Department of
Computer Science
Faculty of Exact Sciences
Bar-Ilan University

Bit-precise Reasoning with Parametric Bit-vectors

Zvika Berger

June 22, 2025

Paper

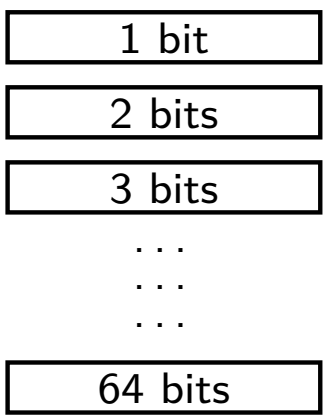
- This thesis led to a paper.
- Authors: Zvika Berger, Yoni Zohar, Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark Barrett, and Cesare Tinelli.
- The paper was accepted to the SAT Conference 2025.

Outline

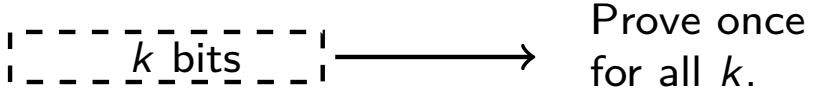
Motivation: Parametric Bit-Vectors

- Software works with sequences of bits.
- Verification needs bit-vector reasoning.
- Example: LLVM optimizations.
- Alive proves one bit-width.
- Our goal: prove all bit-widths.

Fixed Bit-Width



Parametric Bit-Width



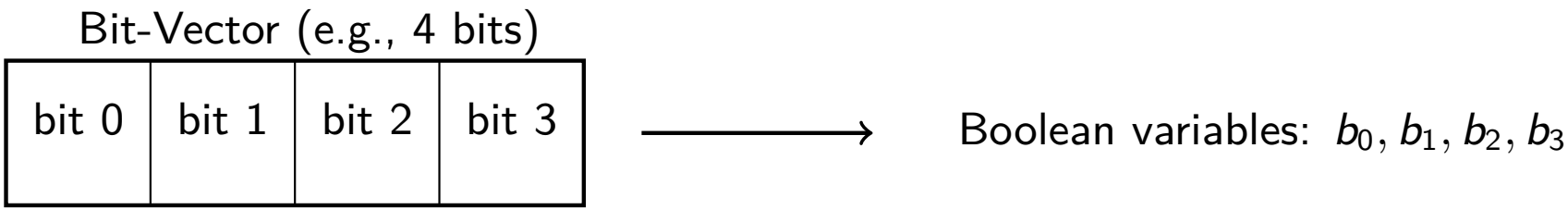
Fixed-Width Bit-Vectors

- **Theory of Fixed-Size Bit-Vectors** provides bit-precise semantics.
- A **bit-vector** is a fixed-length sequence of bits.

Constants:	010, 000, $2_{[3]}$
Variables:	$x_{[3]}$, $y_{[3]}$, $x_{[8]}$
Predicates:	$<_u$, $>_s$, $\ldots$
Bitwise operators:	$\&$, $\mid$, $\oplus$, $\ldots$
Arithmetic operators:	$+$, $-$, $\cdot$, mod, $\ldots$

Fixed-Width Bit-Vectors

- The state of the art for bit-vector formulas is reduction to SAT.
- Each constant or variable is translated into Boolean variables, one per bit of its width.
- 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 bit-widths

Outline

First Order Theories

- A *theory* T is a pair $(\Sigma, \mathcal{I})$.
- Σ is a signature consisting of:
  - A set Σ^s of *sort symbols*.
  - A set Σ^f of *function symbols*.
  - Arities of function symbols.
- $\mathcal{I}$ is a non-empty class of Σ-interpretations.

  - Typically, a single domain.
  - Multiple domains are possible.
  - Many-sorted FOL formalizes them explicitly.

Integer Arithmetic Theory

- $T_{IA} = (\Sigma_{IA}, I_{IA})$.
- Σ_{IA} is defined as follows:

Symbol	**SMT-LIB Syntax**	**Sort**
$\approx_{\mathsf{Int}}, \not\approx_{\mathsf{Int}}$	`=`, `distinct`	$\mathsf{Int} \times \mathsf{Int} \to \mathsf{Bool}$
$0, 1, 2, \ldots$	`0`,`1`,`2`,...	Int
$+, -, \cdot$, div, mod	`+`, `-`, `*`, `div`, `mod`	$\mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$
$\leq, \geq, <, >$	`<=`, `>=`, `<`, `>`	$\mathsf{Int} \times \mathsf{Int} \to \mathsf{Bool}$

- I_{IA} is based on SMT-LIB.

Fixed Width Bit-Vectors Theory

- $T_{\mathrm{BV}} = (\Sigma_{\mathrm{BV}}, I_{\mathrm{BV}})$.
- Σ_{BV} is defined in SMT-LIB
- It includes the following representative set of operators:

Symbol	**SMT-LIB Syntax**	**Sort**
$\approx, \not\approx$	`=, distinct`	$\sigma_{[n]} \times \sigma_{[n]} \rightarrow \mathrm{Bool}$
$<_{\mathrm{u}}, >_{\mathrm{u}}, <_{\mathrm{s}}, >_{\mathrm{s}}$	`bvult, bvugt, bvslt, bvsgt`	$\sigma_{[n]} \times \sigma_{[n]} \rightarrow \mathrm{Bool}$
$\leq_{\mathrm{u}}, \geq_{\mathrm{u}}, \leq_{\mathrm{s}}, \geq_{\mathrm{s}}$	`bvule, bvuge, bvsle, bvsge`	$\sigma_{[n]} \times \sigma_{[n]} \rightarrow \mathrm{Bool}$
$\sim, -^{\mathcal{B}}$	`bvnot, bvneg`	$\sigma_{[n]} \rightarrow \sigma_{[n]}$
$\&, \mid, \oplus$	`bvand, bvor, bvxor`	$\sigma_{[n]} \times \sigma_{[n]} \rightarrow \sigma_{[n]}$
$\ll, \gg, \gg_a$	`bvshl, bvlshr, bvashr`	$\sigma_{[n]} \times \sigma_{[n]} \rightarrow \sigma_{[n]}$
$+^{\mathcal{B}}, \cdot^{\mathcal{B}}, \mathrm{mod}^{\mathcal{B}}, \mathrm{div}^{\mathcal{B}}$	`bvadd, bvmul, bvurem, bvudiv`	$\sigma_{[n]} \times \sigma_{[n]} \rightarrow \sigma_{[n]}$
$[u : l]$	extract $(0 \leq l \leq u < n)$	$\sigma_{[n]} \rightarrow \sigma_{[u-l+1]}$
$\circ$	`concatenation`	$\sigma_{[n]} \times \sigma_{[m]} \rightarrow \sigma_{[n+m]}$
ext_s	`sign_extend`	$Int \times \sigma_{[n]} \rightarrow \sigma_{[n+m]}$
ext_z	`zero_extend`	$Int \times \sigma_{[n]} \rightarrow \sigma_{[n+m]}$

- I_{BV} is based on SMT-LIB.

Fixed-size Bit-Vectors Theory

- A formula over fixed-size bit-vectors in SMT-LIB syntax:

```
(set-logic ALL)
(declare-const x (_ BitVec 4))
(declare-const y (_ BitVec 4))
(assert (bvult x y))
(assert (bvult y (bvadd x x)))
(check-sat)
```

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
$\_[\_:\_]$	`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
$\lvert\_\rvert$	`bvsize`	PBV $\rightarrow$ Int
topbv	`int_to_pbv`	Int $\times$ Int $\rightarrow$ PBV

Parametric Bit-Vectors Interpretation

- I_{PBV}:
  - `Int` domain: all integers.
  - `PBV` domain: all fixed-size bit-vectors.
  - Σ_{IA} operators as in T_{IA}.
  - $\Sigma_{PBV} \setminus \Sigma_{IA}$ operators: same as in T_{BV} if bit-widths match; arbitrary otherwise.
  - $topbv(3, 2) = 010$.
  - $|010| = 3$.
- $\mathcal{A}$ is a T_{PBV}-interpretation if it satisfies all requirements defined by I_{PBV}.

Parametric Bit-Vectors Theory

- A formula over parametric bit-vectors in SMT-LIB style syntax:

```
(set-logic ALL)
(declare-const k Int)
(declare-const m Int)
(declare-const x (_ BitVec k))
(declare-const y (_ BitVec m))
(assert (bvult x y))
(assert (bvult y (bvadd x x)))
(check-sat)
```

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:

- $x \mapsto 0$.
- $x \circ x \mapsto 00$
- $x +^{\mathcal{B}} (x \circ x) \mapsto 0$ (since the bit-widths differ, the result can be arbitrary)

Therefore, φ is T_{PBV}-satisfiable.

Bit-width function - K

function $\mathrm{K}(e)$

match e:

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

end function

Bit-width function - K

function $\mathrm{K}(e)$

match e:

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

end function

Bit-width function - K

function $\mathrm{K}(e)$

match e:

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

end function

Bit-width function - $\textsc{Type}$

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

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

end function

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

Bit-width function - $\textsc{Type}$

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

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

end function

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

Bit-width function - $\textsc{Type}$

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

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

end function

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

Bit-width function - $\textsc{Type}$

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

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

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

Admissible Satisfiability – Examples

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

Example

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

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

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

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

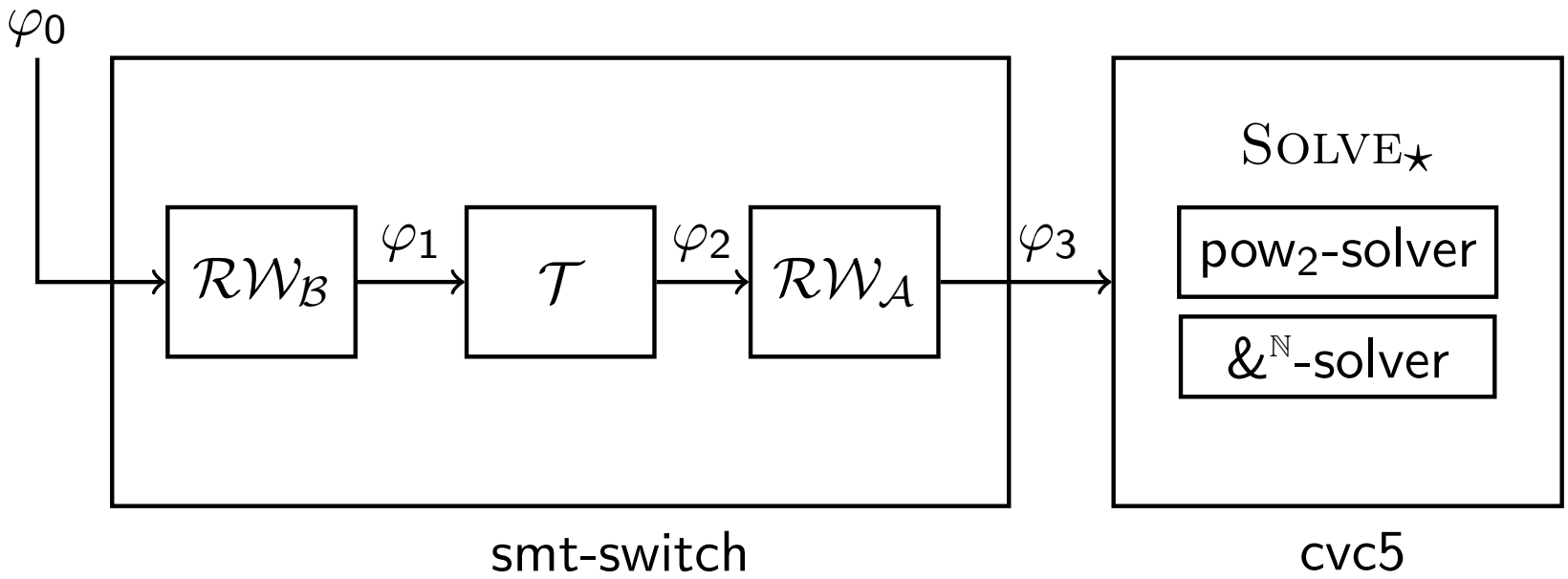
Theory Summary

- T_{BV} can't handle parametric bit-widths.
- The T_{PBV} theory alone is not sufficient.
- admissible-T_{PBV} helps define the theory of parametric bit-vectors.
- More expressive than [CADE'19]: supports bit-width reasoning.
- Simpler and clearer than [CADE'19].

Outline

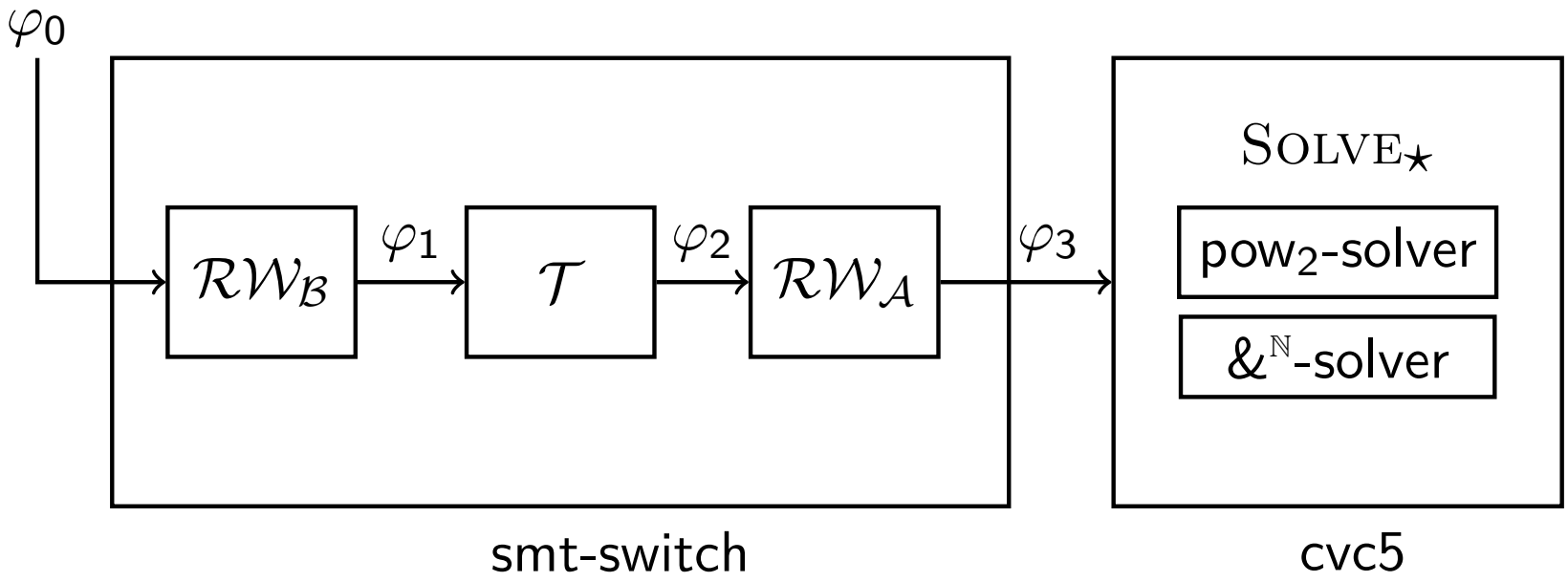
PBV Solver - Architecture

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



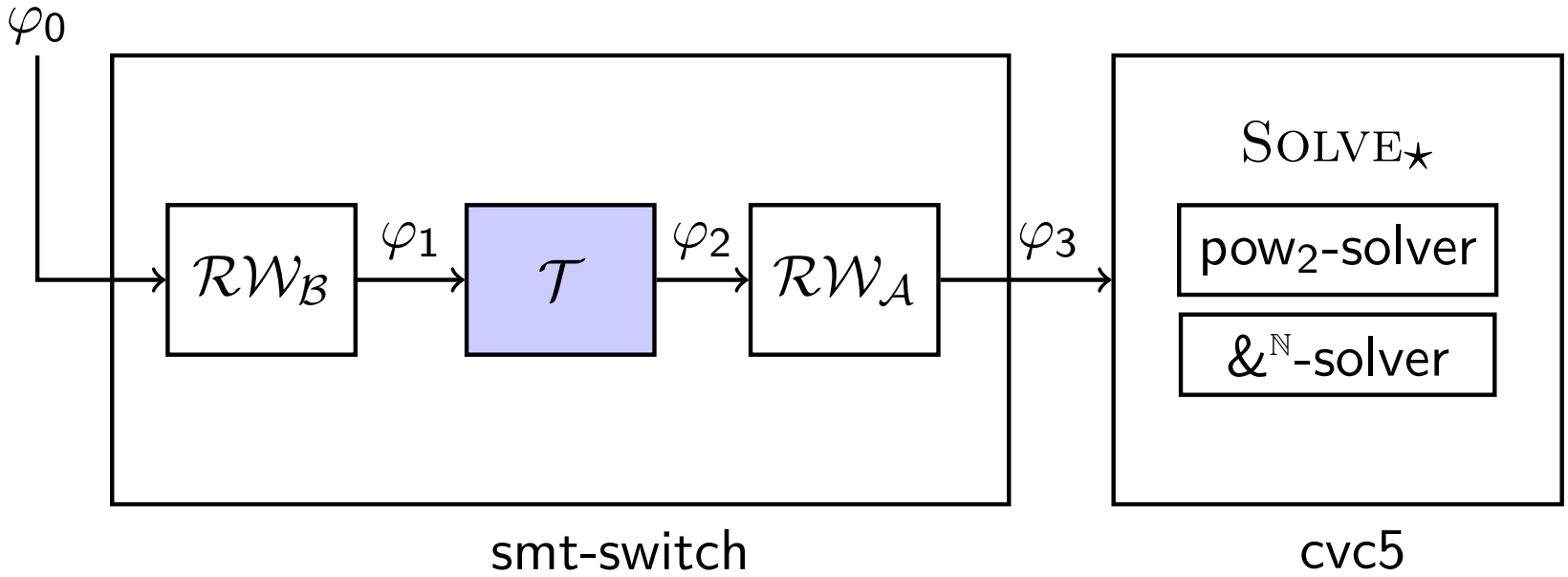
PBV Solver - Architecture

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

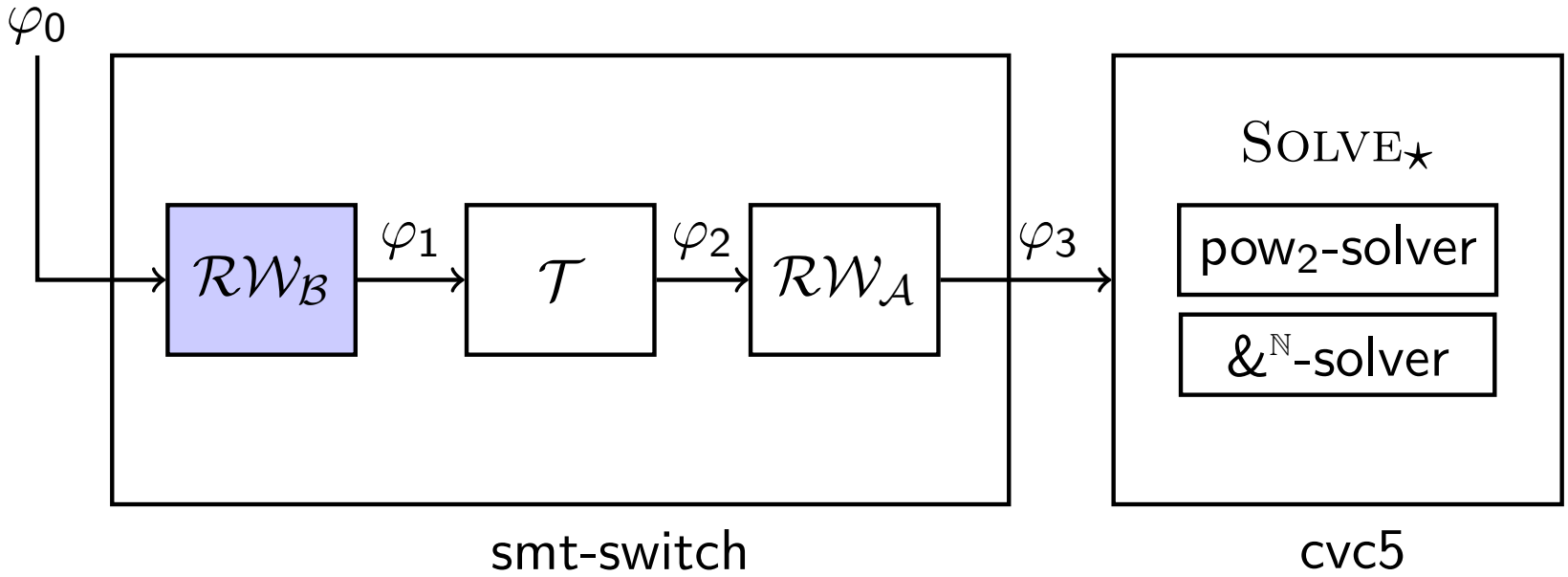


smt-switch is an open-source API for SMT solving.

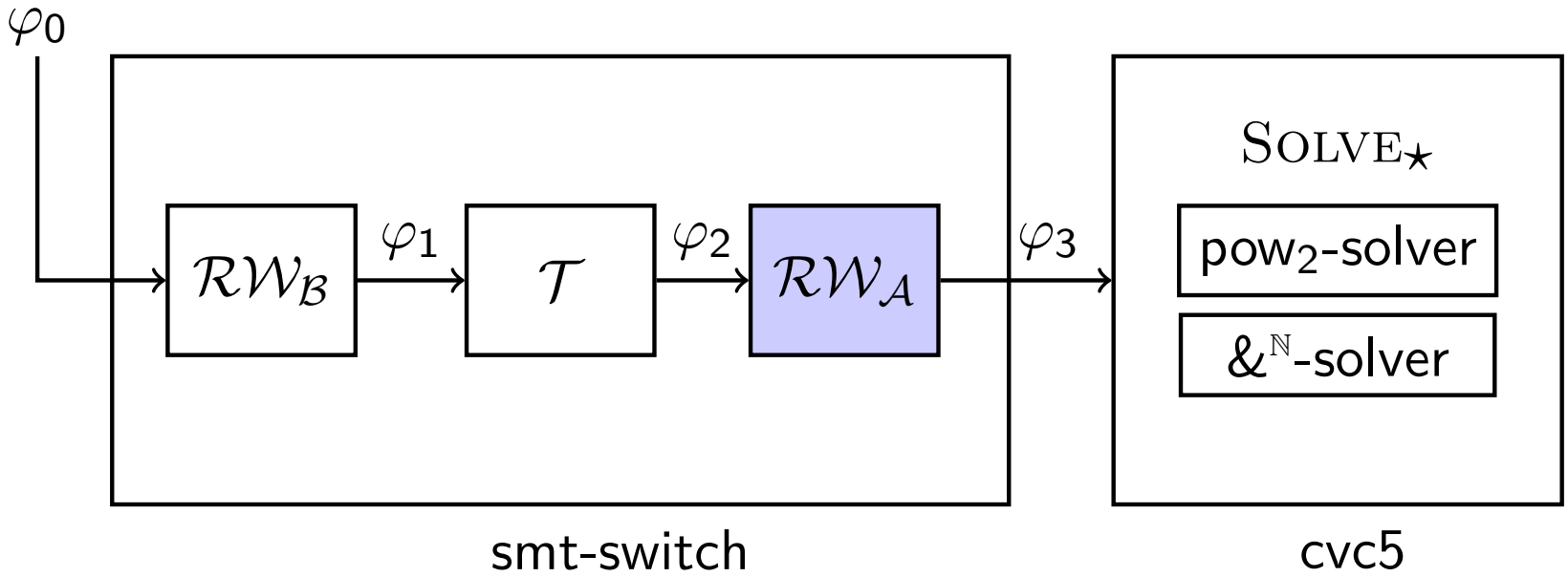
PBV Solver - Architecture



PBV Solver - Architecture

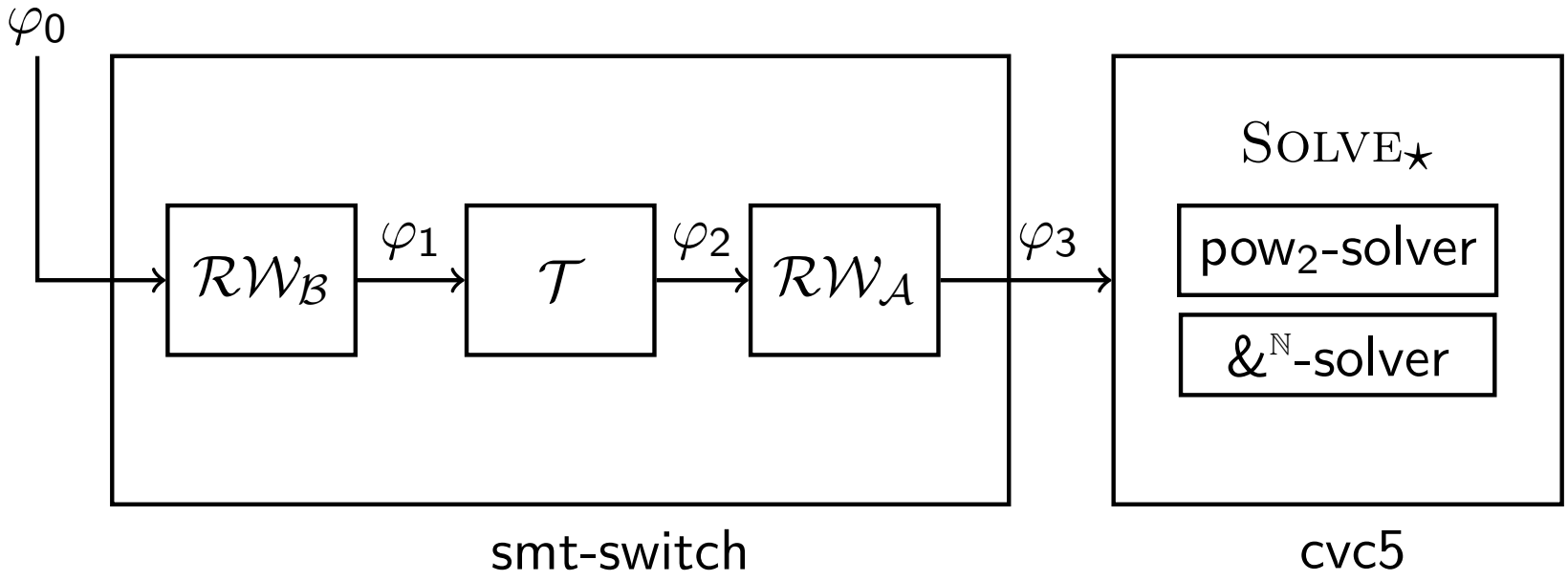


PBV Solver - Architecture



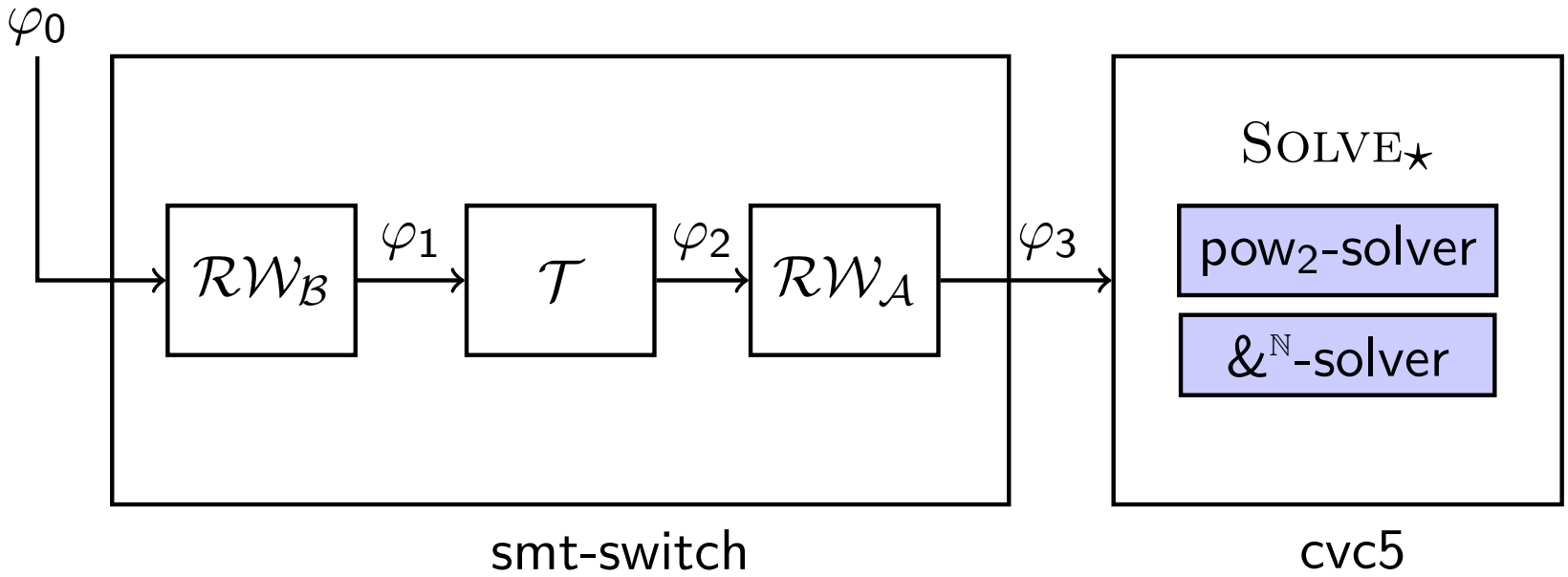
PBV Solver - Architecture

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

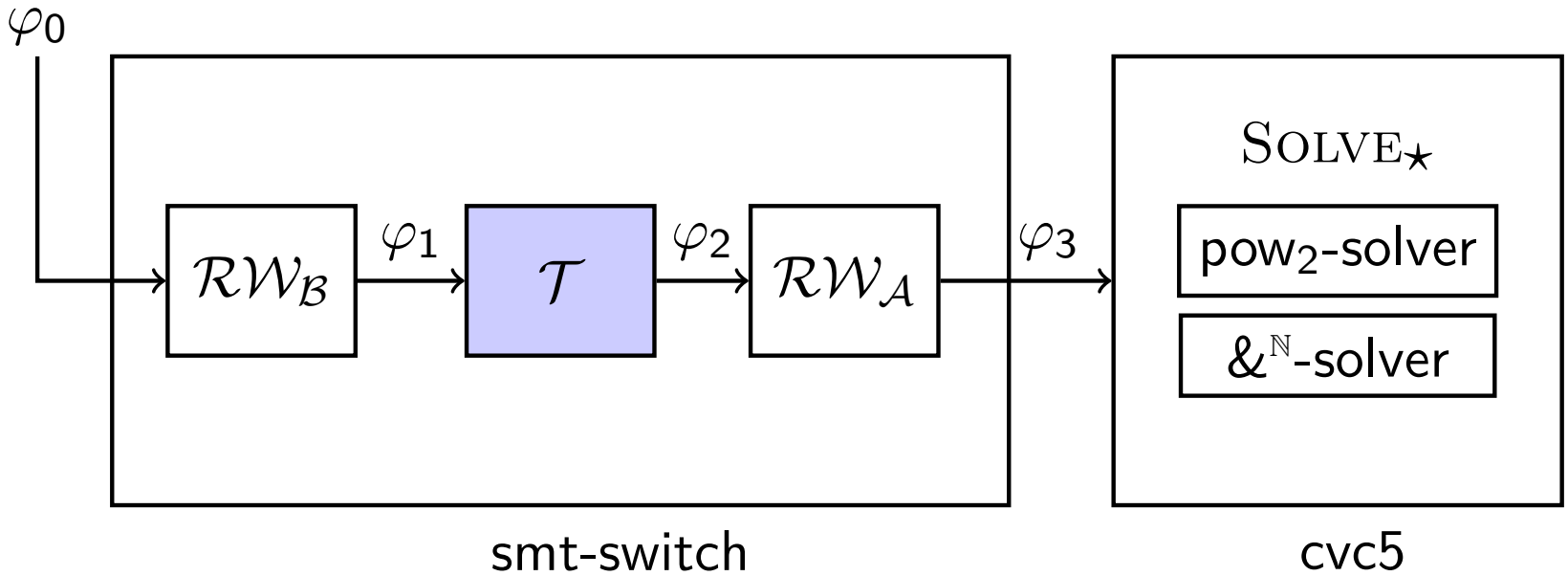


cvc5 is an open-source SMT solver.

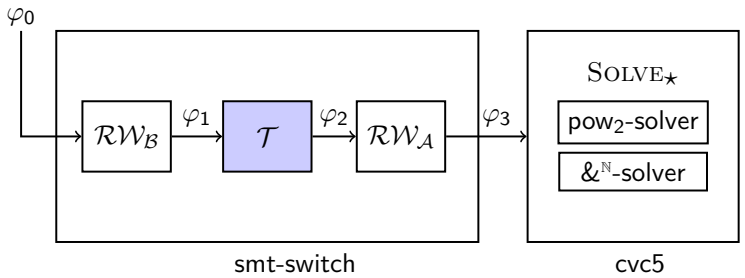
PBV Solver - Architecture



PBV Solver - $\mathcal{T}$



Translation Function $\mathcal{T}$



function $\mathcal{T}(\varphi)$
  return $\mathrm{C}(\varphi) \wedge \textsc{Range}(\varphi) \wedge \textsc{Type}(\varphi)$
end function

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

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

end function

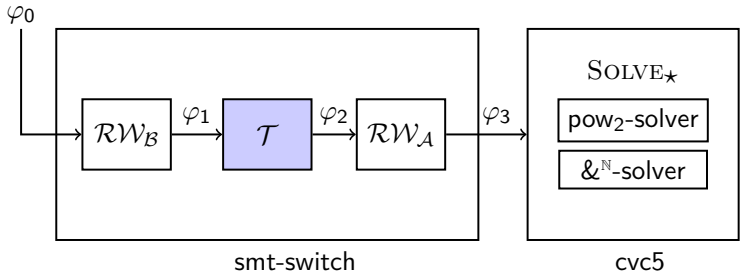
- Based on [CADE'19].
- Our changes are highlighted in blue.

Translation Function C

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

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

end function

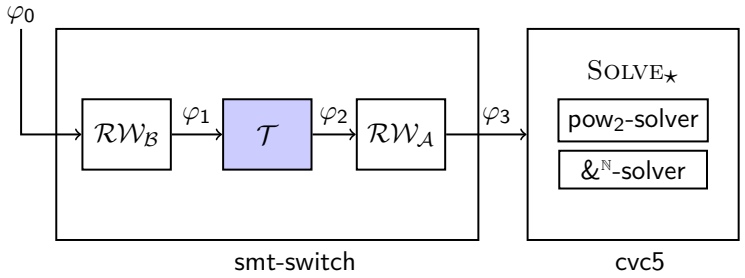


Translation Function C

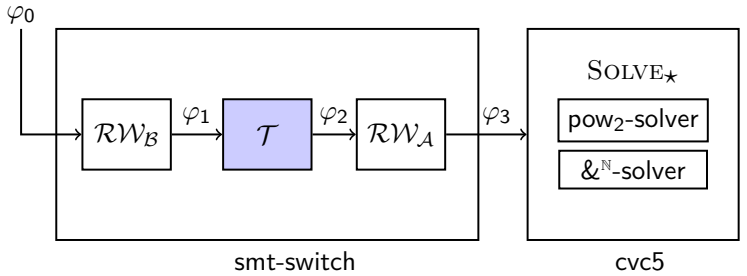
function $\mathrm{C}(e)$
  match e:

z	$\to z$
$\lvert t \rvert$	$\to \kappa(t)$
$\mathit{topbv}(k, t)$	$\to \mathrm{C}(t) \bmod \mathrm{pow}_2(k)$
x	$\to \chi(x)$
$t_1 \approx t_2$	$\to \mathrm{C}(t_1) \approx \mathrm{C}(t_2)$
$\bowtie_u (t_1, t_2)$	$\to \mathrm{C}(t_1) \bowtie \mathrm{C}(t_2)$
$\bowtie_s (t_1, t_2)$	$\to \mathrm{uts}(\kappa(t_1), \mathrm{C}(t_1)) \bowtie \mathrm{uts}(\kappa(t_2), \mathrm{C}(t_2))$
$t_1 +^{\mathcal{B}} t_2$	$\to (\mathrm{C}(t_1) + \mathrm{C}(t_2)) \bmod \mathrm{pow}_2(\kappa(t_1))$
$t_1 -^{\mathcal{B}} t_2$	$\to (\mathrm{C}(t_1) - \mathrm{C}(t_2)) \bmod \mathrm{pow}_2(\kappa(t_1))$
$t_1 \cdot^{\mathcal{B}} t_2$	$\to (\mathrm{C}(t_1) \cdot \mathrm{C}(t_2)) \bmod \mathrm{pow}_2(\kappa(t_1))$
$t_1 \mathrm{div}^{\mathcal{B}} t_2$	$\to \mathit{ite}(\mathrm{C}(t_2) \approx 0, \mathrm{pow}_2(\kappa(t_1)) - 1, \mathrm{C}(t_1) \mathrm{div} \mathrm{C}(t_2))$
$t_1 \mathrm{mod}^{\mathcal{B}} t_2$	$\to \mathit{ite}(\mathrm{C}(t_2) \approx 0, \mathrm{C}(t_1), \mathrm{C}(t_1) \bmod \mathrm{C}(t_2))$
$\sim t$	$\to \mathrm{pow}_2(\kappa(t)) - (\mathrm{C}(t) + 1)$
$-^{\mathcal{B}} t$	$\to (\mathrm{pow}_2(\kappa(t)) - \mathrm{C}(t)) \bmod \mathrm{pow}_2(\kappa(t))$
$t_1 \ll t_2$	$\to (\mathrm{C}(t_1) \cdot \mathrm{pow}_2(\mathrm{C}(t_2))) \bmod \mathrm{pow}_2(\kappa(t_1))$
$t_1 \gg t_2$	$\to \mathrm{C}(t_1) \mathrm{div} \mathrm{pow}_2(\mathrm{C}(t_2))$
$t_1 \,\&\, t_2$	$\to \&^{\mathbb{N}}(\kappa(t_1), \mathrm{C}(t_1), \mathrm{C}(t_2))$
$t_1 \mid t_2$	$\to \mathrm{C}(t_1) + \mathrm{C}(t_2) - \&^{\mathbb{N}}(\kappa(t_1), \mathrm{C}(t_1), \mathrm{C}(t_2))$
$t_1 \oplus t_2$	$\to \mathrm{C}(t_1) + \mathrm{C}(t_2) - 2 \cdot \&^{\mathbb{N}}(\kappa(t_1), \mathrm{C}(t_1), \mathrm{C}(t_2))$
$t_1 \circ t_2$	$\to \mathrm{C}(t_1) \cdot \mathrm{pow}_2(\kappa(t_2)) + \mathrm{C}(t_2)$
$t[i:j]$	$\to (\mathrm{C}(t) \mathrm{div} \mathrm{pow}_2(j)) \bmod \mathrm{pow}_2(i - j + 1)$
$\mathit{ext}_z(n, t)$	$\to \mathrm{C}(t)$
$\mathit{ext}_s(n, t)$	$\to \mathit{ite}(\mathrm{C}(t[\kappa(t)-1]) \approx 1, (\mathrm{pow}_2(n)-1) \cdot \mathrm{pow}_2(\kappa(t)) + \mathrm{C}(t), \mathrm{C}(t))$
$\bullet(t_1, \ldots, t_n)$	$\to \bullet(\mathrm{C}(t_1), \mathrm{C}(t_2), \mathrm{C}(t_3))$

end function

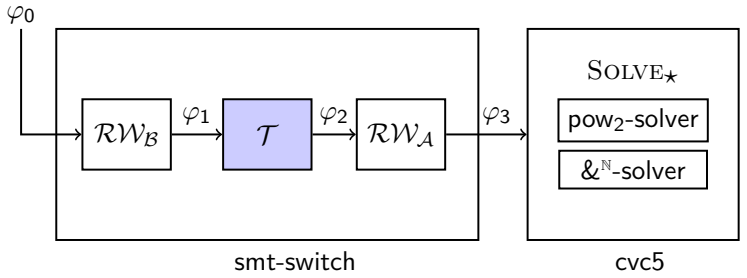


Translation Function $\mathbb{C}$- New Operators



- Operators newly supported in our approach:
  - $|t|$
  - $topbv(k, t)$
  - $t[i : j]$
  - $ext_z(n, t)$
  - $ext_s(n, t)$

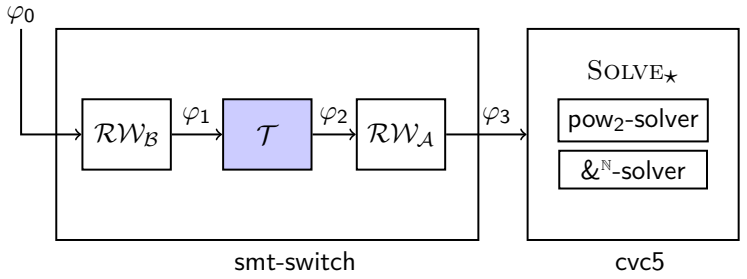
Translation Function C - New Operators



$$ext_z(n, t) \quad \mapsto \quad \mathrm{C}(t)$$

$$\begin{array}{lcl} ext_z(2, x) & \mapsto & \chi(x) \\ ext_z(n, x) & \mapsto & \chi(x) \\ ext_z(n+1, x) & \mapsto & \chi(x) \\ ext_z(n, x+x) & \mapsto & \mathrm{C}(x+x) = (\chi(x) + \chi(x)) \bmod \mathrm{pow}_2(\kappa(x)) \end{array}$$

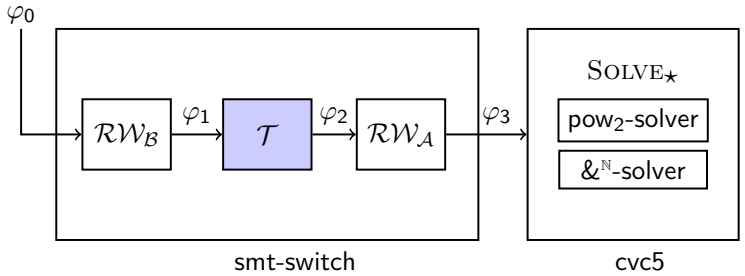
Translation Function C - New Operators



$$t[i:j] \quad \mapsto \quad (\mathrm{C}(t) \text{ div } \mathrm{pow}_2(j)) \bmod \mathrm{pow}_2(i-j+1)$$

$$\begin{aligned} x[0:0] \quad &\mapsto \quad (\chi(x) \text{ div } \mathrm{pow}_2(0)) \bmod \mathrm{pow}_2(1) \\ x[i:j] \quad &\mapsto \quad (\chi(x) \text{ div } \mathrm{pow}_2(j)) \bmod \mathrm{pow}_2(i-j+1) \end{aligned}$$

Improved Shift Translation



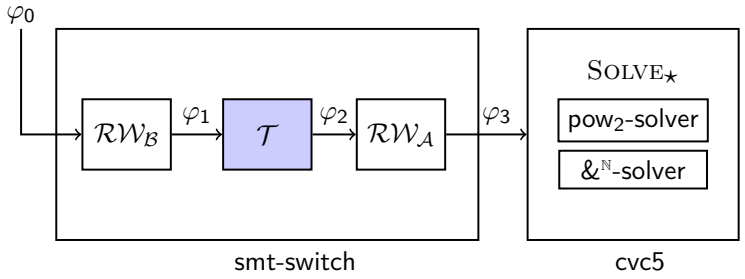
[*CADE'19*] translation:

$$t_1 \gg t_2 \quad \mapsto \quad (\mathrm{C}(t_1) \text{ div } \mathrm{pow}_2(\mathrm{C}(t_2))) \text{ mod } \mathrm{pow}_2(\kappa(t_1))$$

Improve:

$$t_1 \gg t_2 \quad \mapsto \quad \mathrm{C}(t_1) \text{ div } \mathrm{pow}_2(\mathrm{C}(t_2))$$

Improved Bitwise-Or Translation



Hacker's Delight:

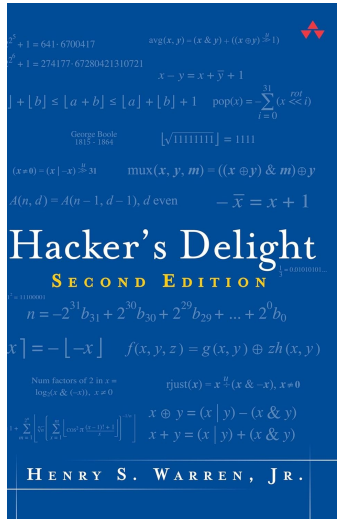
$$t_1 \mid t_2 \quad \mapsto \quad (t_1 + t_2) - (t_1 \,\&\, t_2)$$

Translation to integer:

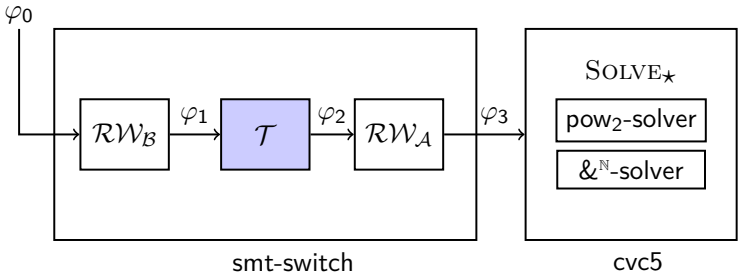
$$t_1 \mid t_2 \quad \mapsto \quad (((\mathrm{C}(t_1) + \mathrm{C}(t_2)) \bmod \mathrm{pow}_2(\kappa(t_1))) - \&^{\mathbb{N}}(\kappa(t_1), \mathrm{C}(t_1), \mathrm{C}(t_2))) \bmod \mathrm{pow}_2(\kappa(t_1))$$

Improve:

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



Improved Bitwise-Or Translation

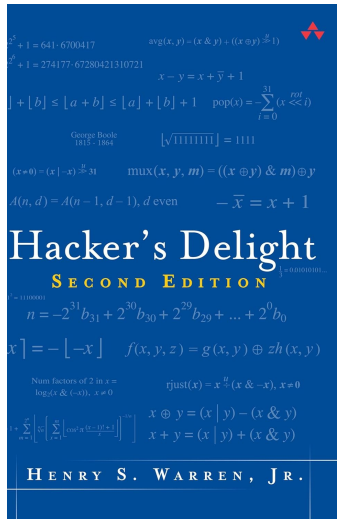


Hacker's Delight:

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

Translation to integer:

$$t_1 \mid t_2 \quad \mapsto \quad (((\mathrm{C}(t_1) + \mathrm{C}(t_2)) \bmod \mathrm{pow}_2(\kappa(t_1))) - \&^\mathbb{N}(\kappa(t_1), \mathrm{C}(t_1), \mathrm{C}(t_2))) \bmod \mathrm{pow}_2(\kappa(t_1))$$

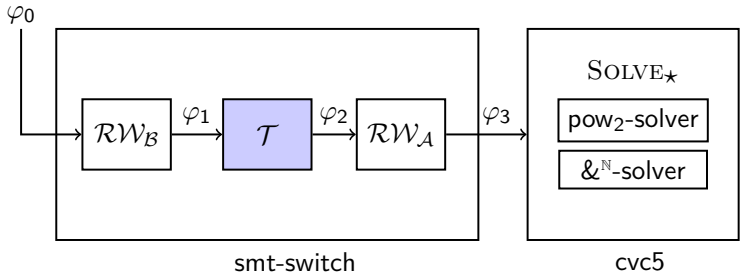


Improve:

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

$$t_1 \oplus t_2 \quad \mapsto \quad \mathrm{C}(t_1) + \mathrm{C}(t_2) - 2 \cdot \&^\mathbb{N}(\kappa(t_1), \mathrm{C}(t_1), \mathrm{C}(t_2))$$

Translation Function $\mathcal{T}$



Example

$\varphi = y \approx z \circ w$ where $\kappa(y) = k$, $\kappa(z) = m$, $\kappa(w) = n$.

$\mathcal{T}(\varphi) \mapsto \mathrm{C}(\varphi) \wedge \textsc{Range}(\varphi) \wedge \textsc{Type}(\varphi)$.

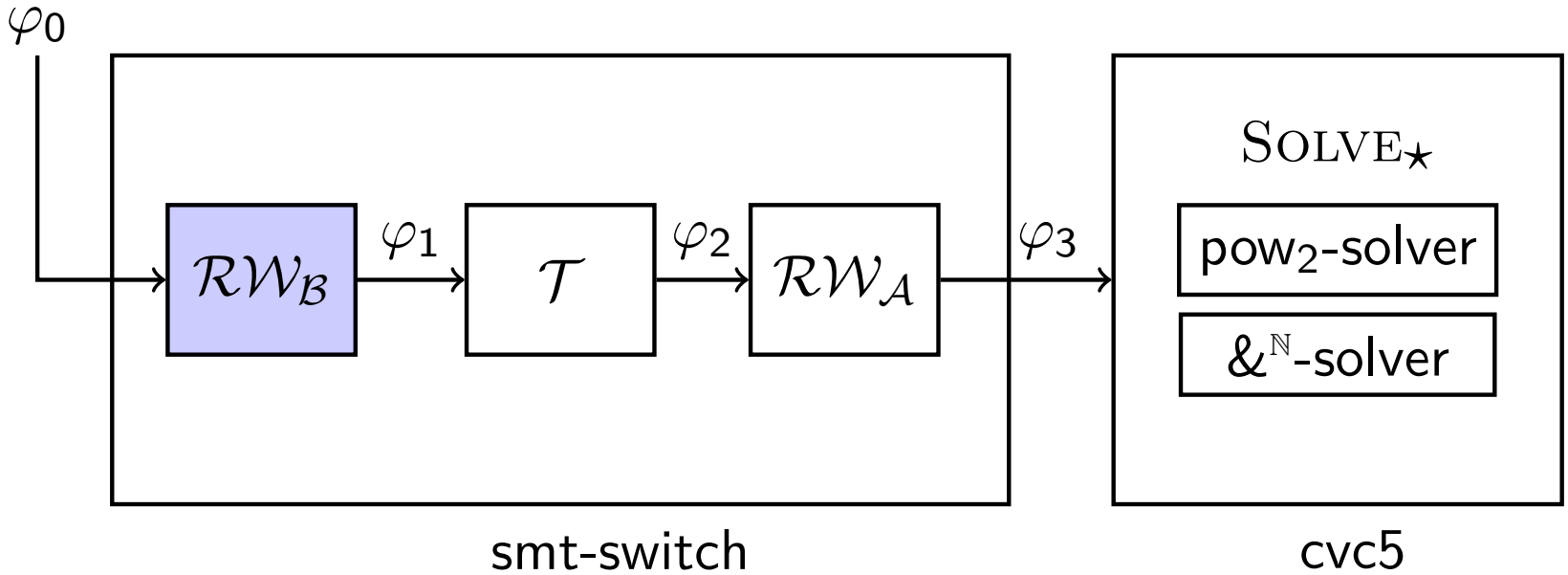
C(φ) : $y' = z' \cdot \mathrm{pow}_2(n) + w'$

Range(φ) :
$$y' \geq 0 \wedge y' < \mathrm{pow}_2(k)$$
$$z' \geq 0 \wedge z' < \mathrm{pow}_2(m)$$
$$w' \geq 0 \wedge w' < \mathrm{pow}_2(n)$$

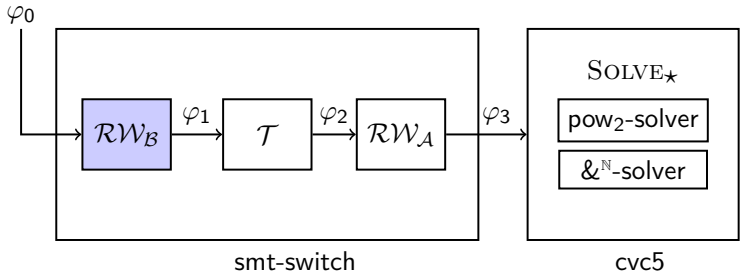
Type(φ) : $k \approx m + n \wedge k > 0 \wedge m > 0 \wedge n > 0$

where $y' = \chi(y)$, $z' = \chi(z)$, $w' = \chi(w)$.

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



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



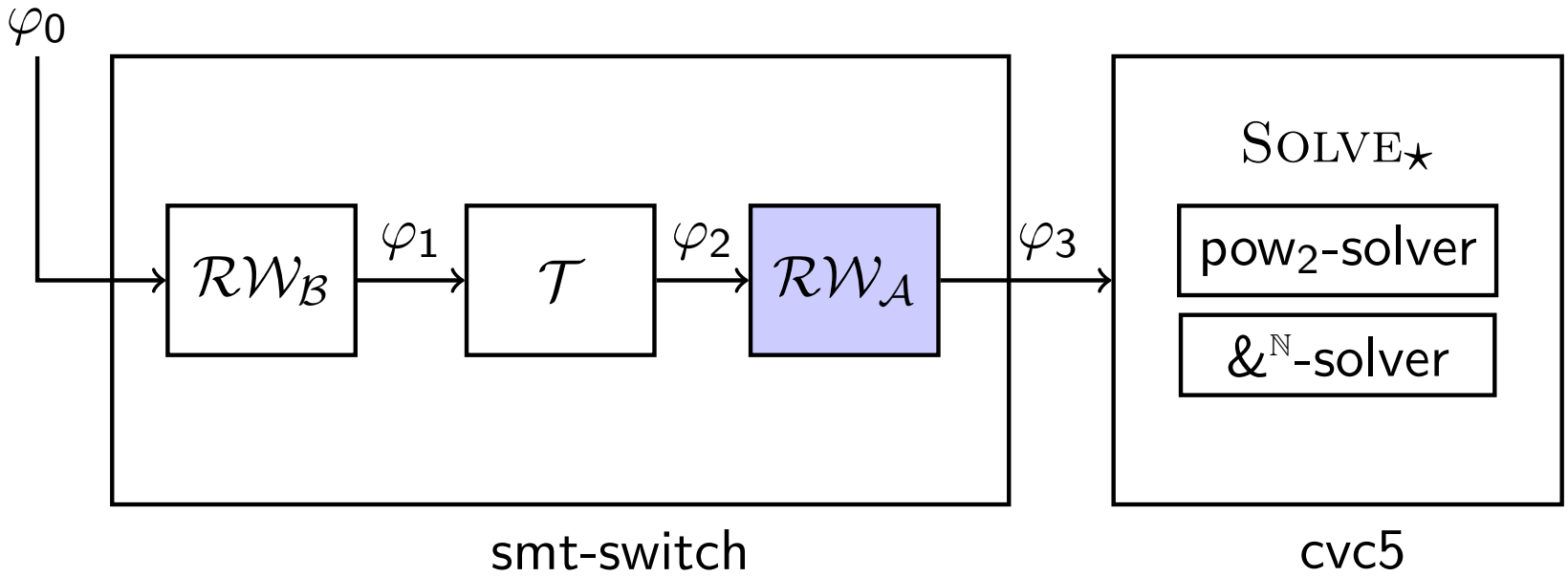
- SMT solvers implement many rewrite rules for T_{BV}.
- In cvc5, rules are documented in RARE[2].
- Implemented 47 rewrite rules useful for the T_{PBV} theory.
- In some cases, we implement their reverse.

Example

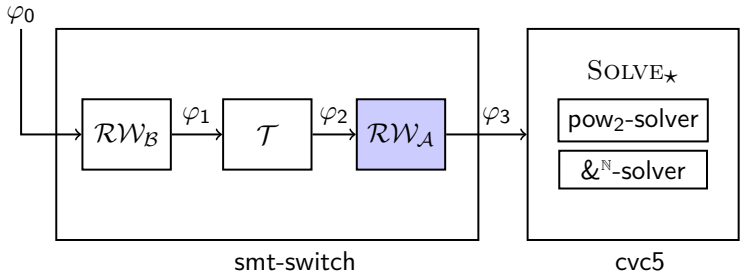
```
(define-rule bv-or-zero
((x ?BitVec) (n Int))
(bvor x (@bv 0 n))
x)
```

[2] Nötzli, Andres, et al. "Reconstructing Fine-Grained Proofs of Rewrites Using a Domain-Specific Language." FMCAD 2022.

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



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

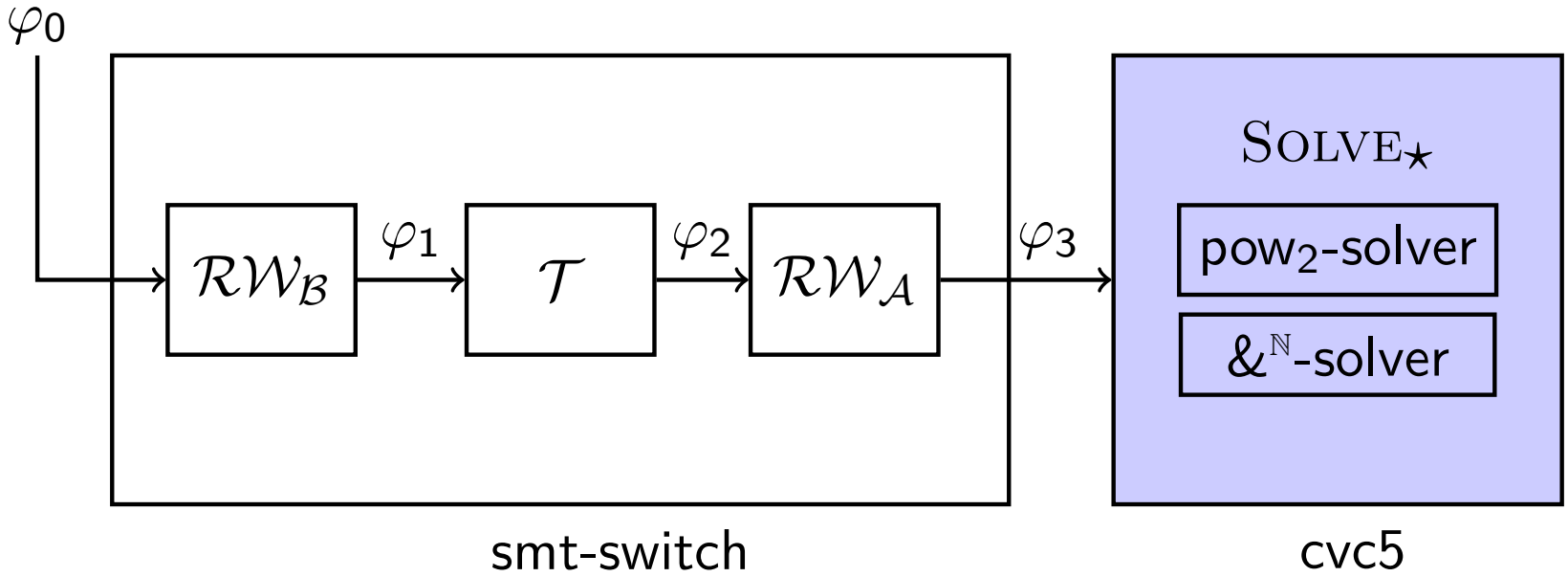


- Rewrite rules to eliminate modulo operations:

$$
\begin{array}{lll}
+^1_{rr} & ((x \bmod \mathrm{pow}_2(k)) + (y \bmod \mathrm{pow}_2(k))) \bmod \mathrm{pow}_2(k) & \to \quad (x+y) \bmod \mathrm{pow}_2(k) \\
+^2_{rr} & ((x \bmod \mathrm{pow}_2(k)) + y) \bmod \mathrm{pow}_2(k) & \to \quad (x+y) \bmod \mathrm{pow}_2(k) \\
+^3_{rr} & (x + (y \bmod \mathrm{pow}_2(k))) \bmod \mathrm{pow}_2(k) & \to \quad (x+y) \bmod \mathrm{pow}_2(k) \\
\hline
-^1_{rr} & ((x \bmod \mathrm{pow}_2(k)) - (y \bmod \mathrm{pow}_2(k))) \bmod \mathrm{pow}_2(k) & \to \quad (x-y) \bmod \mathrm{pow}_2(k) \\
-^2_{rr} & ((x \bmod \mathrm{pow}_2(k)) - y) \bmod \mathrm{pow}_2(k) & \to \quad (x-y) \bmod \mathrm{pow}_2(k) \\
-^3_{rr} & (x - (y \bmod \mathrm{pow}_2(k))) \bmod \mathrm{pow}_2(k) & \to \quad (x-y) \bmod \mathrm{pow}_2(k) \\
\hline
\cdot^1_{rr} & ((x \bmod \mathrm{pow}_2(k)) \cdot (y \bmod \mathrm{pow}_2(k))) \bmod \mathrm{pow}_2(k) & \to \quad (x \cdot y) \bmod \mathrm{pow}_2(k) \\
\cdot^2_{rr} & ((x \bmod \mathrm{pow}_2(k)) \cdot y) \bmod \mathrm{pow}_2(k) & \to \quad (x \cdot y) \bmod \mathrm{pow}_2(k) \\
\cdot^3_{rr} & (x \cdot (y \bmod \mathrm{pow}_2(k))) \bmod \mathrm{pow}_2(k) & \to \quad (x \cdot y) \bmod \mathrm{pow}_2(k)
\end{array}
$$

- cvc5 does not implement them: modulo assumed positive.

PBV Solver - SOLVE$_\star$

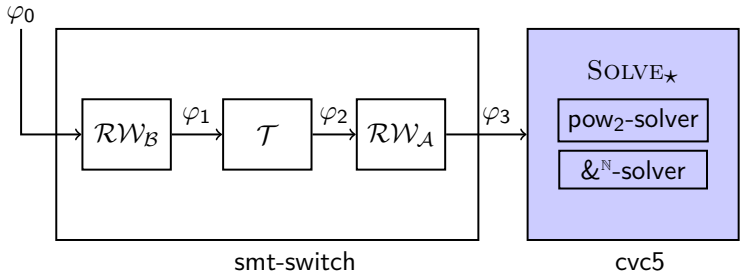


Lazy vs. Eager Algorithm

- Eager: adds all axioms upfront.
- Lazy: adds lemmas only when needed.
- Lazy algorithm uses CEGAR[3] to add lemmas.
- The key difference between the two CEGAR algorithms is which lemmas are added.

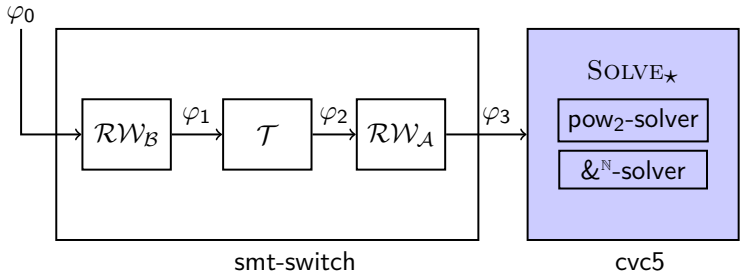
[3] Clarke et al., "Counterexample-Guided Abstraction Refinement", CAV 2000.

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

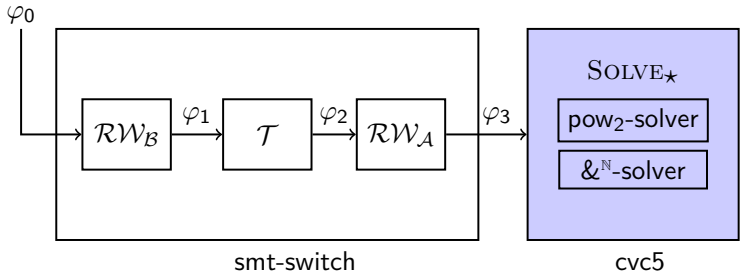
CEGAR Procedure



function SOLVE$_\star(\varphi)$
  $\Gamma \leftarrow \{\varphi\}$
  loop
    $result, \mathcal{I} \leftarrow$ SOLVE(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.

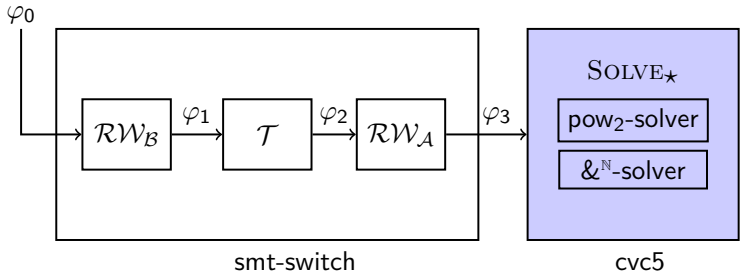
CEGAR Procedure



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

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

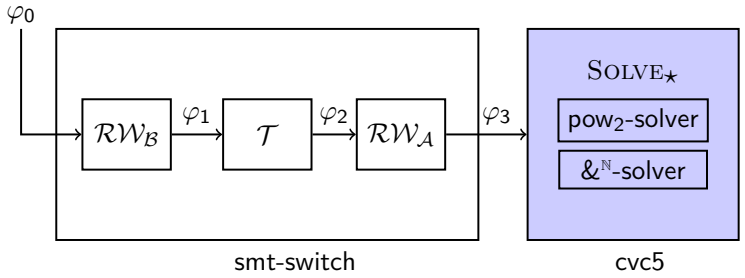
CEGAR Procedure



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 $\Rightarrow$ formula is satisfiable.

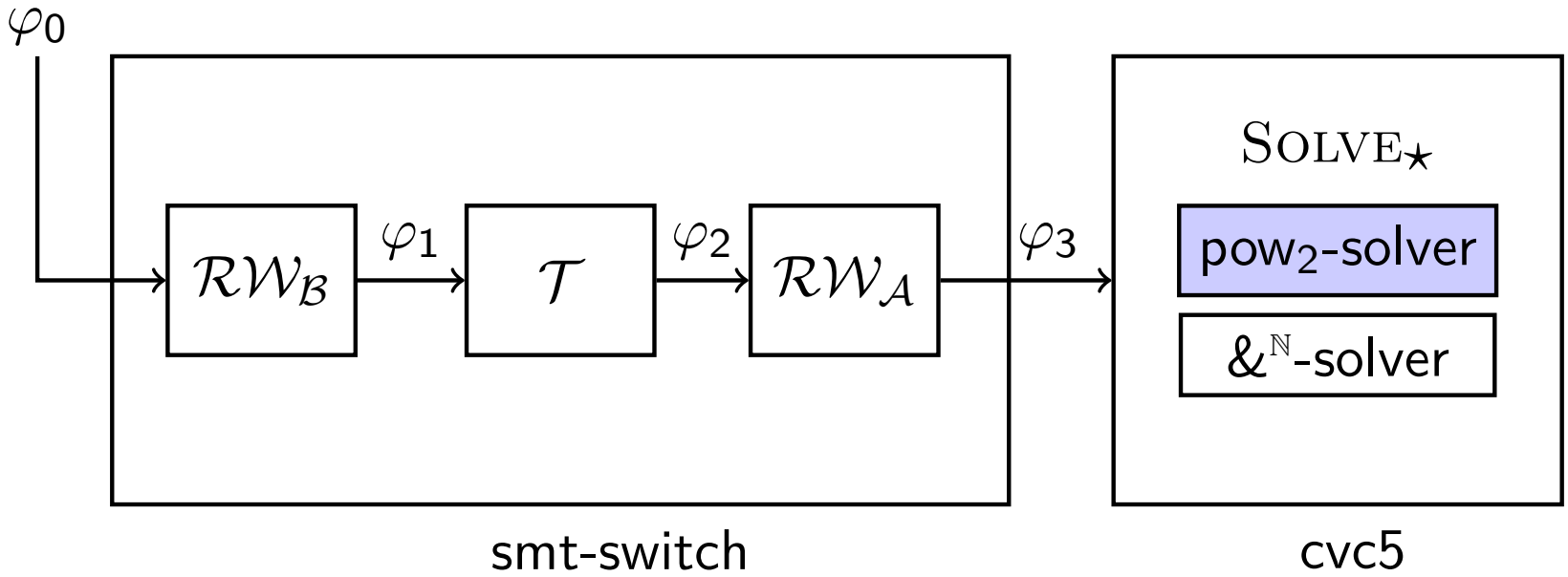
CEGAR Procedure



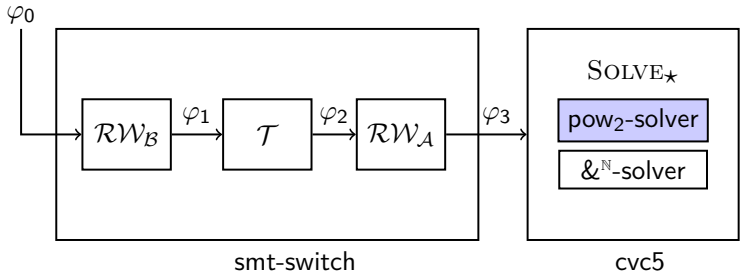
function SOLVE$_\star(\varphi)$
  $\Gamma \leftarrow \{\varphi\}$
  loop
    $result, \mathcal{I} \leftarrow$ SOLVE(Γ)
    if *result* is "unsat" **return** "unsat"
    $\mathcal{L} \leftarrow \mathcal{L}_{\&^\mathbb{N}}(\{\&^\mathbb{N}(k, t_1, t_2) \mid \&^\mathbb{N}(k, t_1, t_2) \in \mathit{Terms}(\Gamma)\}, \mathcal{I})$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_{\mathsf{pow}_2}(\{\mathsf{pow}_2(t) \mid \mathsf{pow}_2(t) \in \mathit{Terms}(\mathcal{L} \cup \Gamma)\}, \mathcal{I})$
    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.

PBV Solver - pow_2



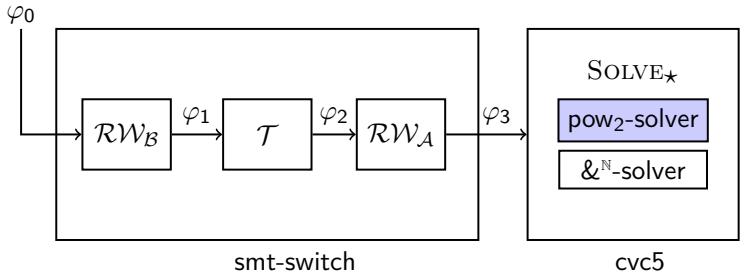
pow_2-solver



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$_\star$.

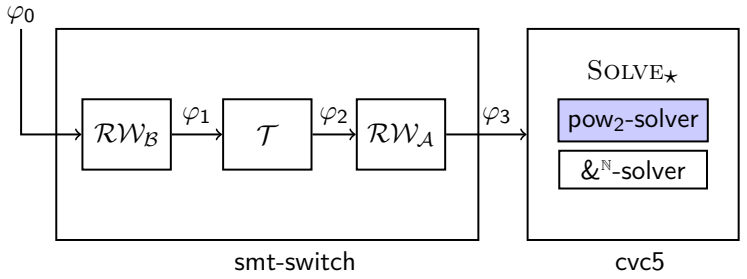
bound Lemma

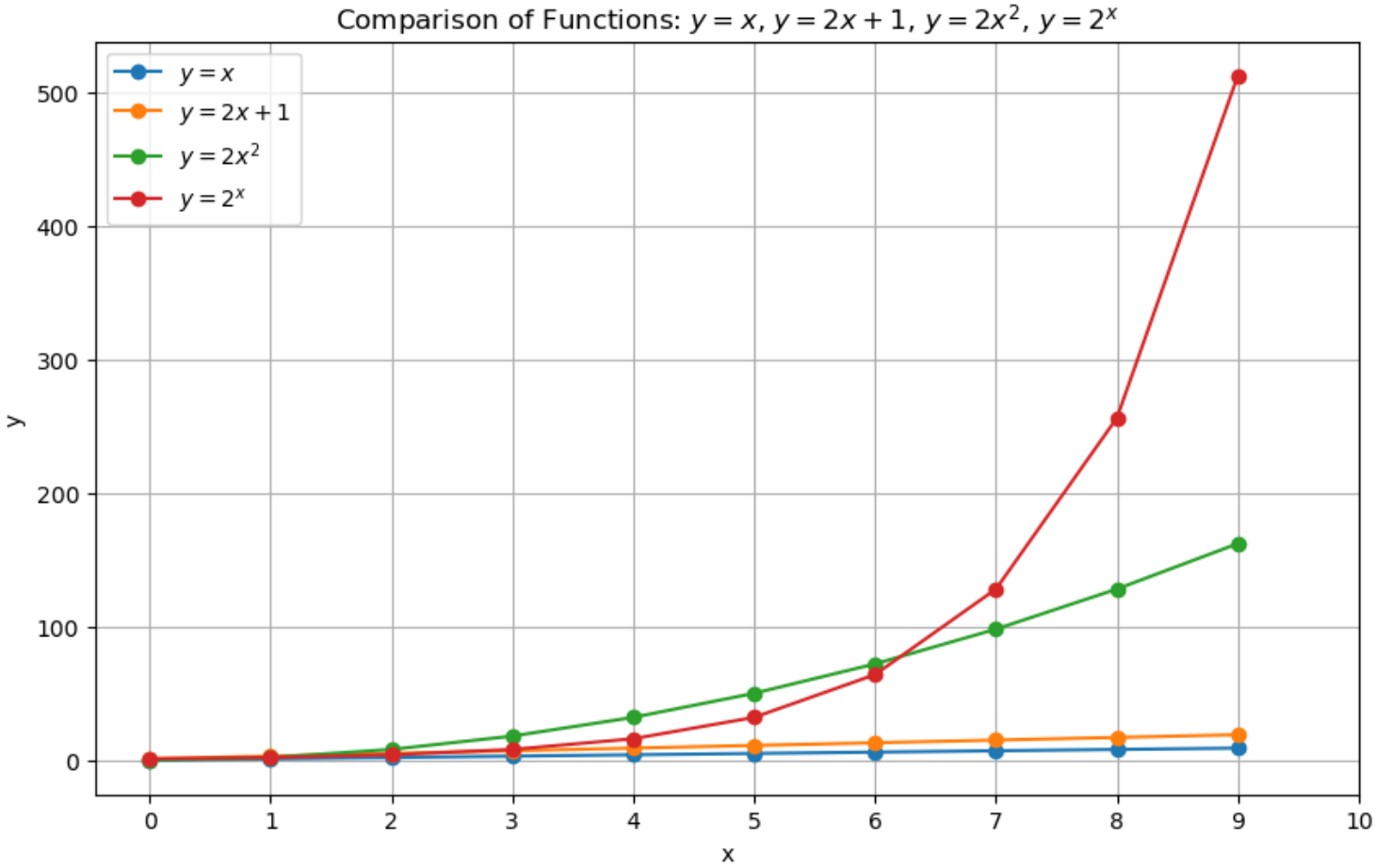


- Naive bound lemma: $x < \text{pow}_2(x)$.
- More efficent from [IJCAR'24][4]:
$$x \geq 3 \Rightarrow 2 \cdot x + 1 < \text{pow}_2(x).$$
- General to polinomial bound:
$$x \geq 7 \Rightarrow 2 \cdot x^2 < \text{pow}_2(x).$$
- Using v to obtain a linear bound:
$$(x \geq v \wedge v \geq 7) \Rightarrow v \cdot x + v^2 < \text{pow}_2(x).$$

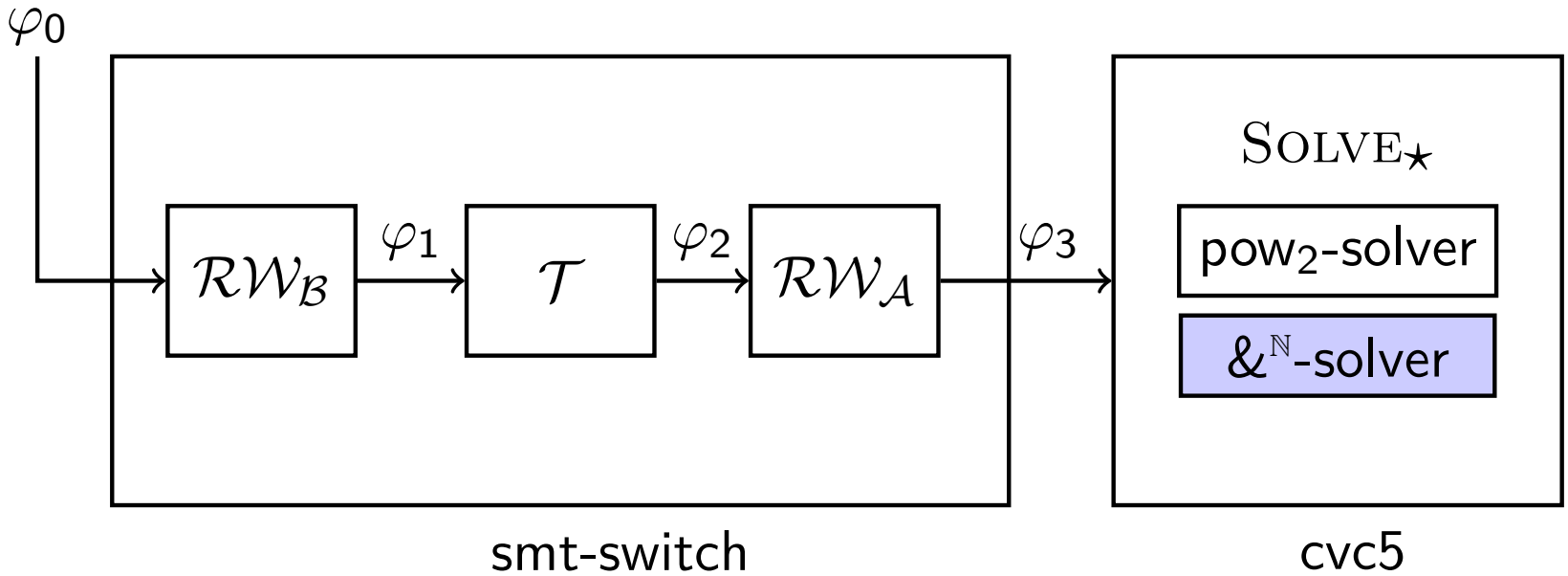
[4] Frohn, Florian, and Jürgen Giesl. "Satisfiability Modulo Exponential Integer Arithmetic." IJCAR 2024.

bound Lemma

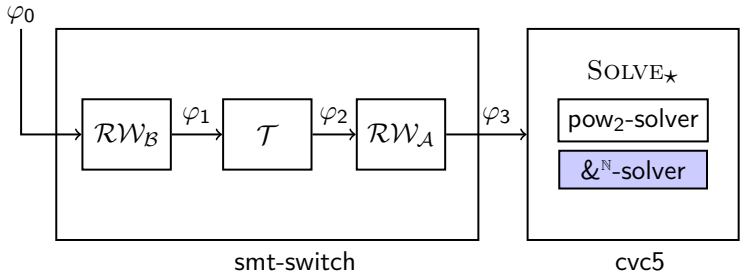

φ_0
$\mathcal{RW}_{\mathcal{B}}$
φ_1
$\mathcal{T}$
φ_2
$\mathcal{RW}_{\mathcal{A}}$
φ_3
SOLVE$_\star$
pow$_2$-solver
&$^{\mathbb{N}}$-solver
smt-switch
cvc5


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

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

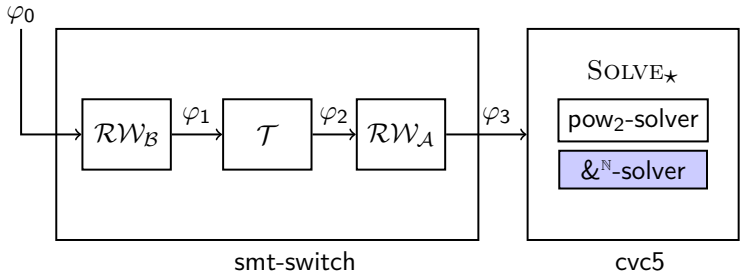


$\&^{\mathbb{N}}$-solver



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	$0 \leq x \wedge 0 \leq y \Rightarrow 0 \leq \&^{\mathbb{N}}(k,x,y) \leq \min(x,y)$	[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]

Bitwise AND Consistency with Zero Extension



Original (3 bits)

A: 1 0 1

B: 1 1 0

AND

Result: 1 0 0

Extended (5 bits)

Original bits

A: 0 0 1 0 1

B: 0 0 1 1 0

AND

Same result

Result: 0 0 1 0 0

Outline

Benchmark Sets

. **Compiler Optimizations**

- *alive* (200 benchmarks)

Crafted benchmarks

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

SMT solvers internals

- *ic* (180 benchmarks)
- *icfb* (46 benchmarks)
- *lemmas* (70 benchmarks)
  - quantified benchmarks

Mutated Benchmarks

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

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	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓
$\mathcal{RW}_{\mathcal{A}}$	✗	✓	✓

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	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓
$\mathcal{RW}_{\mathcal{A}}$	✗	✓	✓

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	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓
$\mathcal{RW}_{\mathcal{A}}$	✗	✓	✓

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	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓
$\mathcal{RW}_{\mathcal{A}}$	✗	✓	✓

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	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓
$\mathcal{RW}_{\mathcal{A}}$	✗	✓	✓

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	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓
$\mathcal{RW}_{\mathcal{A}}$	✗	✓	✓

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

- 45 benchmarks added to *alive*, 12 to *ic*.
- First verification of **shift** optimizations in *alive*.
- First verification of **concatenation** benchmarks in *ic*.
- First verification of *lemmas* and *icfb* sets.
  - Found a known typo regarding *icfb* from [FMCAD'20][5].
- 23 benchmarks in *lemmas*, 12 in *icfb*.

[5] Niemetz, Aina, and Mathias Preiner. "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	✗	✓	✓
$\mathcal{RW}_{\mathcal{B}}$	✗	✓	✓
$\mathcal{RW}_{\mathcal{A}}$	✗	✓	✓

Summary and Future Work

- **Summary:**
  - New theory for parametric bit-vectors
  - Ability to reason about bit-widths
  - Lazy algorithm for parametric bit-vectors
  - New tool using the lazy approach shows significant improvement
- **Future Work:**
  - Explore proof production for parametric bit-vectors.
  - Investigate the scalability of our approach.

Questions?

Thank you for listening!

Admissible Satisfiability – Satisfiable Example

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

Example

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

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

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

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.

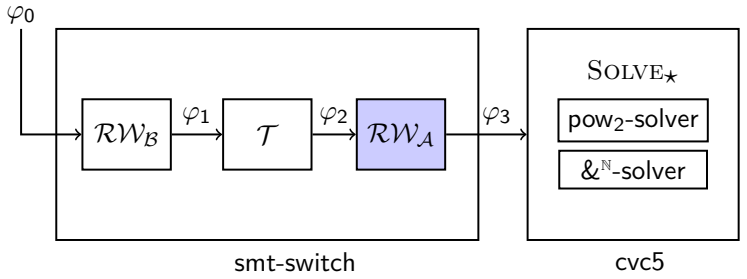
Parametric Bit Vector Rewriter – $\mathcal{RW}_{\mathcal{B}}$ (Part 1)

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 – $\mathcal{RW}_{\mathcal{A}}$



Example

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

$\mathrm{C}(\varphi)$:

$$(((x' + x') \mathrm{mod} 2^k) + x') \bmod 2^k$$

$\mathcal{RW}_{\mathcal{A}}$ $(\mathrm{C}(\varphi))$:

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

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

Evaluation on *alive*

Family	Considered	Proved			
		[6]	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

[6] Niemetz, Aina, et al. 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:
  - $\& \leftrightarrow |$
  - $+^{\mathcal{B}} \leftrightarrow -^{\mathcal{B}}$
  - $-^{\mathcal{B}} \leftrightarrow \sim$
  - $\ll \leftrightarrow \gg$
  - $\mathit{topbv}(k, 1) \leftrightarrow \mathit{topbv}(k, 0)$
  - $\mathit{topbv}(k, k) \leftrightarrow \mathit{topbv}(k, 0)$
- 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][7]:

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

- 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.

[7] Niemetz, Aina, and Mathias Preiner. "Ternary propagation-based local search for more bit-precise reasoning." FMCAD 2020.

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.

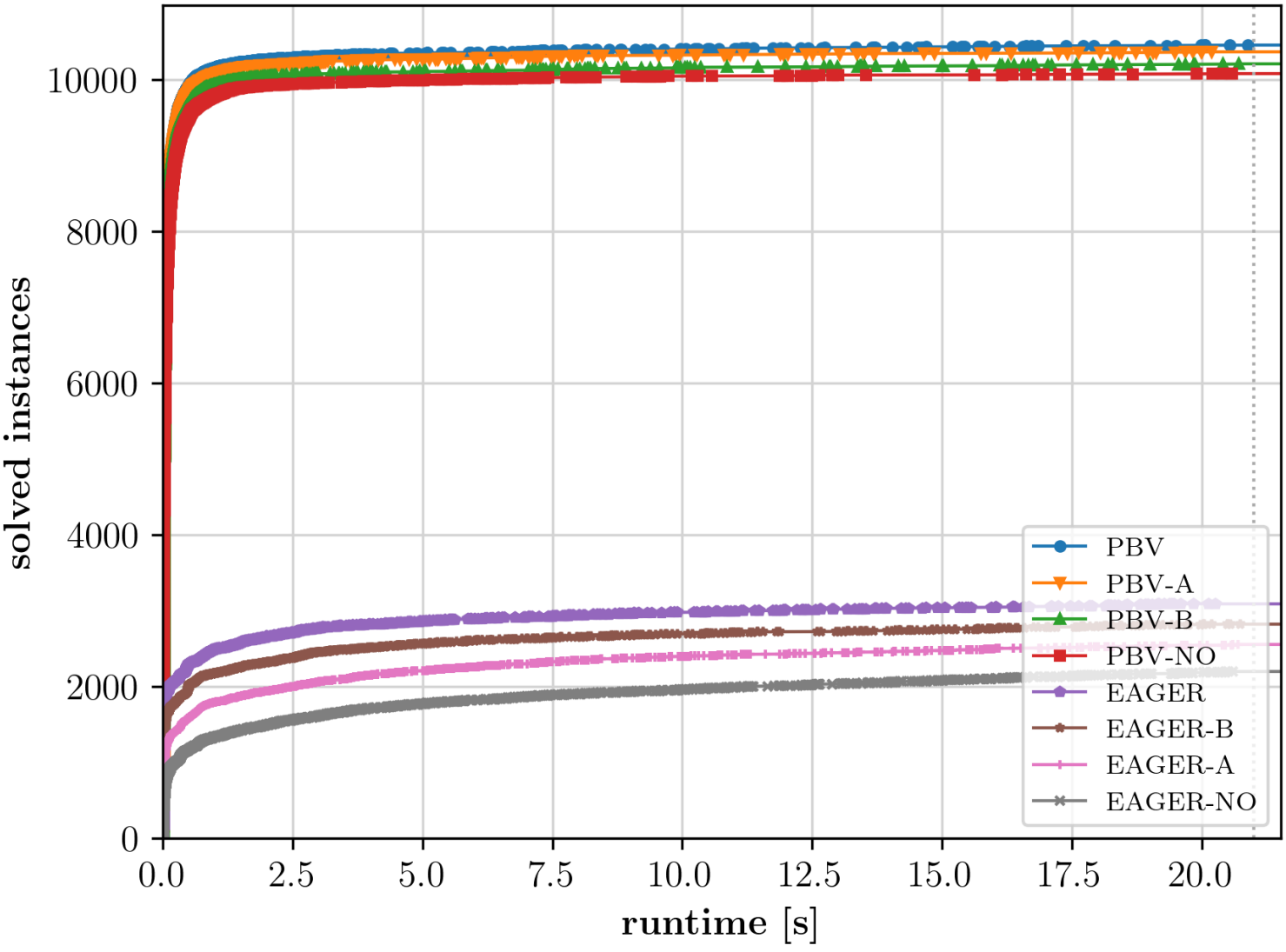
Evaluation Attributes Results

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