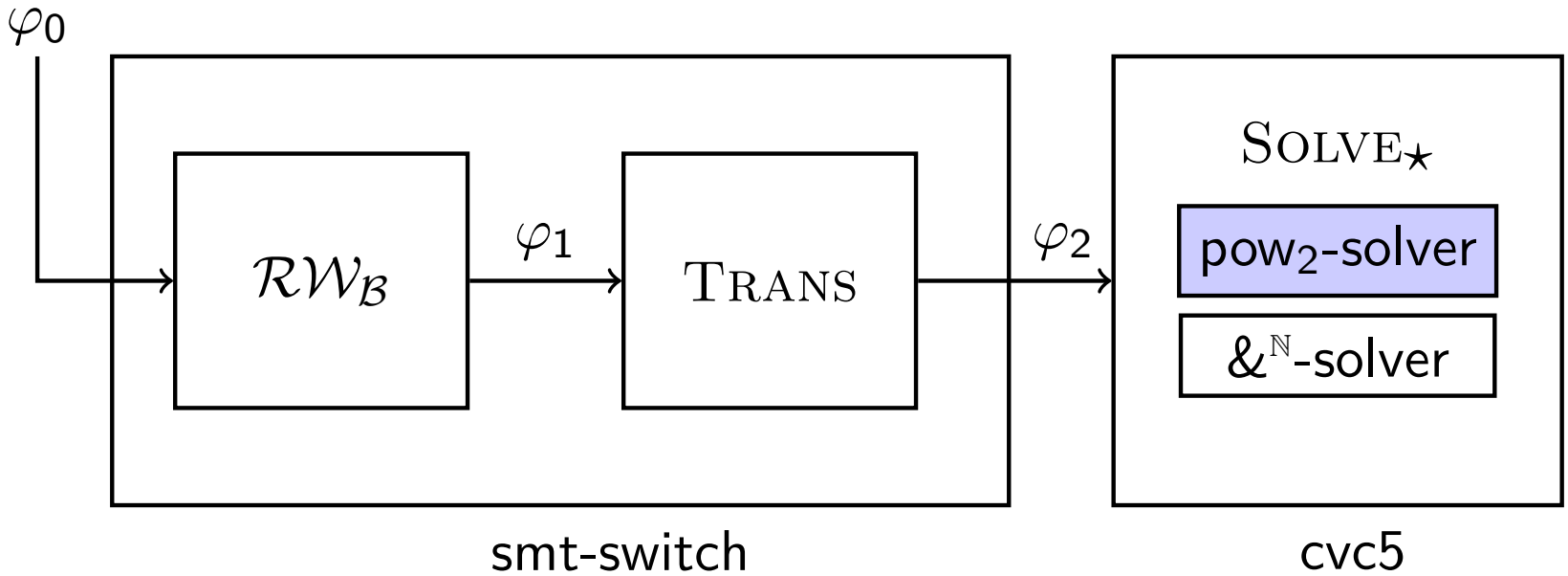
Lazy vs. Eager Algorithm



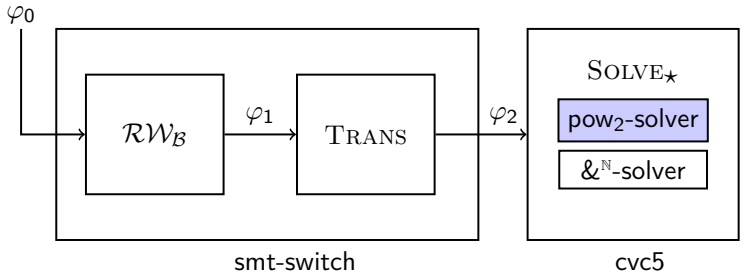
- Eager: adds all axioms upfront.
- Lazy: adds lemmas only when needed.
- Lazy algorithm uses CEGAR[2] to add lemmas.
- The key is selecting the right lemmas.

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

PBV Solver - pow_2



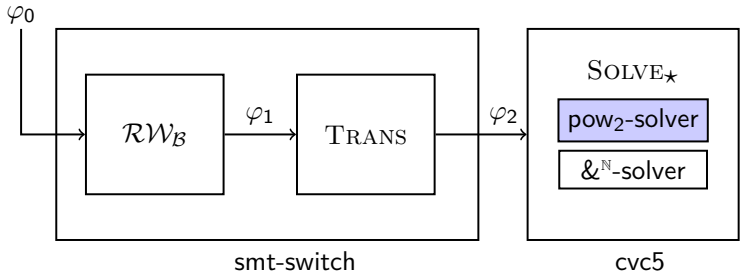
pow_2-solver



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$_\star$.

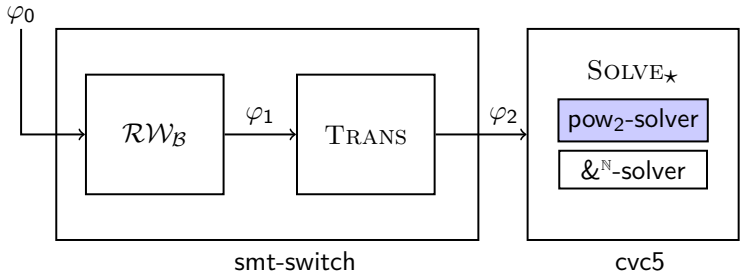
pow_2-solver



Name	Lemma	Source
	$\text{pow}_2(x) \in S$	
positive	$0 \leq x \Rightarrow \text{pow}_2(x) > 0$	[CADE'19]
bound:	$(x \geq v \wedge v \geq 7) \Rightarrow v \cdot x + v^2 < \text{pow}_2(x)$	
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$_\star$.

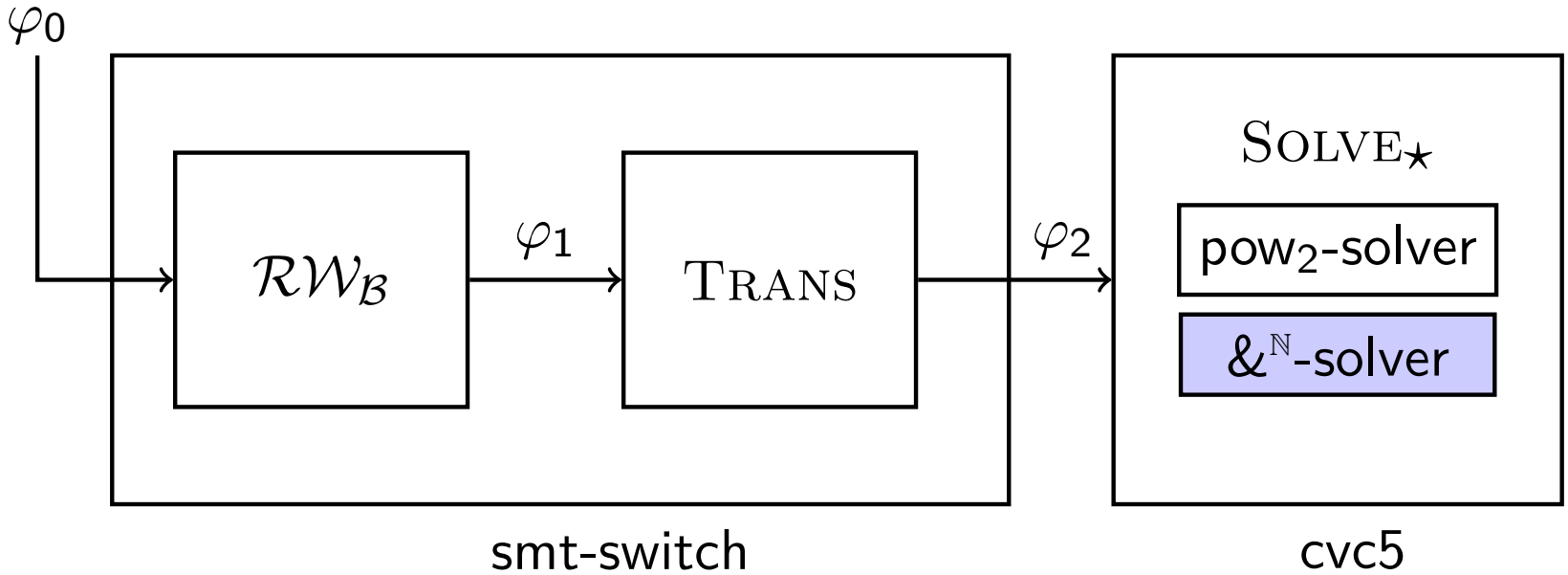
bound Lemma



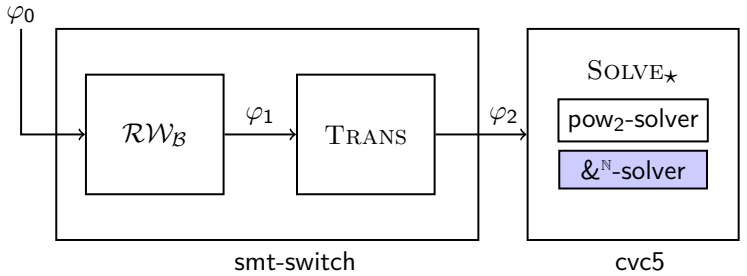
- Naive bound lemma: $x < \mathsf{pow}_2(x)$.
- More efficient from [IJCAR'24][3]:
$$x \geq 3 \Rightarrow 2 \cdot x + 1 < \mathsf{pow}_2(x).$$
- Tighter bound:
$$x \geq 7 \Rightarrow 2 \cdot x^2 < \mathsf{pow}_2(x).$$
- Using v to obtain a linear bound:
$$(x \geq v \wedge v \geq 7) \Rightarrow v \cdot x + v^2 < \mathsf{pow}_2(x).$$

[3] Frohn Florian, Jürgen Giesl, Satisfiability Modulo Exponential Integer Arithmetic, IJCAR'24.

PBV Solver - $\&^{\mathbb{N}}$

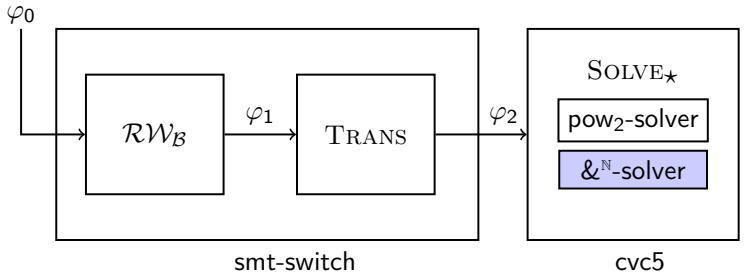


$\&^{\mathbb{N}}$-solver



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

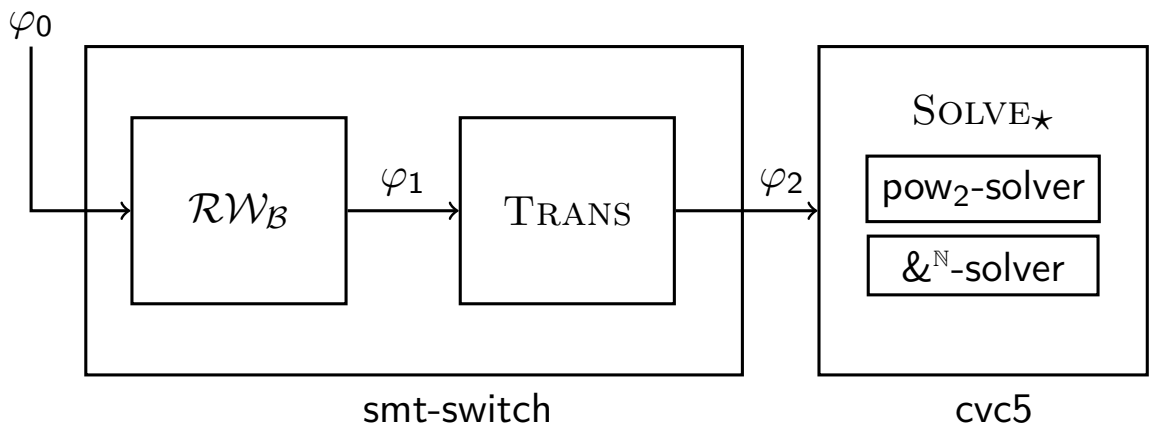
$\&^{\mathbb{N}}$-solver



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

Algorithm Summary

- Rewriting for Parametric Bit-Vectors.
- Efficient translation to integers.
- Lazy algorithm with new lemmas.



Outline

1 Introduction

2 Theory

3 Solver

4 Evaluation

Benchmark Sets

Compiler Optimizations

- *alive* (200 benchmarks)[4]

Crafted benchmarks

- *rewrite* (2006 benchmarks)[4]
- *syrew* (1500 benchmarks)[5]
  - enumerated by SyGuS (cvc5) for w = 4

.

SMT solvers internals

- *lemmas* (70 benchmarks)[5]
- *ic* (180 benchmarks)[6]
- *icfb* (46 benchmarks)[7]
  - quantified benchmarks

Mutated Benchmarks

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

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

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

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

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

Evaluation Configuration

Attribute	BASELINE	EAGER	PBV
Multiple Bit-widths	✗	✓	✓
Lazy pow_2	✗	✗	✓
Lazy $\&^{\mathbb{N}}$	✗	✗	✓
$\mid$*-elimination*	✗	✓	✓
$\oplus$*-elimination*	✗	✓	✓
$\gg$ *without* mod	✗	✓	✓
New lemmas for $\&^{\mathbb{N}}$	✗	✓	✓
New lemmas for pow_2	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓

Evaluation Configuration

Attribute	BASELINE	EAGER	PBV
Multiple Bit-widths	✗	✓	✓
Lazy pow_2	✗	✗	✓
Lazy $\&^{\mathbb{N}}$	✗	✗	✓
$\mid$*-elimination*	✗	✓	✓
$\oplus$*-elimination*	✗	✓	✓
$\gg$ *without* mod	✗	✓	✓
New lemmas for $\&^{\mathbb{N}}$	✗	✓	✓
New lemmas for pow_2	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓

Evaluation Configuration

Attribute	BASELINE	EAGER	PBV
Multiple Bit-widths	✗	✓	✓
Lazy pow_2	✗	✗	✓
Lazy $\&^{\mathbb{N}}$	✗	✗	✓
$\mid$*-elimination*	✗	✓	✓
$\oplus$*-elimination*	✗	✓	✓
$\gg$ *without* mod	✗	✓	✓
New lemmas for $\&^{\mathbb{N}}$	✗	✓	✓
New lemmas for pow_2	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓

Evaluation Configuration

Attribute	BASELINE	EAGER	PBV
Multiple Bit-widths	✗	✓	✓
Lazy pow_2	✗	✗	✓
Lazy $\&^{\mathbb{N}}$	✗	✗	✓
$\mid$*-elimination*	✗	✓	✓
$\oplus$*-elimination*	✗	✓	✓
$\gg$ *without* mod	✗	✓	✓
New lemmas for $\&^{\mathbb{N}}$	✗	✓	✓
New lemmas for pow_2	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓

Evaluation Configuration

Attribute	BASELINE	EAGER	PBV
Multiple Bit-widths	✗	✓	✓
Lazy pow_2	✗	✗	✓
Lazy $\&^{\mathbb{N}}$	✗	✗	✓
$\mid$*-elimination*	✗	✓	✓
$\oplus$*-elimination*	✗	✓	✓
$\gg$ *without* mod	✗	✓	✓
New lemmas for $\&^{\mathbb{N}}$	✗	✓	✓
New lemmas for pow_2	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓

Evaluation Configuration

Attribute	BASELINE	EAGER	PBV
Multiple Bit-widths	✗	✓	✓
Lazy pow_2	✗	✗	✓
Lazy $\&^{\mathbb{N}}$	✗	✗	✓
$\mid$*-elimination*	✗	✓	✓
$\oplus$*-elimination*	✗	✓	✓
$\gg$ *without* mod	✗	✓	✓
New lemmas for $\&^{\mathbb{N}}$	✗	✓	✓
New lemmas for pow_2	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓

Evaluation Results

Benchmarks	#	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**
mut	9441	669	1084	**4863**
total	13443	2012	3199	**7325**

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

Evaluation Results

Benchmarks	#	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**
mut	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.
- BASELINE and EAGER did not solve any satisfiable benchmarks.

Contributions on Benchmark Sets

- Improvements over [CADE'19]:
  - 45 benchmarks in *alive*.
  - 12 benchmarks in *ic*.
- First parametric verification of *lemmas* and *icfb*:
  - 23 benchmarks in *lemmas*.
  - 12 benchmarks in *icfb*.
  - Found a known typo in *icfb* from [FMCAD'20].

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 $\varphi \wedge \textsc{Adm}(\varphi)$.

Example

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

$\textsc{Adm}(\varphi)$:

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

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.

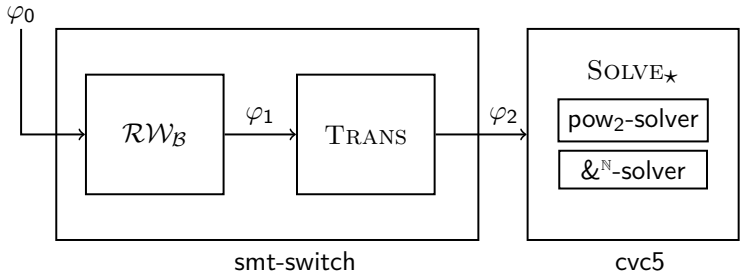
Parametric Bit Vector Rewriter – $\mathcal{RW}_{\mathcal{B}}$ (Part 1)

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

Parametric Bit Vector Rewriter – $\mathcal{RW}_{\mathcal{B}}$ (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	(bvsle 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	(bvlshr (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))

Arithmetic Rewriter – $\mathcal{RW}_{\mathcal{A}}$

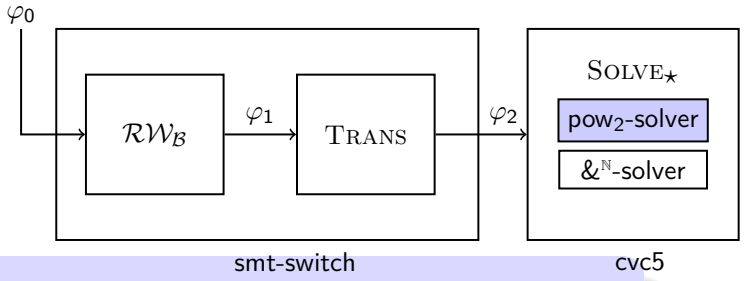


- Rewrite rules to eliminate modulo operations:

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

- cvc5 does not implement them: modulo assumed positive.

Arithmetic Rewriter – $\mathcal{RW}_{\mathcal{A}}$



Example

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

$\textsc{Conv}(\varphi)$:

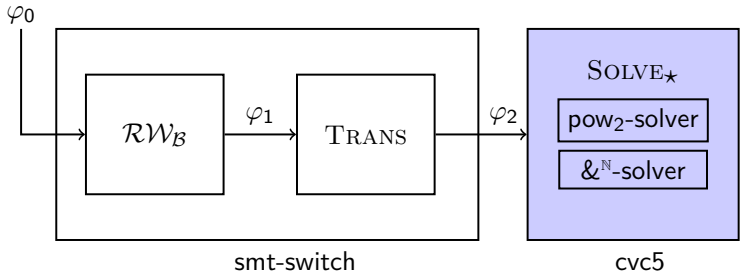
$$(((x' + x') \bmod 2^k) + x') \bmod 2^k$$

$\mathcal{RW}_{\mathcal{A}}$ ($\textsc{Conv}(\varphi)$):

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

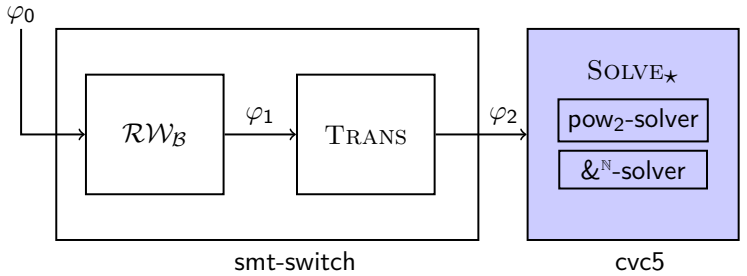
where $x' = \chi(x)$.

CEGAR Procedure



function SOLVE$_\star(\varphi)$
  $\Gamma \leftarrow \{\varphi\}$
  loop
    $result, \mathcal{I} \leftarrow$ SOLVE(Γ)
    if *result* is "unsat" **return** "unsat"
    $\mathcal{L} \leftarrow \mathcal{L}_{\&^\mathbb{N}}(\{\&^\mathbb{N}(k, t_1, t_2) \mid \&^\mathbb{N}(k, t_1, t_2) \in \mathit{Terms}(\Gamma)\}, \mathcal{I})$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{\mathsf{pow}_2}(\{\mathsf{pow}_2(t) \mid \mathsf{pow}_2(t) \in \mathit{Terms}(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$
    if $\mathcal{I} \models \mathcal{L}$ **return** "sat"
    $\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L}$ and $\mathcal{I} \not\models \psi\}$
  end loop
end function

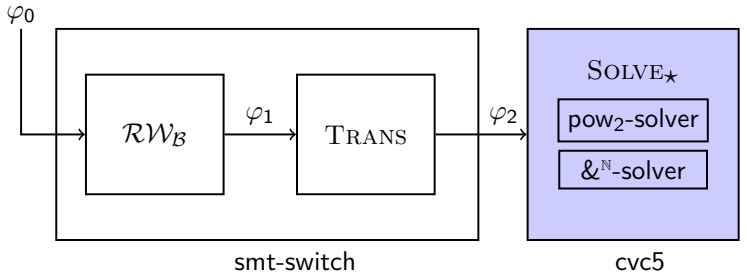
CEGAR Procedure



function SOLVE$_\star(\varphi)$
  $\Gamma \leftarrow \{\varphi\}$
  loop
    $result, \mathcal{I} \leftarrow$ SOLVE(Γ)
    if *result* is "unsat" **return** "unsat"
    $\mathcal{L} \leftarrow \mathcal{L}_{\&^\mathbb{N}}(\{\&^\mathbb{N}(k, t_1, t_2) \mid \&^\mathbb{N}(k, t_1, t_2) \in \mathit{Terms}(\Gamma)\}, \mathcal{I})$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{\mathsf{pow}_2}(\{\mathsf{pow}_2(t) \mid \mathsf{pow}_2(t) \in \mathit{Terms}(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$
    if $\mathcal{I} \models \mathcal{L}$ **return** "sat"
    $\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L}$ and $\mathcal{I} \not\models \psi\}$
  end loop
end function

SOLVE: SMT solver for integer arithmetic with uninterpreted functions.

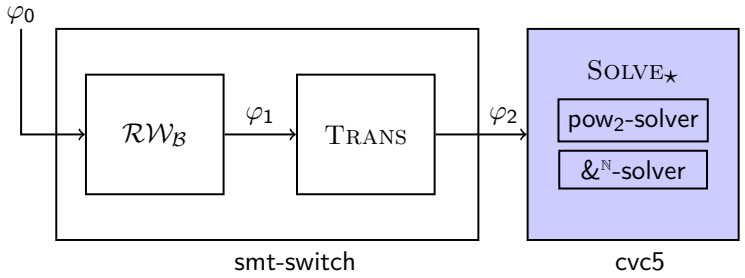
CEGAR Procedure



function SOLVE$_\star(\varphi)$
  $\Gamma \leftarrow \{\varphi\}$
  loop
    $result, \mathcal{I} \leftarrow$ SOLVE(Γ)
    if *result* is "unsat" **return** "unsat"
    $\mathcal{L} \leftarrow \mathcal{L}_{\&^{\mathbb{N}}}(\{\&^{\mathbb{N}}(k, t_1, t_2) \mid \&^{\mathbb{N}}(k, t_1, t_2) \in \mathit{Terms}(\Gamma)\}, \mathcal{I})$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{\mathsf{pow}_2}(\{\mathsf{pow}_2(t) \mid \mathsf{pow}_2(t) \in \mathit{Terms}(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$
    if $\mathcal{I} \models \mathcal{L}$ **return** "sat"
    $\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L}$ and $\mathcal{I} \not\models \psi\}$
  end loop
end function

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

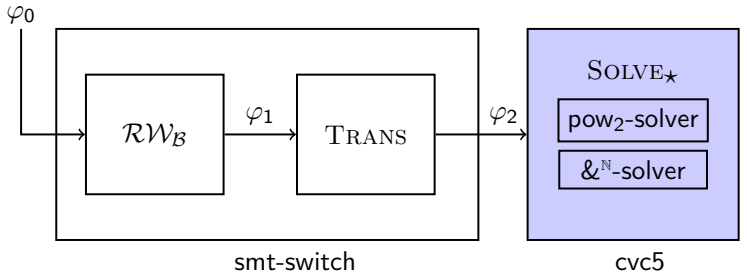
CEGAR Procedure



function Solve$_\star(\varphi)$
  $\Gamma \leftarrow \{\varphi\}$
  loop
    $result, \mathcal{I} \leftarrow$ Solve(Γ)
    if *result* is "unsat" **return** "unsat"
    $\mathcal{L} \leftarrow \mathcal{L}_{\&^\mathbb{N}}(\{\&^\mathbb{N}(k, t_1, t_2) \mid \&^\mathbb{N}(k, t_1, t_2) \in \mathit{Terms}(\Gamma)\}, \mathcal{I})$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{\mathsf{pow}_2}(\{\mathsf{pow}_2(t) \mid \mathsf{pow}_2(t) \in \mathit{Terms}(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$
    if $\mathcal{I} \models \mathcal{L}$ **return** "sat"
    $\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L}$ and $\mathcal{I} \not\models \psi\}$
  end loop
end function

Interpretation satisfies all lemmas ⇒ formula is satisfiable.

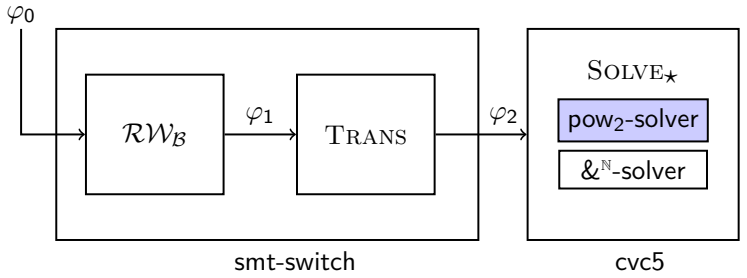
CEGAR Procedure

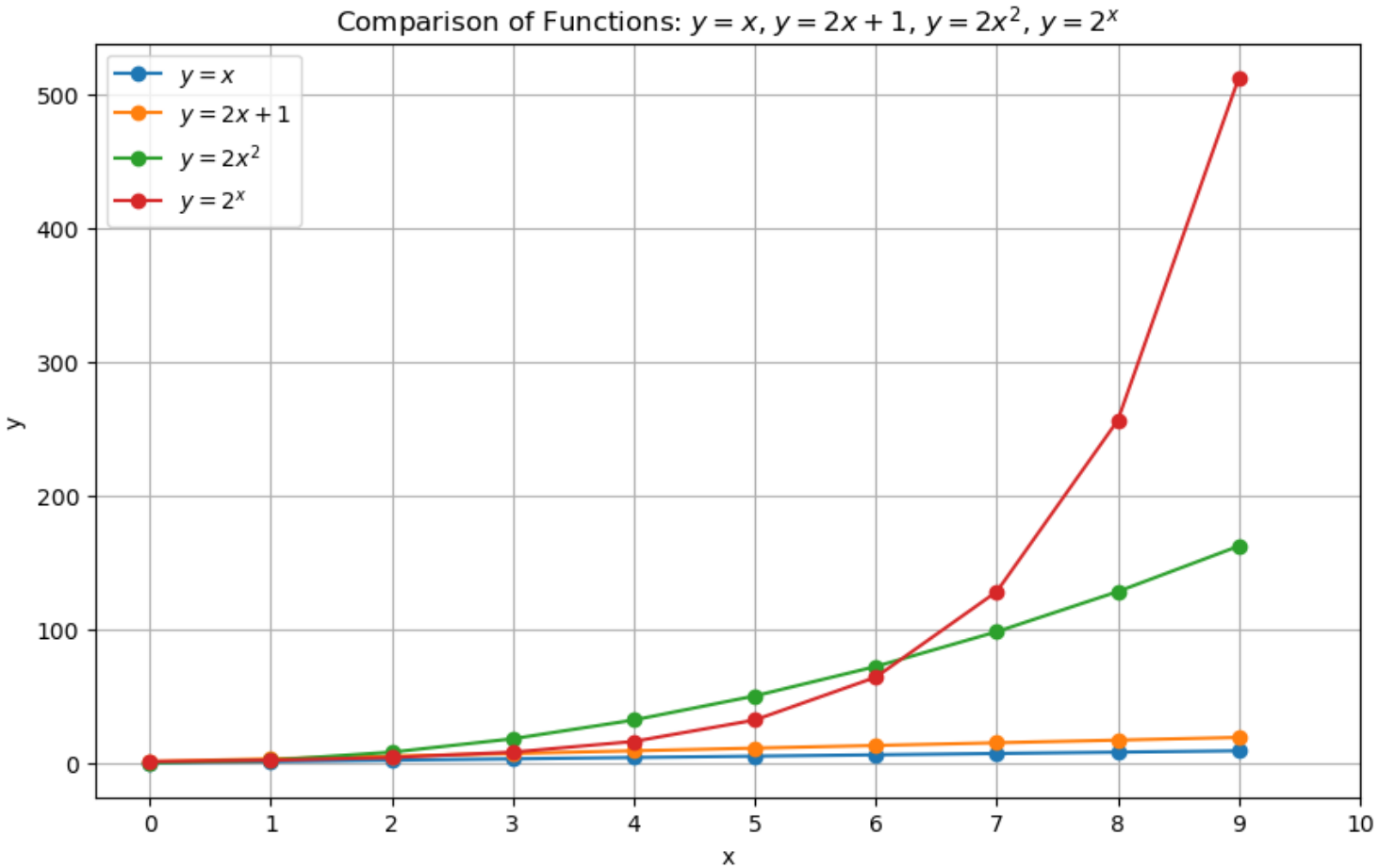


function Solve$_\star(\varphi)$
  $\Gamma \leftarrow \{\varphi\}$
  loop
    $result, \mathcal{I} \leftarrow$ Solve(Γ)
    if *result* is "unsat" **return** "unsat"
    $\mathcal{L} \leftarrow \mathcal{L}_{\&^\mathbb{N}}(\{\&^\mathbb{N}(k, t_1, t_2) \mid \&^\mathbb{N}(k, t_1, t_2) \in \mathit{Terms}(\Gamma)\}, \mathcal{I})$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{\mathsf{pow}_2}(\{\mathsf{pow}_2(t) \mid \mathsf{pow}_2(t) \in \mathit{Terms}(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$
    if $\mathcal{I} \models \mathcal{L}$ **return** "sat"
    $\Gamma \leftarrow \Gamma \cup \{\psi \mid \psi \in \mathcal{L}$ and $\mathcal{I} \not\models \psi\}$
  end loop
end function

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

bound Lemma


φ_0
$\mathcal{RW}_{\mathcal{B}}$
φ_1
Trans
φ_2
Solve$_\star$
pow$_2$-solver
&$^{\mathbb{N}}$-solver
smt-switch
cvc5


Comparison of Functions: $y = x$, $y = 2x + 1$, $y = 2x^2$, $y = 2^x$
$y = x$
$y = 2x + 1$
$y = 2x^2$
$y = 2^x$
y
x
0
100
200
300
400
500
0
1
2
3
4
5
6
7
8
9
10

Evaluation on *alive*

Family	Considered	Proved			
		[8]	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

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

Evaluation on *ic*

$\ell[x]$	$\approx$	$\not\approx$	$<_{\mathrm{u}}$	$>_{\mathrm{u}}$	$\leq_{\mathrm{u}}$	$\geq_{\mathrm{u}}$	$<_{\mathrm{s}}$	$>_{\mathrm{s}}$	$\leq_{\mathrm{s}}$	$\geq_{\mathrm{s}}$
$-^{\mathcal{B}} x \bowtie t$	✓✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓✓	✓
$\sim x \bowtie t$	✓✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓✓	✓✓
$x \,\&\, s \bowtie t$	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✗	✗	✗	✗
$x \mid s \bowtie t$	✓	✓✓	✓✓	✓	✓✓	✓✓	✗	✗	✗	✗
$x \ll s \bowtie t$	✓	✓	✓✓	✓	✓✓	✓	✗	✗	✗	✗
$s \ll x \bowtie t$	✓✓	✓	✓	✓	✓✓	✓	✗	✓	✗	✓
$x \gg s \bowtie t$	✓✓	✓	✓✓	✓	✓✓	✓✓	✓	✗	✓	✗
$s \gg x \bowtie t$	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓	✓
$x \gg_a s \bowtie t$	✓	✓	✓✓	✓✓	✓✓	✓	✗	✓	✗	✓
$s \gg_a x \bowtie t$	✓✓	✓	✓	✓	✓✓	✓	✗	✗	✗	✓
$x +^{\mathcal{B}} s \bowtie t$	✓✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓✓	✓✓
$x \cdot^{\mathcal{B}} s \bowtie t$	✗	✓	✓✓	✗	✓✓	✗	✗	✗	✗	✗
$x \mathrm{div}^{\mathcal{B}} s \bowtie t$	✓	✓✓	✓✓	✓✓	✓✓	✗	✓	✓	✓	✓
$s \mathrm{div}^{\mathcal{B}} x \bowtie t$	✓	✓	✓✓	✓✓	✓✓	✓✓	✓	✓	✓✓	✗
$x \mathrm{mod}^{\mathcal{B}} s \bowtie t$	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✗	✓	✗	✓
$s \mathrm{mod}^{\mathcal{B}} x \bowtie t$	✗	✓✓	✓✓	✓✓	✓✓	✓	✓	✗	✓	✗
$x \circ s \bowtie t$	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗
$s \circ x \bowtie t$	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗

Mutated Benchmarks

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

Evaluation With Multiple Bit-Widths

Benchmarks	#	BASELINE	EAGER	PBV	VBS
alive	17	0	9	**10**	10
ic	20	0	5	**8**	8
rewrite	0	0	0	0	0
syrew	0	0	0	0	0
lemmas	2	0	0	0	0
icfb	16	0	8	**10**	10
mut	316	0	88	**150**	151
total	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][9]:

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

- The corrected condition is:

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

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

[9] Niemetz, 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	$T_{IA}(\mathsf{pow}_2, \&^{\mathbb{N}}, \mid^{\mathbb{N}}, \oplus^{\mathbb{N}})$						T_3	T_4
Multiple Bit-widths	✗	✗	✗	✗	✗	✗	✗	✗
Lazy pow_2	✗	✗	✗	✗	✗	✗	✓	✗
Lazy $\&^{\mathbb{N}}$	✗	✗	✗	✗	✗	✗	✗	✓
$\mid$*-elimination*	✓	✗	✗	✓	✗	✗	✗	✗
$\oplus$*-elimination*	✗	✓	✗	✓	✗	✗	✗	✗
$\gg$ *without* mod	✗	✗	✓	✓	✗	✗	✗	✗
New lemmas for $\&^{\mathbb{N}}$	✗	✗	✗	✗	✗	✓	✗	✓
New lemmas for pow_2	✗	✗	✗	✗	✓	✗	✓	✗
No redundant axioms	✗	✗	✗	✗	✗	✗	✗	✗
$\mathcal{RW}_{\mathcal{B}}$	✗	✗	✗	✗	✗	✗	✗	✗
$\mathcal{RW}_{\mathcal{A}}$	✗	✗	✗	✗	✗	✗	✗	✗

$T_3 = T_{IA}(\mathsf{pow}_{2\star}, \&^{\mathbb{N}})$: pow_2 has a fixed interpretation, $\&^{\mathbb{N}}$ is freely interpreted.
$T_4 = T_{IA}(\mathsf{pow}_2, \&^{\mathbb{N}}_{\star})$: $\&^{\mathbb{N}}$ has a fixed interpretation, pow_2 is freely interpreted.

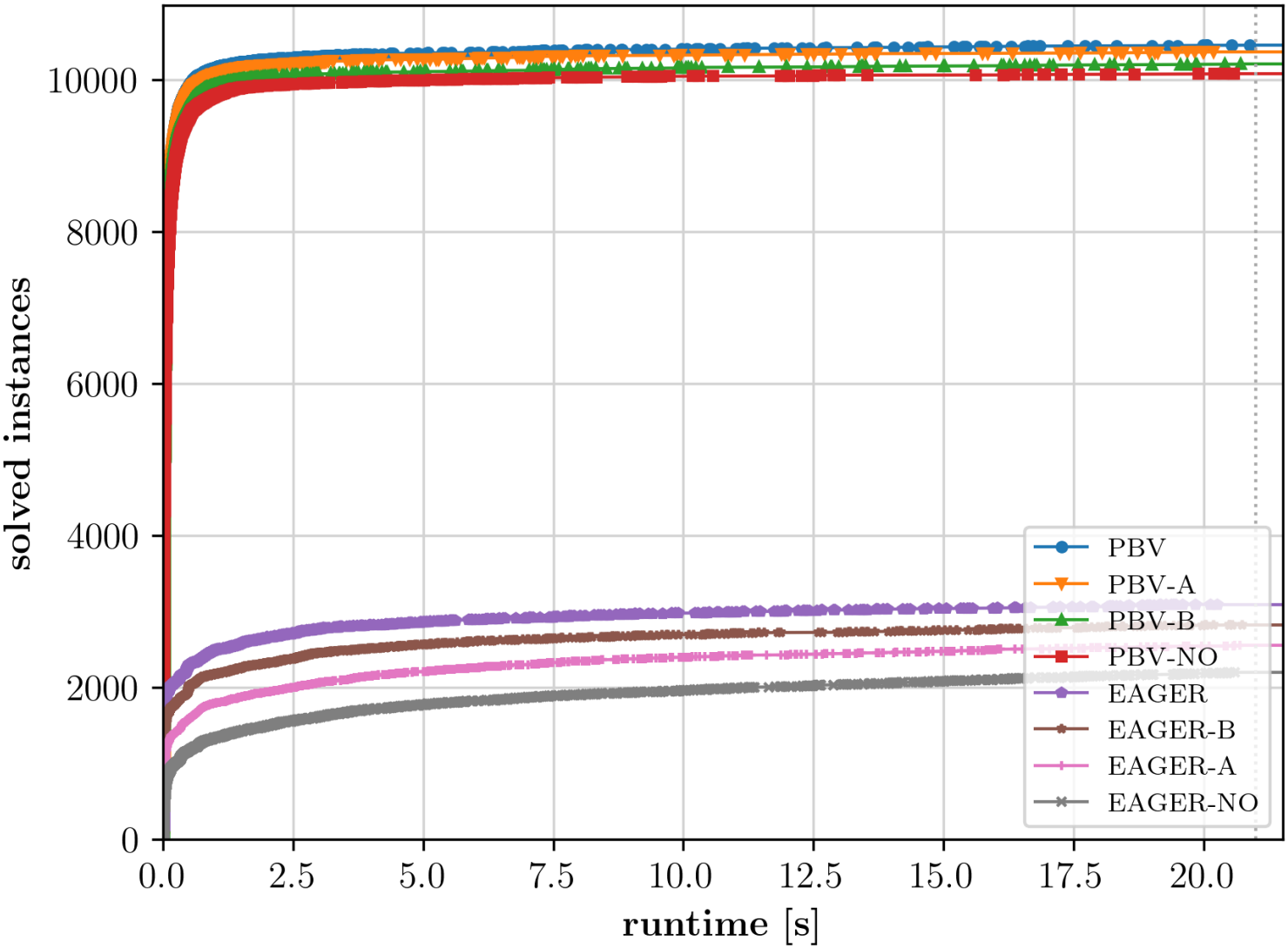
Evaluation Attributes Results

Benchmarks	#	OR-E	XOR-E	SH-M-E	ALL-E	POW2++	PIAND++	POW2-L	PIAND-L
alive	200	71	71	71	71	71	71	63	**74**
ic	180	44	44	48	49	43	43	**55**	45
rewrite	2006	692	653	724	758	669	633	681	**797**
syrew	1500	561	556	**604**	603	570	551	580	593
lemmas	70	12	12	13	13	12	12	**17**	13
icfb	46	1	1	1	1	1	1	1	1
mut	9441	692	663	774	**794**	669	634	732	733
total	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
Multiple Bit-widths	✓	✓	✓	✓	✓	✓
Lazy pow_2	✗	✗	✗	✓	✓	✓
Lazy $\&^{\mathbb{N}}$	✗	✗	✗	✓	✓	✓
$\mid$*-elimination*	✓	✓	✓	✓	✓	✓
$\oplus$*-elimination*	✓	✓	✓	✓	✓	✓
$\gg$ *without* mod	✓	✓	✓	✓	✓	✓
New lemmas for $\&^{\mathbb{N}}$	✓	✓	✓	✓	✓	✓
New lemmas for pow_2	✓	✓	✓	✓	✓	✓
No redundant axioms	✓	✓	✓	✓	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✗	✓	✗	✗	✓
$\mathcal{RW}_{\mathcal{A}}$	✗	✓	✗	✓	✗	✗

Evaluation of Rewriters



Evaluation Lazy and Eager

Benchmarks	#	EAGER-NO	EAGER-A	EAGER-B	PBV-A	PBV-NO	PBV-B	VBS
alive	200	87	95	85	105	97	**100**	125
ic	180	58	59	59	75	75	**77**	83
rewrite	2006	732	925	1068	1296	1148	**1189**	1381
syrew	1500	608	721	606	911	796	**798**	956
lemmas	70	14	14	14	23	**23**	**23**	25
icfb	46	9	9	9	11	11	**12**	12
mut	9441	885	902	1088	4832	4840	**4881**	5138
total	13443	2393	2725	2929	7253	6990	**7080**	7720
sat		0	0	0	3652	**3657**	3651	3841
unsat		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