Bit-Precise Interpolation in Bitwuzla

Aina Niemetz and Mathias Preiner

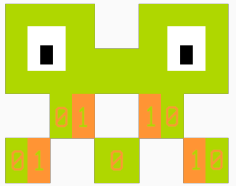
Stanford University

TACAS, April 14, 2026

Motivation

Bitwuzla

- an SMT solver for bit-precise reasoning
- fixed-size bit-vectors, floating-point arithmetic
  + arrays, uninterpreted functions
  + with and without quantifiers

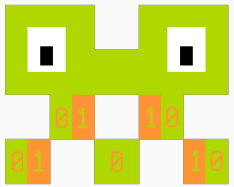


https://bitwuzla.github.io

Motivation

Bitwuzla

- an SMT solver for bit-precise reasoning
- fixed-size bit-vectors, floating-point arithmetic
  + arrays, uninterpreted functions
  + with and without quantifiers



https://bitwuzla.github.io

Interpolation

- bit-precise reasoning at the back end of verification engines
- prominent example: symbolic model checking
- lifting interpolation-based model checking procedures from SAT to SMT

➤ interpolation queries not standardized in SMT-LIB

➤ not many SMT solvers support interpolation

➤ Bitwuzla now with **full** interpolation support for QF_BV

Craig Interpolant

Given **unsatisfiable** formula $\varphi := A \wedge B$.
An interpolant of (A, B) is a formula I such that

(i) $A \Rightarrow I$ is **valid**
(ii) $I \wedge B$ is **unsat**
(iii) $\mathcal{S}(I) \subseteq \mathcal{S}(A) \cap \mathcal{S}(B) = \mathcal{S}_G$

Craig Interpolant

Given **unsatisfiable** formula $\varphi := A \wedge B$.
An interpolant of (A, B) is a formula I such that

(i) $A \Rightarrow I$ is **valid**
(ii) $I \wedge B$ is **unsat**
(iii) $\mathcal{S}(I) \subseteq \mathcal{S}(A) \cap \mathcal{S}(B) = \mathcal{S}_G$

A/B-Labeling

$\mathcal{S}(A)$ ... symbols **local** to A
$\mathcal{S}(B)$ ... symbols **local** to B

$\mathcal{S}_G = \mathcal{S}(A) \cap \mathcal{S}(B)$ ... **global** symbols

Interpolation

Craig Interpolant

Given **unsatisfiable** formula $\varphi := A \wedge B$.
An interpolant of (A, B) is a formula I such that

(i) $A \Rightarrow I$ is **valid**
(ii) $I \wedge B$ is **unsat**
(iii) $\mathcal{S}(I) \subseteq \mathcal{S}(A) \cap \mathcal{S}(B) = \mathcal{S}_G$

A/B-Labeling

$\mathcal{S}(A)$ ... symbols **local** to A
$\mathcal{S}(B)$ ... symbols **local** to B

$\mathcal{S}_G = \mathcal{S}(A) \cap \mathcal{S}(B)$ ... **global** symbols

Inductive Interpolation Sequence

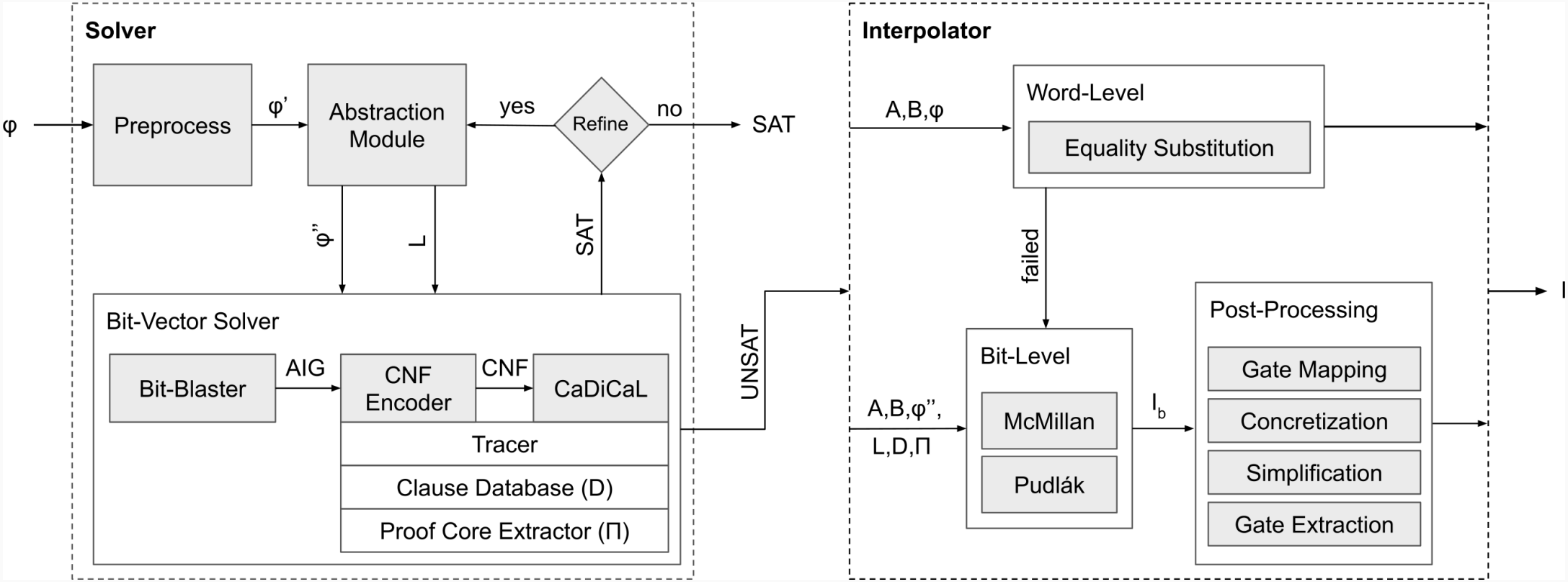
Given **unsatisfiable** formula $\varphi = \{\varphi_1, ..., \varphi_n\}$ and a **sequence of partitions** $\langle (A_1, B_1), ..., (A_n, B_n) \rangle$ with $A_i = \{\varphi_1, ..., \varphi_i\}$ and $B_i = \{\varphi_{i+1}, ..., \varphi_n\}$.
An interpolation sequence $\langle I_0, ..., I_n \rangle$ is a sequence of formulas I_i such that

(i) $I_0 = \top$ and $I_n = \bot$
(ii) I_i is an interpolant for (A_i, B_i)
(iii) $I_i \wedge \varphi_{i+1} \Rightarrow I_{i+1}$

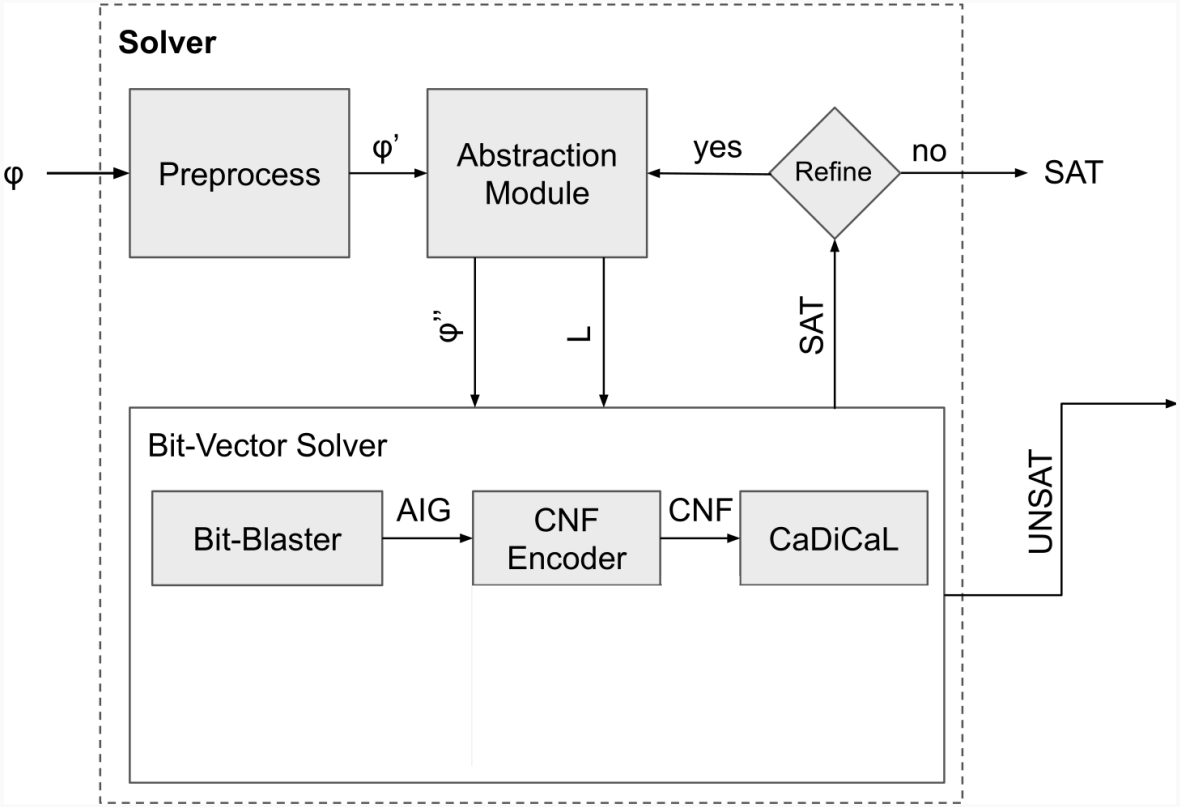
Interpolation in Bitwuzla

- **full** interpolation support for QF_BV
  - limited support for combinations with arrays, UF, FP
- **single** interpolation queries and **inductive interpolation sequences**
- **SMT-LIB** extension and via the **API**
  - `(get-interpolants (a1 a2))` single query
  - `(get-interpolants (a1) (a2) (a3))` interpolation sequence
- two interpolation **modes**
  - compute bit-level interpolants from SAT proofs, post-process and lift to word-level
  - compute word-level interpolants

Interpolation Workflow



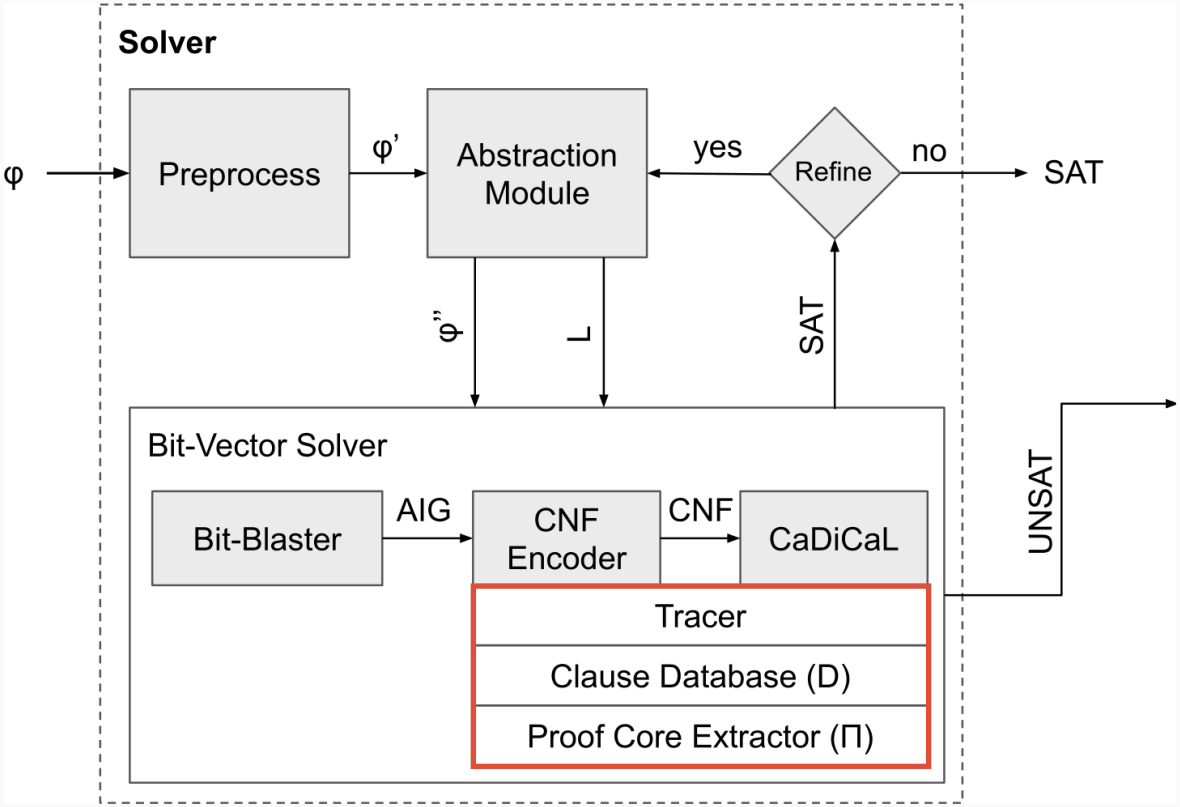
Interpolation Workflow



- Lemmas on demand (CEGAR) over bit-vector abstraction
- bit-vector solver at the core
- main technique: bit-blasting

  ➜ reduction to SAT

Interpolation Workflow

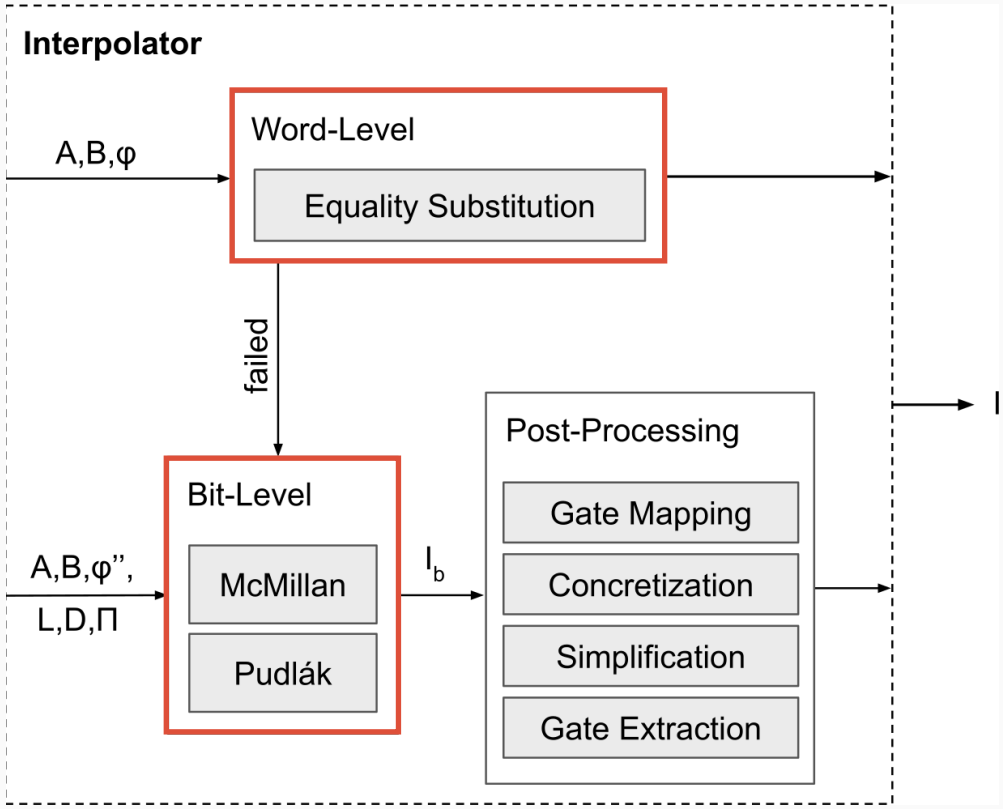


- interpolant generation from SAT proof core
- one solve call for multiple interpolation queries
- dynamic A/B partitioning

Interpolation Modes

Bit-Level

- construct I_b from proof core as AIG
  - McMillan (default) or Pudlàk construction
  - A/B-labeling via mapping from BV to AIG to CNF



Interpolation Modes

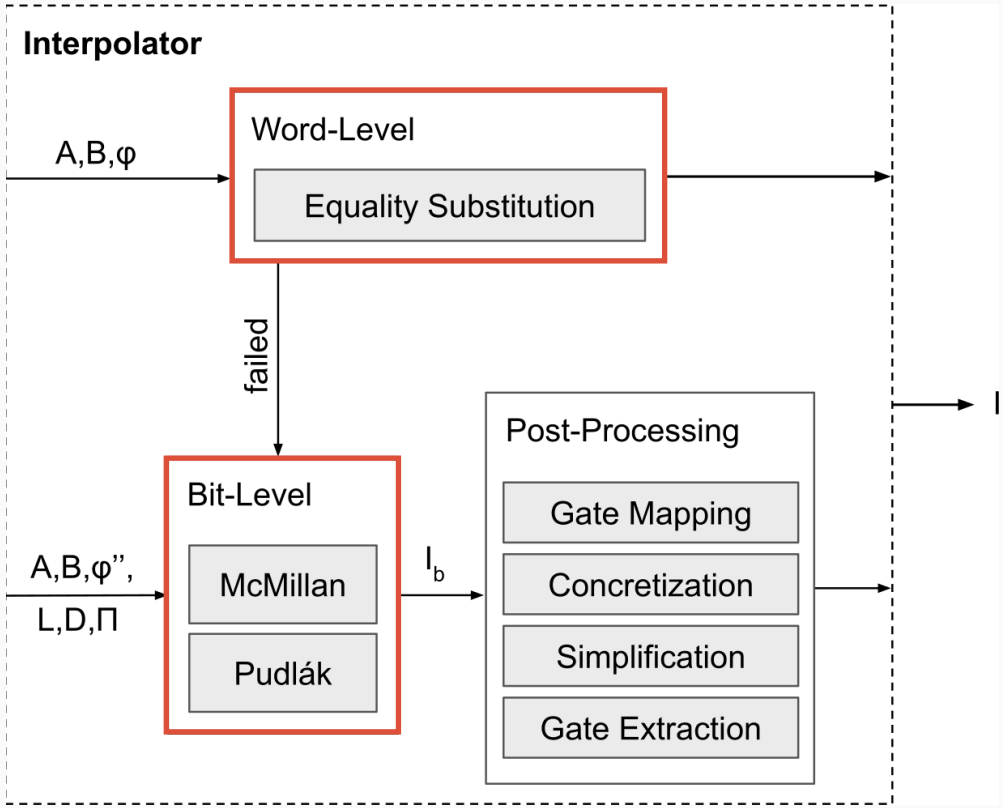
Bit-Level

- construct I_b from proof core as AIG
  - McMillan (default) or Pudlàk construction
  - A/B-labeling via mapping from BV to AIG to CNF

Word-Level (optional)

- equality substitution
- special case of quantifier elimination
  - substitution of local symbols:

$A[a/t]$ if $a = t \in A$ and $\mathcal{S}(A[a/t]) \cap \mathcal{S}(A) = \emptyset$
$\neg B[b/t]$ if $b = t \in B$ and $\mathcal{S}(B[b/t]) \cap \mathcal{S}(B) = \emptyset$



Bit-Level Interpolant Post-Processing

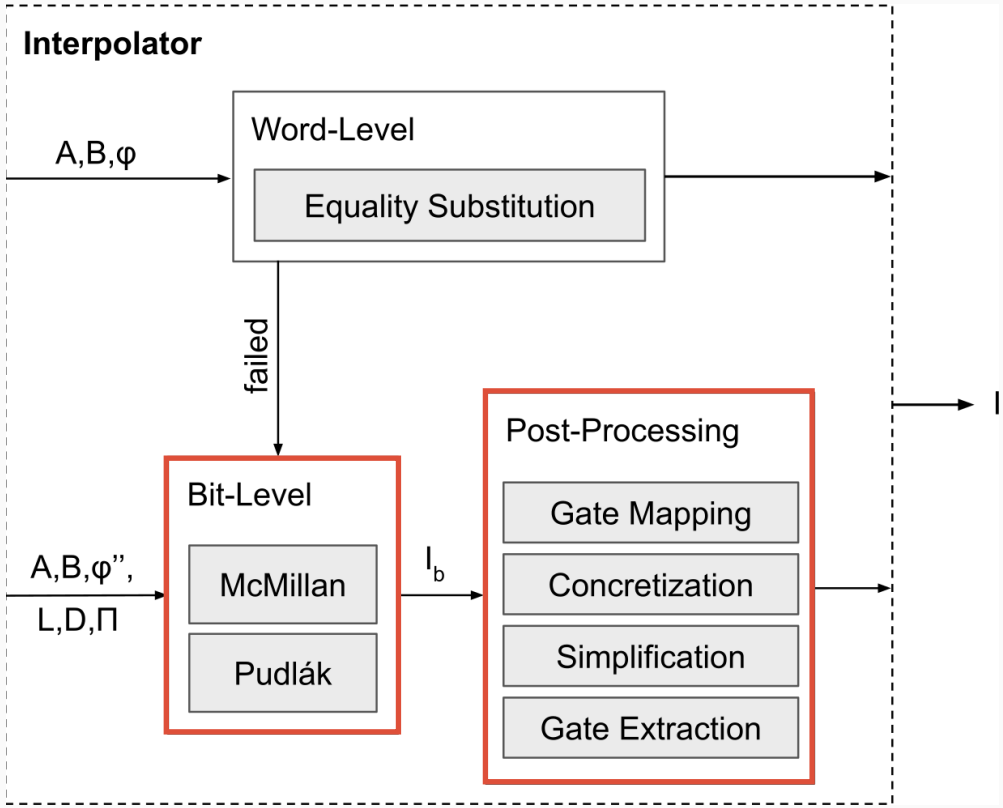
- **bit-level** interpolant I_b is constructed as AIG

↪ lift to word-level

- **lifted interpolant** still has a lot of bit-level structure

↪ post-process to recover structure

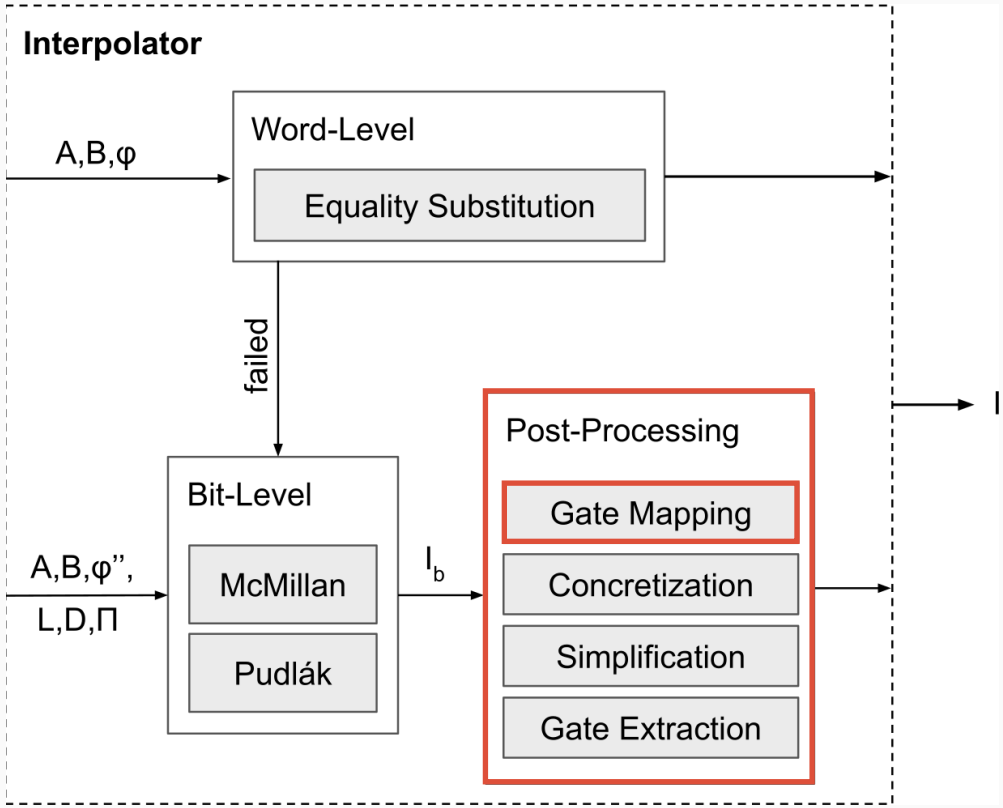
  - configurable
  - some interpolation-based algorithms (ic3ia) utilize structure



Bit-Level Interpolant Post-Processing

Gate Mapping

- lift I_b to word-level
  - map gates to nodes with only global symbols
  - reconstruct gates for nodes with local symbols



Bit-Level Interpolant Post-Processing

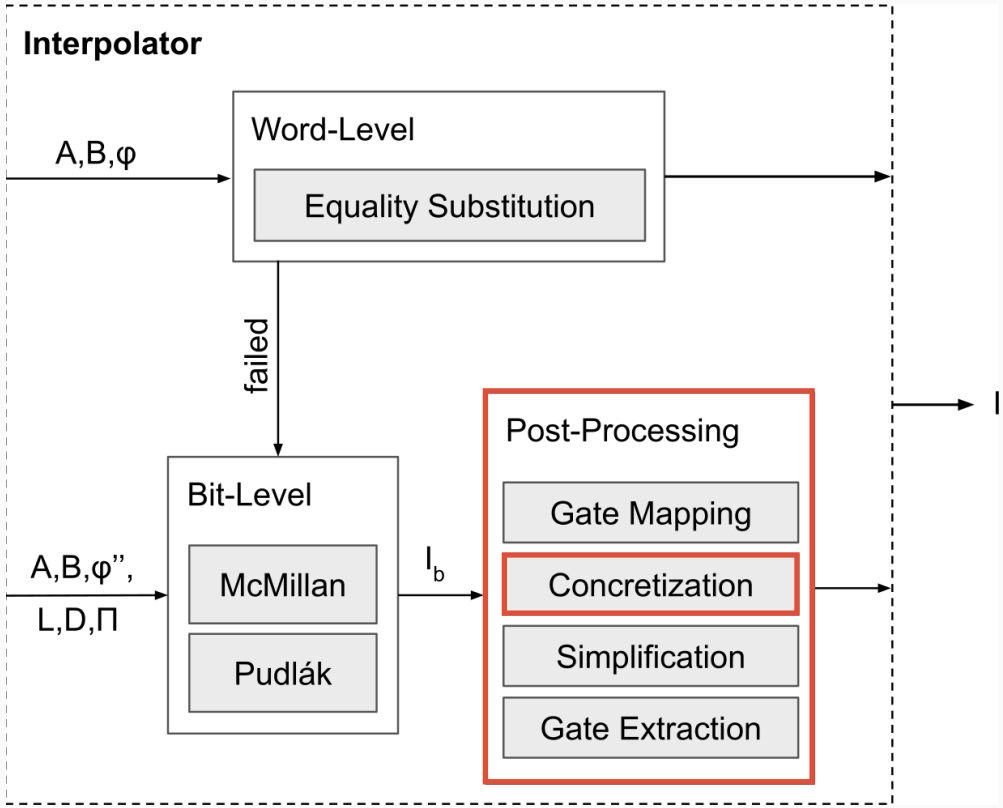
Gate Mapping

- lift I_b to word-level
  - map gates to nodes with only global symbols
  - reconstruct gates for nodes with local symbols

Concretization

- abstraction-refinement procedure for (large) bit-vector arithmetic operations
  - introduces fresh constants for abstracted terms

↪ concretize abstracted terms
  - replace abstractions with term they abstract



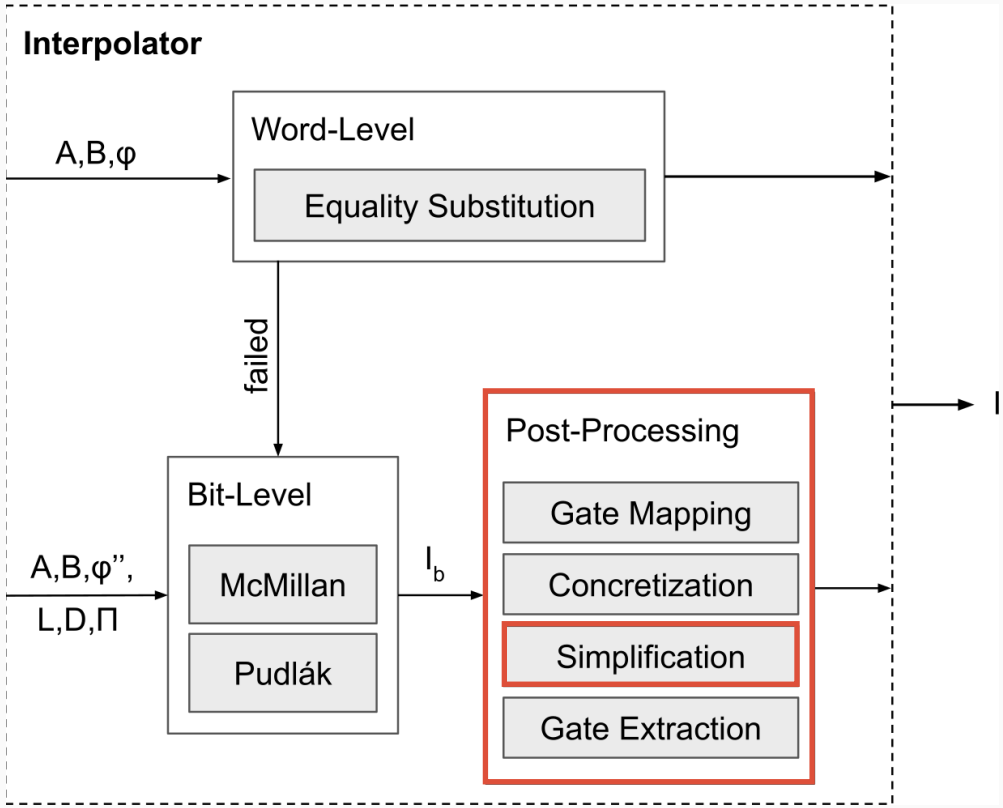
Bit-Level Interpolant Post-Processing

Simplification

- reduce size of interpolant

↪ simplify interpolant by utilizing Bitwuzla's preprocessor

  - term rewriting
  - term substitution
  - propositional skeleton preprocessing
  - ...



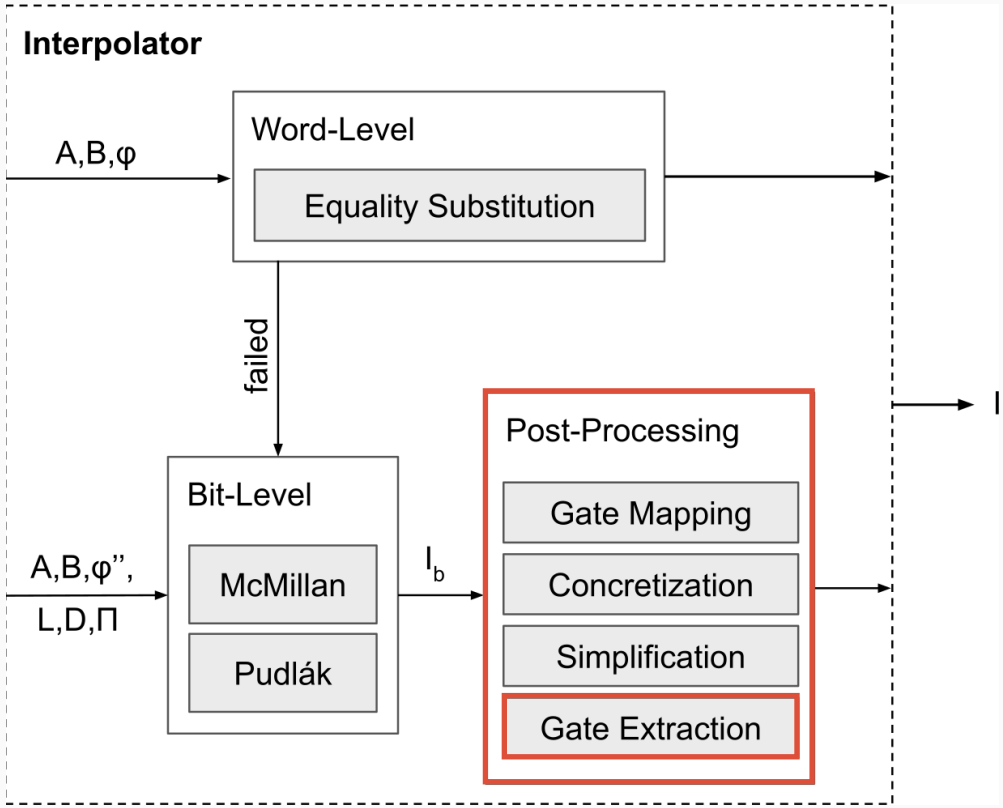
Bit-Level Interpolant Post-Processing

Gate Extraction

- recover as many word-level equalities over bit-ranges as possible

Example. $\sim(x[0] \oplus y[0])\ \&\ \sim(x[1] \oplus y[1])\ \&\ \sim(x[2] \oplus y[2])$ ⇝
$\texttt{cmp}(x[3:0], 1100)$
$\sim x[0]\ \&\ \sim x[1]\ \&\ x[2]\ \&\ x[3]$ ⇝
$\texttt{cmp}(x[3:0], 1100)$

- extract `xor`, `xnor` gates and bit-level comparisons and combine to comparisons over bit ranges
  ↪ **share aware**
- translate bitwise comparison back to equalities



Evaluation

➤ in the context of the Pono and Kind2 model checkers

- support **Bitwuzla** as solving engine, **MathSAT** as default interpolator

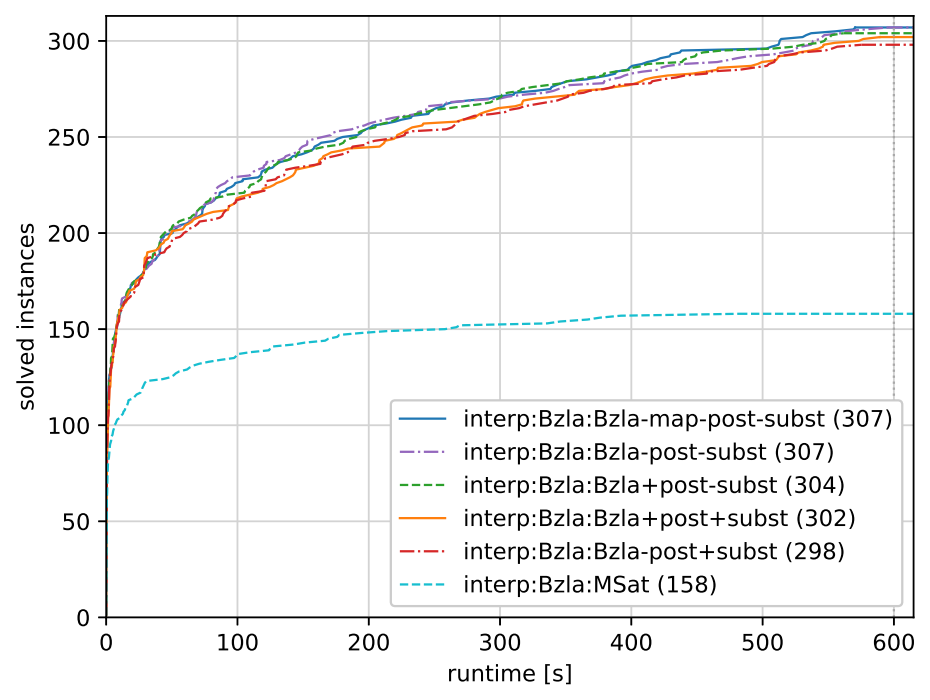
- Pono targets **HW verification**

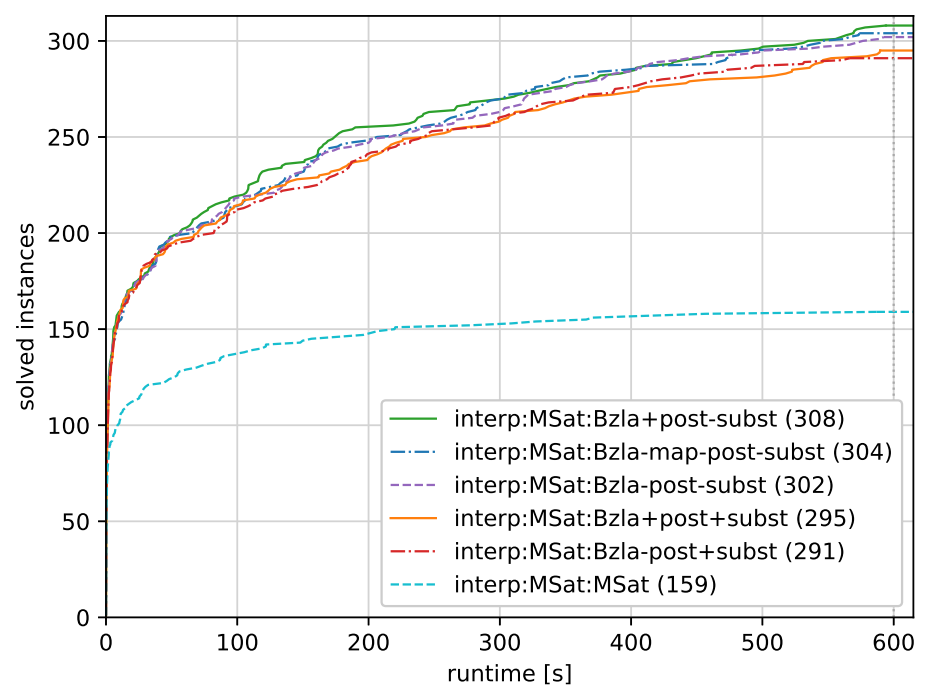
  ↪ implements 4 **interpolation-based** algorithms
  - interp [McMillan CAV'03]
  - ismc [VizelG FMCAD'09]
  - dar [VizelGS TACAS'13]
  - ic3ia [CimattiGMT TACAS'14]

- Kind2 targets **software verification**, implements ic3ia

➤ now also support **Bitwuzla** as interpolator

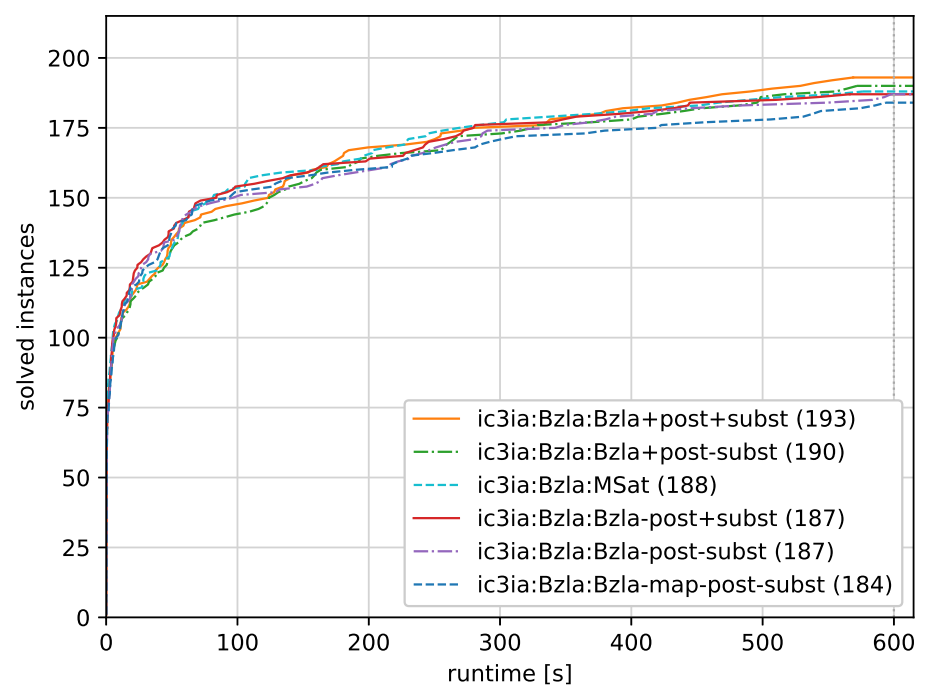
Evaluation: Pono Spotlight

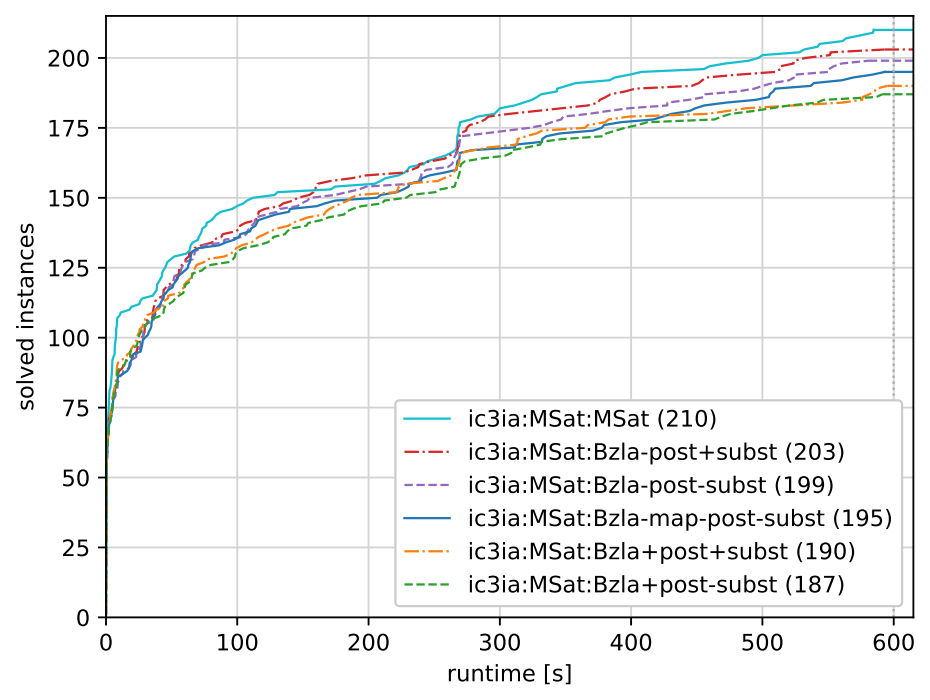




Pono with **interp** engine, **HWMCC 2019-2024 benchmarks** (839), 600s time and 16GB memory limit

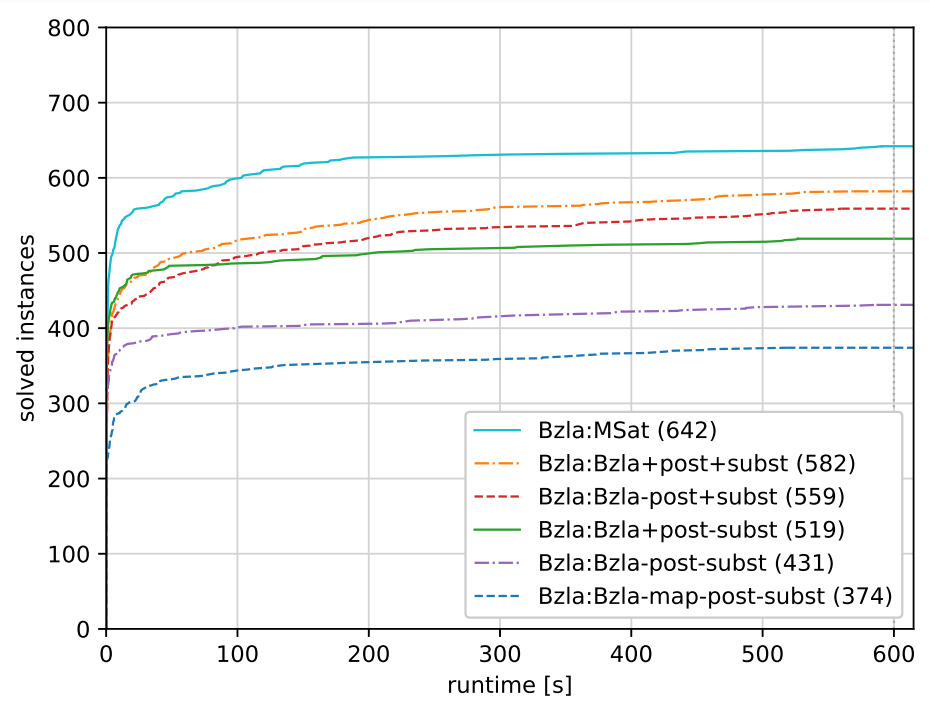
Evaluation: Pono Spotlight

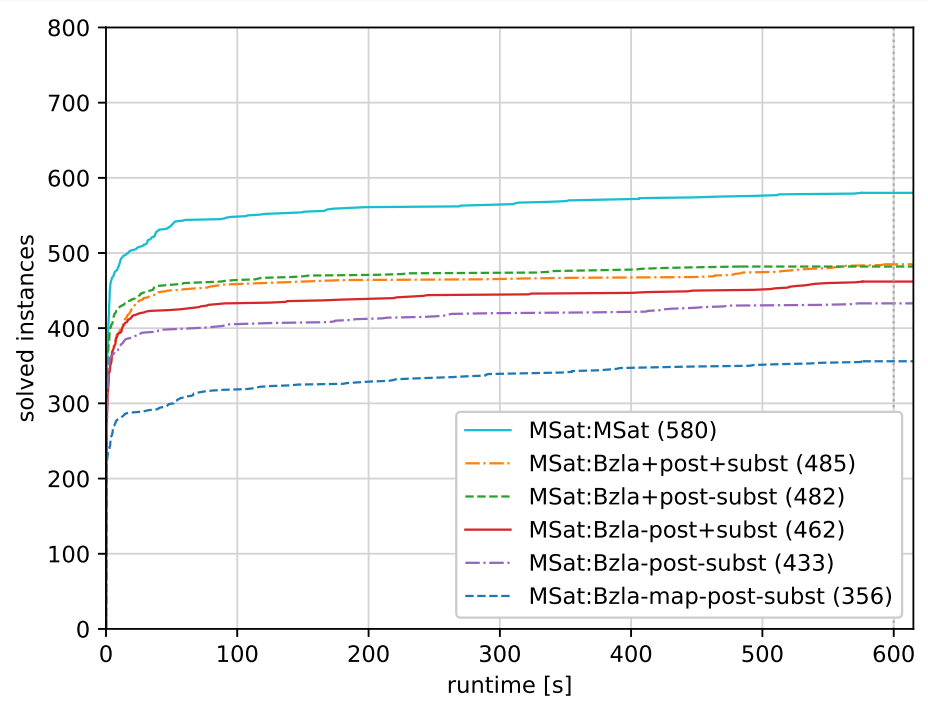




Pono with **ic3ia** engine, **HWMCC 2019-2024 benchmarks** (839), 600s time and 16GB memory limit

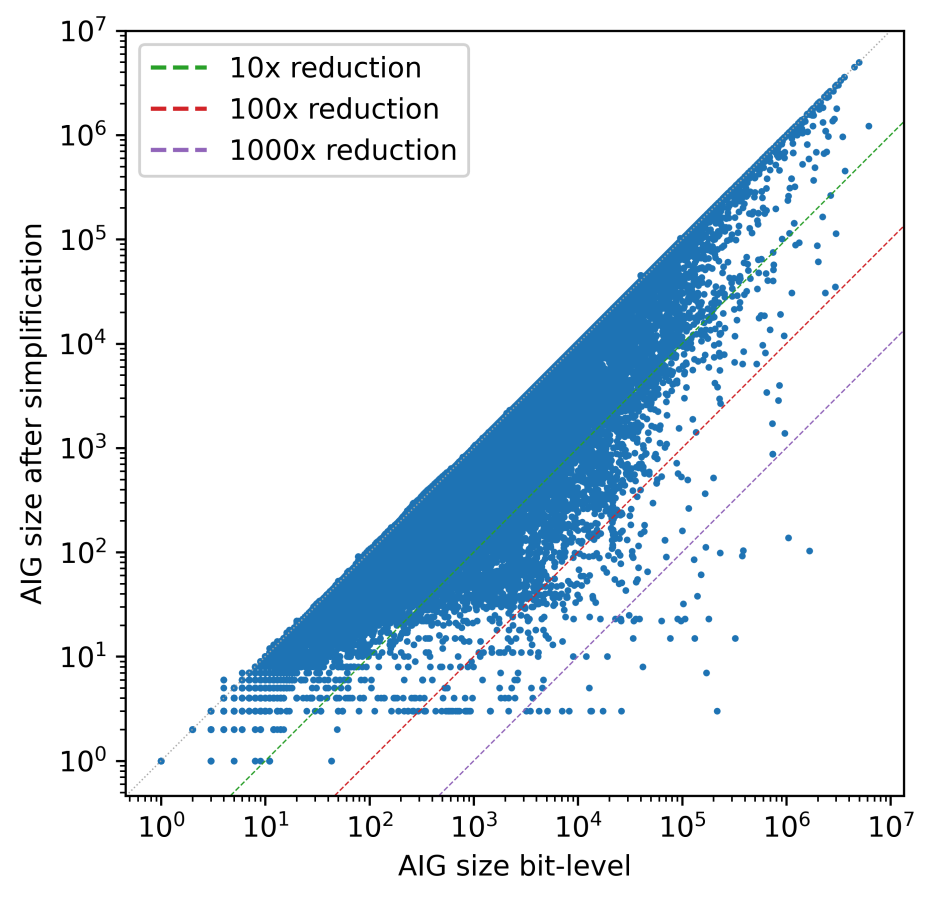
Evaluation: Kind2

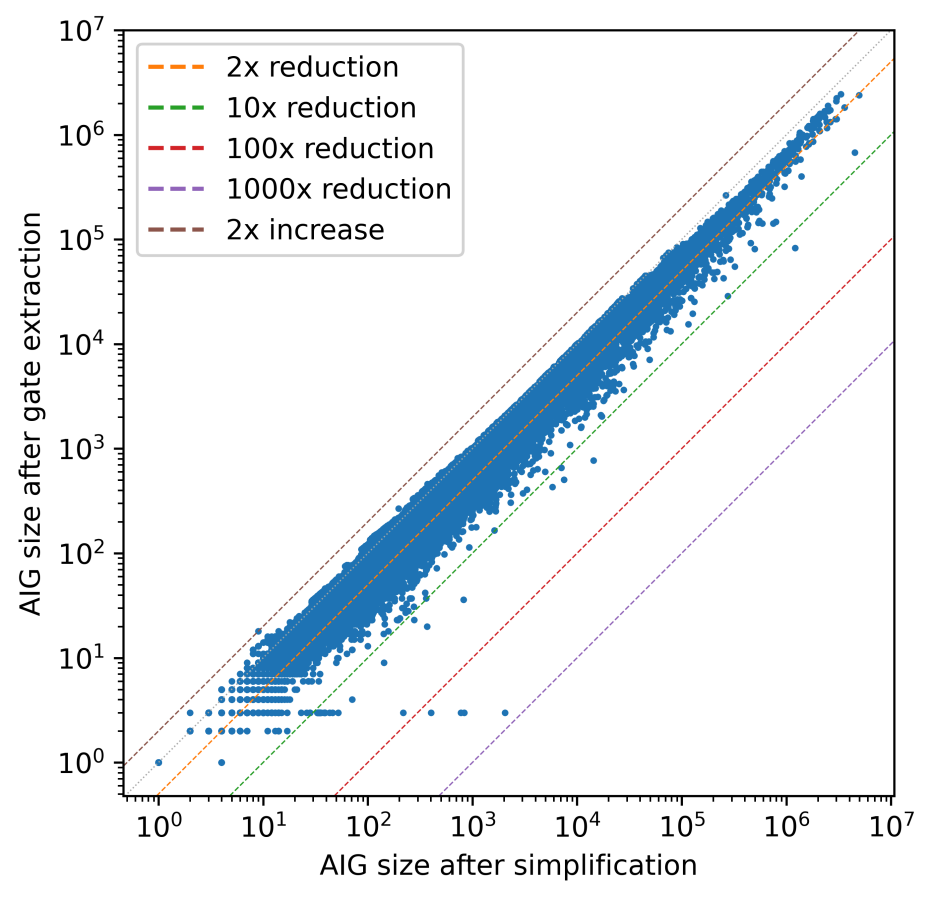




Kind2 with **ic3ia** engine, on **Lustre benchmarks** (786), 600s time and 16GB memory limit

Evaluation: Impact of Post-Processing



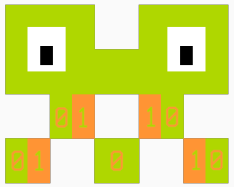


371,592 interpolation queries, size in terms of # and gates

Conclusion

- Bitwuzla now supports **interpolation** queries
  - **full** support for **quantifier-free bit-vector** formulas
  - **limited** support for combinations with arrays, UF, FP
    ↪ **full** support future work

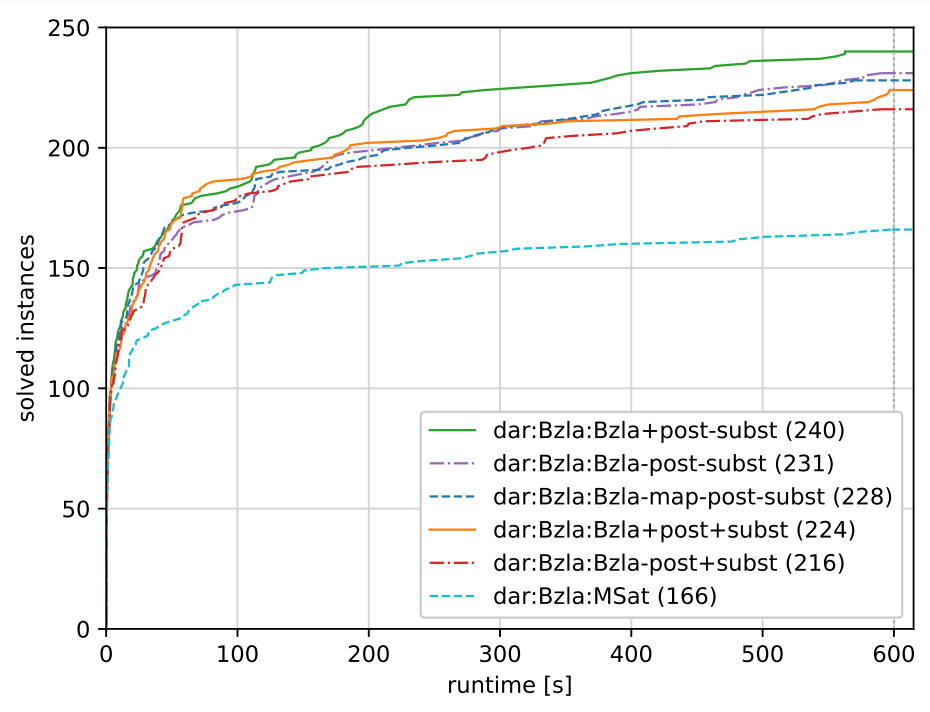
- **Artifact:** https://zenodo.org/record/17427360

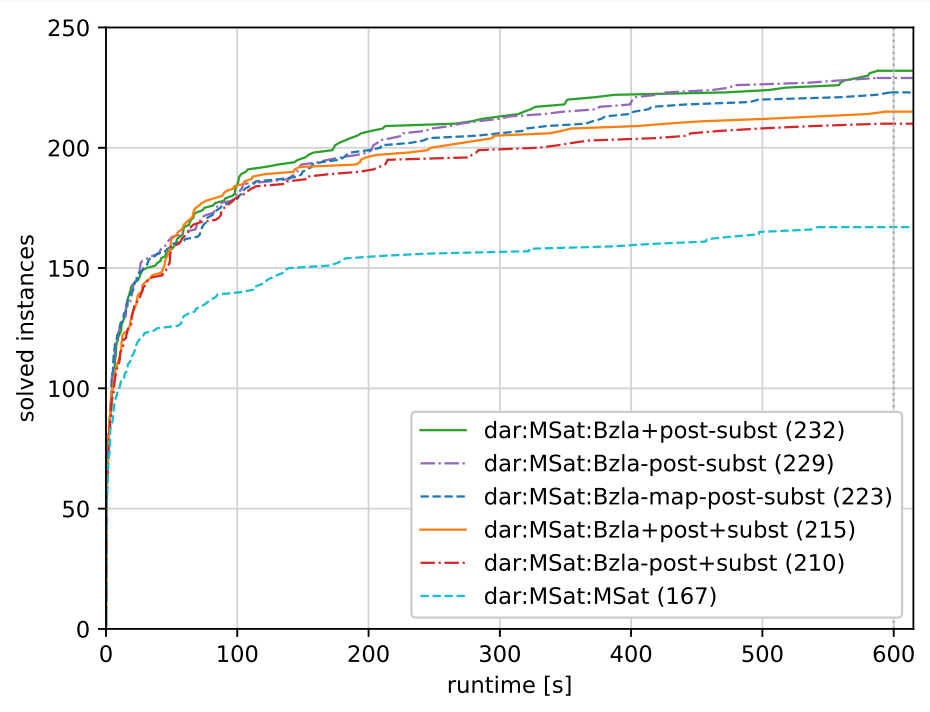


https://bitwuzla.github.io

Appendix

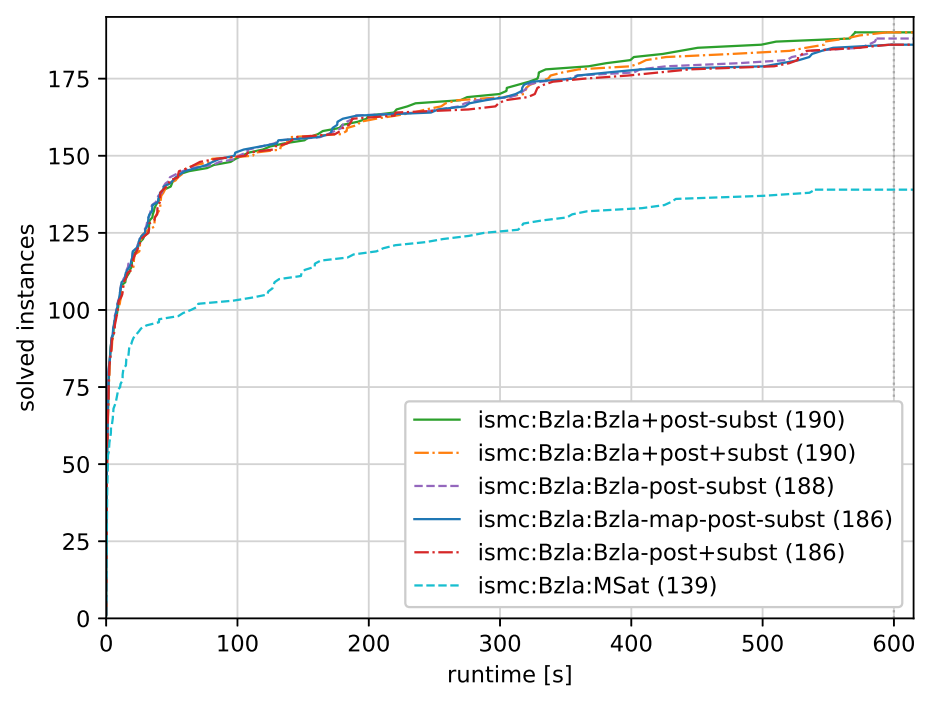
Appendix: Pono Spotlight

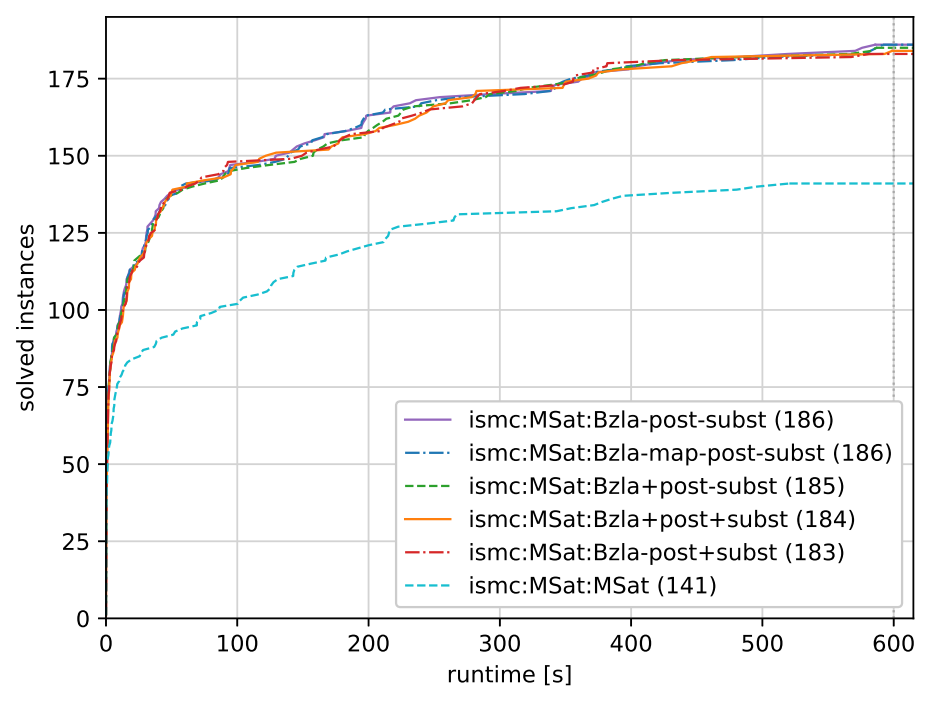




Pono with **ic3ia** engine, on **HWMCC 2019-2024 benchmarks**, 600s time and 16GB memory limit

Appendix: Pono Spotlight





Pono with **ic3ia** engine, on **HWMCC 2019-2024 benchmarks**, 600s time and 16GB memory limit

Appendix; Interpolation Example

```
(set-logic QF_BV)
(set-option :produce-interpolants true)
(declare-const x1 (_ BitVec 2))
(declare-const x2 (_ BitVec 2))
(declare-const x3 (_ BitVec 2))
(assert (! (bvslt (_ bv0 4) (bvsub (concat (_ bv0 2) x1) (_ bv1 4))) :named a1))
(assert (! (= x2 x1) :named a2))
(assert (! (= x3 ((_ extract 1 0) (bvneg (concat (_ bv0 2) x2)))) :named a3))
(assert (= x3 (_ bv0 2)))
(check-sat)
```

Appendix; Interpolation Example

```
(set-logic QF_BV)
(set-option :produce-interpolants true)
(declare-const x1 (_ BitVec 2))
(declare-const x2 (_ BitVec 2))
(declare-const x3 (_ BitVec 2))
(assert (! (bvslt (_ bv0 4) (bvsub (concat (_ bv0 2) x1) (_ bv1 4))) :named a1))
(assert (! (= x2 x1) :named a2))
(assert (! (= x3 ((_ extract 1 0) (bvneg (concat (_ bv0 2) x2)))) :named a3))
(assert (= x3 (_ bv0 2)))
(check-sat)
```

```
; unsat
```

Appendix; Interpolation Example

```
(set-logic QF_BV)
(set-option :produce-interpolants true)
(declare-const x1 (_ BitVec 2))
(declare-const x2 (_ BitVec 2))
(declare-const x3 (_ BitVec 2))
(assert (! (bvslt (_ bv0 4) (bvsub (concat (_ bv0 2) x1) (_ bv1 4))) :named a1))
(assert (! (= x2 x1) :named a2))
(assert (! (= x3 ((_ extract 1 0) (bvneg (concat (_ bv0 2) x2)))) :named a3))
(assert (= x3 (_ bv0 2)))
(check-sat)

; unsat

(get-interpolants (a1 a2))
```

Appendix; Interpolation Example

```
(set-logic QF_BV)
(set-option :produce-interpolants true)
(declare-const x1 (_ BitVec 2))
(declare-const x2 (_ BitVec 2))
(declare-const x3 (_ BitVec 2))
(assert (! (bvslt (_ bv0 4) (bvsub (concat (_ bv0 2) x1) (_ bv1 4))) :named a1))
(assert (! (= x2 x1) :named a2))
(assert (! (= x3 ((_ extract 1 0) (bvneg (concat (_ bv0 2) x2)))) :named a3))
(assert (= x3 (_ bv0 2)))
(check-sat)

; unsat

(get-interpolants (a1 a2))

; (
; (not (= x2 #b00))
; )
```

Appendix; Interpolation Example

```
(set-logic QF_BV)
(set-option :produce-interpolants true)
(declare-const x1 (_ BitVec 2))
(declare-const x2 (_ BitVec 2))
(declare-const x3 (_ BitVec 2))
(assert (! (bvslt (_ bv0 4) (bvsub (concat (_ bv0 2) x1) (_ bv1 4))) :named a1))
(assert (! (= x2 x1) :named a2))
(assert (! (= x3 ((_ extract 1 0) (bvneg (concat (_ bv0 2) x2)))) :named a3))
(assert (= x3 (_ bv0 2)))
(check-sat)

; unsat

(get-interpolants (a1 a2))

; (
; (not (= x2 #b00))
; )

(get-interpolants (a1) (a2) (a3))
```

Appendix; Interpolation Example

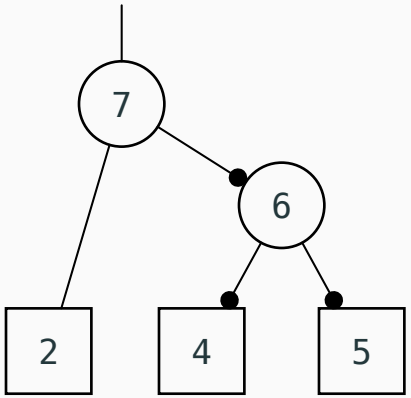
```
(set-logic QF_BV)
(set-option :produce-interpolants true)
(declare-const x1 (_ BitVec 2))
(declare-const x2 (_ BitVec 2))
(declare-const x3 (_ BitVec 2))
(assert (! (bvslt (_ bv0 4) (bvsub (concat (_ bv0 2) x1) (_ bv1 4))) :named a1))
(assert (! (= x2 x1) :named a2))
(assert (! (= x3 ((_ extract 1 0) (bvneg (concat (_ bv0 2) x2)))) :named a3))
(assert (= x3 (_ bv0 2)))
(check-sat)

; unsat

(get-interpolants (a1 a2))

; (
; (not (= x2 #b00))
; )

(get-interpolants (a1) (a2) (a3))
```

expands to `(a1) (a1, a2) (a1, a2, a3)`

Appendix; Interpolation Example

```
(set-logic QF_BV)
(set-option :produce-interpolants true)
(declare-const x1 (_ BitVec 2))
(declare-const x2 (_ BitVec 2))
(declare-const x3 (_ BitVec 2))
(assert (! (bvslt (_ bv0 4) (bvsub (concat (_ bv0 2) x1) (_ bv1 4))) :named a1))
(assert (! (= x2 x1) :named a2))
(assert (! (= x3 ((_ extract 1 0) (bvneg (concat (_ bv0 2) x2)))) :named a3))
(assert (= x3 (_ bv0 2)))
(check-sat)

; unsat

(get-interpolants (a1 a2))

; (
; (not (= x2 #b00))
; )

(get-interpolants (a1) (a2) (a3))

; (
; (= #b0 ((_ extract 3 3) (bvadd (concat #b00 x1) #b1111)))
; (not (= x2 #b00))
; (not (= x3 #b00))
; )
```

Gate Mapping Example



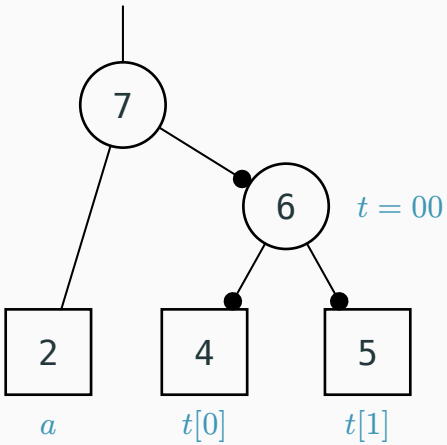
AIG

$\mathcal{S}(A) := \{x, a_{[1]}, t_{[2]}\}$
$\mathcal{S}(B) := \{\quad a_{[1]}, t_{[2]}\}$

$\mathcal{S}_G := \{a_{[1]}, t_{[2]}\}$

Gate Mapping Example

$$\texttt{ite}(t_{[2]} = 00,\ \texttt{ite}(x, 00, 01),\ \texttt{ite}(a_{[1]} = 1, 10, 00))[1]$$

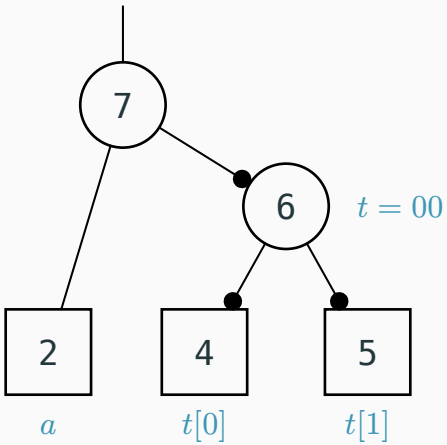


AIG

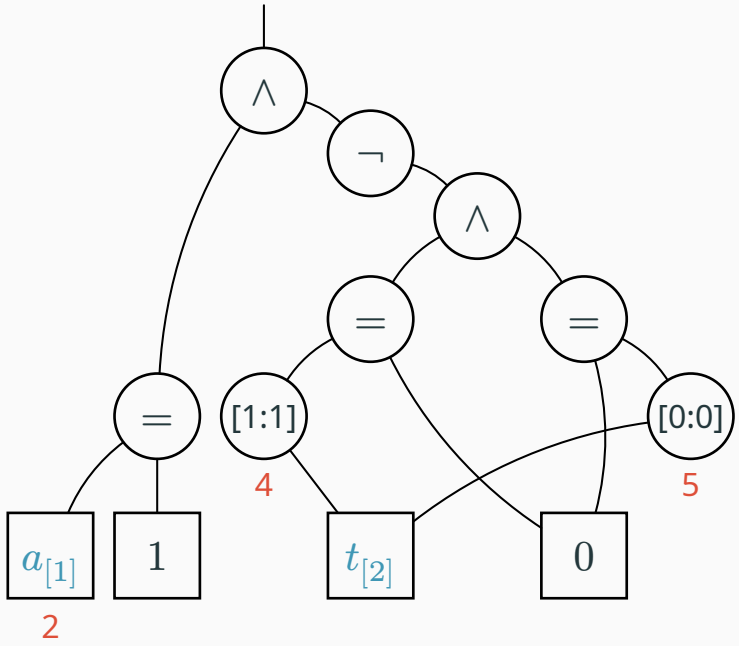
$$\mathcal{S}(A) := \{x, a_{[1]}, t_{[2]}\}$$
$$\mathcal{S}(B) := \{\quad a_{[1]}, t_{[2]}\}$$
$$\mathcal{S}_G := \{a_{[1]}, t_{[2]}\}$$

Gate Mapping Example

$\texttt{ite}(t_{[2]} = 00,$
$\quad \texttt{ite}(x, 00, 01),$
$\quad \texttt{ite}(a_{[1]} = 1, 10, 00))[1]$



AIG



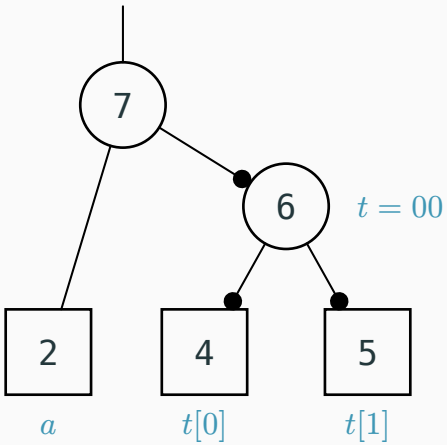
Reconstructed AIG

$\mathcal{S}(A) := \{x, a_{[1]}, t_{[2]}\}$
$\mathcal{S}(B) := \{\quad a_{[1]}, t_{[2]}\}$

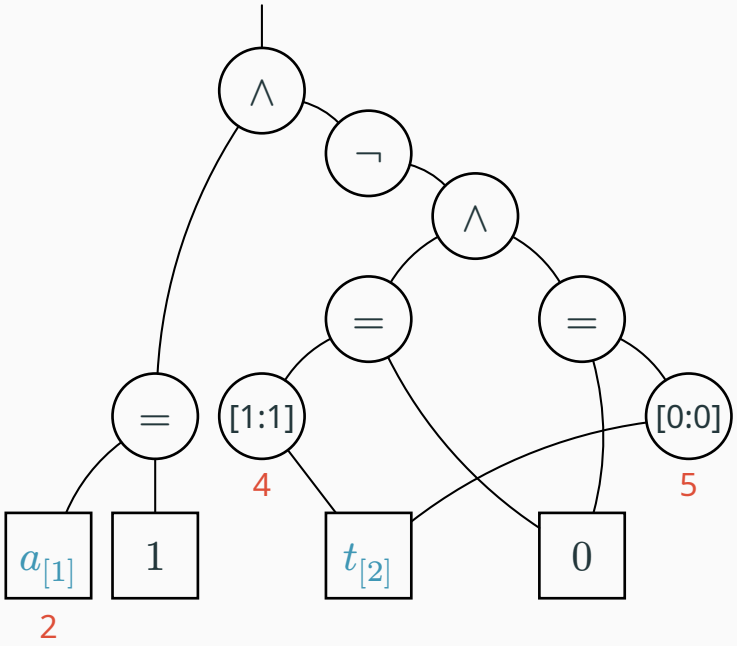
$\mathcal{S}_G := \{a_{[1]}, t_{[2]}\}$

Gate Mapping Example

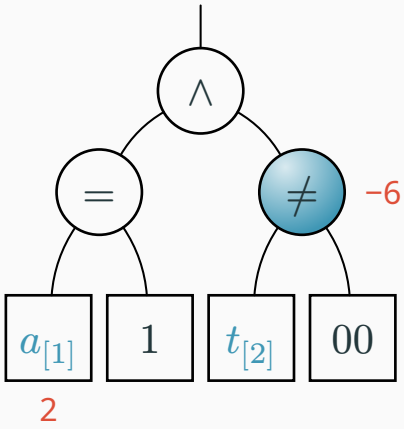
$\texttt{ite}(t_{[2]} = 00,$
$\texttt{ite}(x, 00, 01),$
$\texttt{ite}(a_{[1]} = 1, 10, 00))[1]$



AIG



Reconstructed AIG



Gate Mapping

$\mathcal{S}(A) := \{x, a_{[1]}, t_{[2]}\}$
$\mathcal{S}(B) := \{\quad a_{[1]}, t_{[2]}\}$

$\mathcal{S}_G := \{a_{[1]}, t_{[2]}\}$