Challenges in State-of-the-Art Bit-Precise Reasoning

Aina Niemetz

SLMath Workshop, April 11, 2025

Satisfiability Modulo Theories (SMT)

» **Deciding the satisfiability of FOL formula with respect to background theories**

- fixed interpretation of **theory symbols**
- Boolean + domain-specific reasoning
- more expressive than SAT

Satisfiability Modulo Theories (SMT)

» **Deciding the satisfiability of FOL formula with respect to background theories**

- fixed interpretation of **theory symbols**
- Boolean + domain-specific reasoning
- more expressive than SAT

- **Focus: Theory of fixed-size bit-vectors**

Satisfiability Modulo Theories (SMT)

» **Deciding the satisfiability of FOL formula with respect to background theories**

- fixed interpretation of **theory symbols**
- Boolean + domain-specific reasoning
- more expressive than SAT

- **Focus: Theory of fixed-size bit-vectors**

Example. $(x \ll 001) \geq_s 000 \ \wedge \ x <_u 100 \ \wedge \ (x \cdot 010) \bmod 011 \ = \ x + 001$

satisfiable with $x := 001$

Fixed-Size Bit-Vectors

- **Theory of Fixed-Size Bit-Vectors** provides **bit-precise semantics**
- **Finite** domain
- Semantics of arithmetic operations **different** from Real/Int (modulo 2^n)

Fixed-Size Bit-Vectors

- **Theory of Fixed-Size Bit-Vectors** provides **bit-precise semantics**
- **Finite** domain
- Semantics of arithmetic operations **different** from Real/Int (modulo 2^n)

$$(a > b) \longleftrightarrow (a - b > 0)$$

✓ for Real and Int

? for machine integers

Fixed-Size Bit-Vectors

- **Theory of Fixed-Size Bit-Vectors** provides **bit-precise semantics**
- **Finite** domain
- Semantics of arithmetic operations **different** from Real/Int (modulo 2^n)

$$(a > b) \longleftrightarrow (a - b > 0) \quad \checkmark \text{ for Real and Int}$$

? for machine integers

Example. 8-bit integers (`int8_t`), counterexample: $a = 127$, $b = -128$

✓ $a > b$

✗ $a - b > 0$

» overflow: $-(-128) = -128$, thus $a - b = -1$

Fixed-Size Bit-Vectors

- **Theory of Fixed-Size Bit-Vectors** provides **bit-precise semantics**
- **Finite** domain
- Semantics of arithmetic operations **different** from Real/Int (modulo 2^n)

$$(a > b) \longleftrightarrow (a - b > 0) \quad \checkmark \text{ for Real and Int}$$

? for machine integers

Example. 8-bit integers (`int8_t`), counterexample: $a = 127$, $b = -128$

✓ $a > b$

✗ $a - b > 0$

» overflow: $-(-128) = -128$, thus $a - b = -1$

Theory of Fixed-Size Bit-Vectors

- **bit-vector** as a sequence of bits of a fixed length

  - **constants, variables:** 010, $2_{[3]}$, $x_{[3]}$
  - **bit-vector** operators:
    - **predicates:** $<_u$, $>_s$, ...
    - **bit-wise:** $\sim$, $\&$, $|$, $\oplus$, ...
    - **shift:** $\ll$, $\gg$, $\gg_a$, rotate
    - **word:** $\circ$, $[:]$, $\langle\rangle_u$, $\langle\rangle_s$, repeat
    - **arithmetic:** $-$, $+$, $\cdot$, $\div$, mod, ...
    - **overflow predicates** for arithmetic operators

- **arithmetic** operators modulo 2^n (**overflow semantics!**)

- **natural representation** for machine integers, hardware registers
- widely used in **hardware and software verification**

Theory of Fixed-Size Bit-Vectors: Semantics

Most vs. Least Significant Bit:

LSB: $x_3 x_2 x_1 x_0$

MSB: $x_3 x_2 x_1 x_0$

Value Ranges for Size n:

- unsigned: $[0, 2^n - 1]$ $[0, 15]$
- signed: $[-2^{n-1}, 2^{n-1} - 1]$ $[-8, 7]$

Theory of Fixed-Size Bit-Vectors: Semantics

Most vs. Least Significant Bit:

LSB: $x_3 x_2 x_1 x_0$

MSB: $x_3 x_2 x_1 x_0$

Value Ranges for Size n:

- unsigned: $[0, 2^n - 1]$ $[0, 15]$
- signed: $[-2^{n-1}, 2^{n-1} - 1]$ $[-8, 7]$

Unsigned vs. Signed Interpretation:

$1100_u = 8 + 4 = 12$

$1100_s = -8 + 4 = -4$

$1101_s = -8 + 4 + 1 = -3$

$0011_s = 2 + 1 = 3$

Theory of Fixed-Size Bit-Vectors: Semantics

Most vs. Least Significant Bit:

LSB: $x_3 x_2 x_1 x_0$
MSB: $x_3 x_2 x_1 x_0$

Value Ranges for Size n:

- unsigned: $[0, 2^n - 1]$ $[0, 15]$
- signed: $[-2^{n-1}, 2^{n-1} - 1]$ $[-8, 7]$

Unsigned vs. Signed Interpretation:

$1100_u = 8 + 4 = 12$
$1100_s = -8 + 4 = -4$

$1101_s = -8 + 4 + 1 = -3$
$0011_s = 2 + 1 = 3$

Arithmetic

$8_{[4]} + 11_{[4]} = 3_{[4]}$
$8_{[4]} \cdot 2_{[4]} = 0_{[4]}$
$-8_{[4]} \div_s -1_{[4]} = -8_{[4]}$

Division by Zero

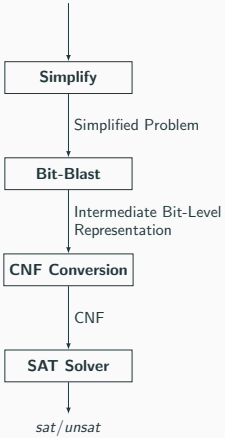
$a_{[n]} \div 0_{[n]} = \sim 0_{[n]} = 1111...1111$
$a_{[n]} \bmod 0_{[n]} = a_{[n]}$

» division is total function

Bit-Blasting

- **The State of The Art** for **quantifier-free** BV formulas
  - » rewriting + simplifications + **reduction** to SAT
  - » BV terms » intermediate representation » CNF
- **efficient** in practice (**state-of-the-art SAT solvers**)

$$(x \ll 001) \geq_s 000 \ \wedge \ x <_u 100$$
$$\wedge \ (x \cdot 010) \bmod 011 \ = \ x + 001$$



Bit-Blasting

- **The State of The Art** for **quantifier-free** BV formulas

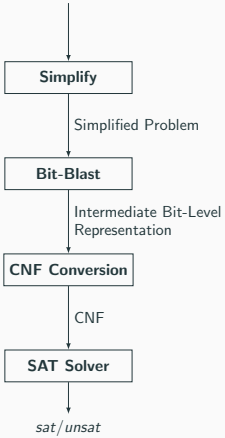
» rewriting + simplifications + **reduction** to SAT

» BV terms » intermediate representation » CNF

- **Main** technique in **state-of-the-art SMT solvers**
  - *Bitwuzla*, *Boolector*, *cvc5*, *MathSAT5*, *STP*, *Yices2*, *Z3*, ...

- **efficient** in practice (**state-of-the-art SAT solvers**)

$$(x \ll 001) \geq_s 000 \ \wedge \ x <_u 100$$
$$\wedge \ (x \cdot 010) \bmod 011 \ = \ x + 001$$



Bit-Blasting: Simplify

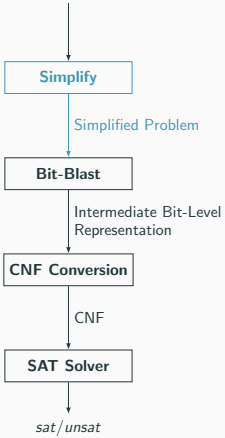
1. Simplify Formula

- Term rewriting (**local** simplifications)
  - $a_{[n]} + 0_{[n]} \quad \rightsquigarrow a$
  - $a_{[n]} \cdot 0_{[n]} \quad \rightsquigarrow 0$
  - ...

  » SMT solvers implement **hundreds** of rewriting rules

- Preprocessing (**global** simplifications)
  - $a = t \wedge F[a] \rightsquigarrow F[t]$
  - ...

$(x \ll 001) \geq_s 000 \ \wedge \ x <_u 100$
$\wedge \ (x \cdot 010) \bmod 011 \ = \ x + 001$

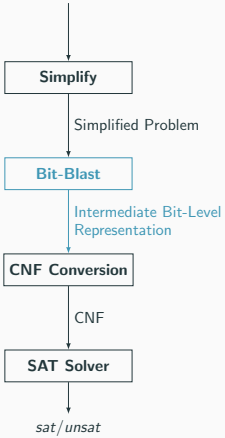


Bit-Blasting: Intermediate Representation

2. Translation into Intermediate Bit-Level Representation

- preserve **hierarchical** bit-level structure
  - » simplify bit-level structure prior to flattening into CNF
- bit-level representation can get **very large**
  - » efficient and compact representation necessary
    - And-Inverter Graphs (AIG) » Bitwuzla, Boolector, MathSAT5, STP
    - Xor/Or/Not Graphs » Yices2
    - Arbitrary Boolean » cvc5, Z3

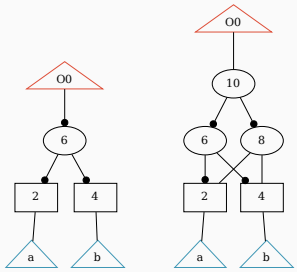
$(x \ll 001) \geq_s 000 \ \wedge \ x <_u 100$
$\wedge \ (x \cdot 010) \bmod 011 \ = \ x + 001$



Bit-Blasting: Intermediate Representation

2. Translation into Intermediate Bit-Level Representation

- **And-Inverter Graphs (AIG)**
  - directed acyclic graphs over $\wedge$, $\neg$
    - nodes represent AND gates
    - negation as edge attribute
  - structural hashing: detect and share identical subgraphs
  - compact and scalable representation
  - efficient simplifications [Mishchenko'06, Brummayer'06]

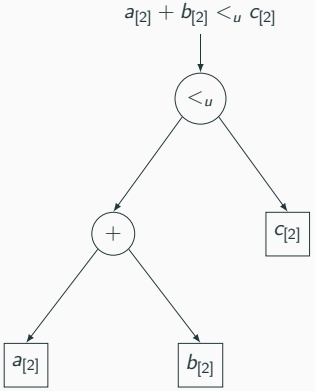


$a \vee b$
$\equiv \neg(\neg a \wedge \neg b)$

$a \oplus b$
$\equiv \neg(\neg a \wedge \neg b)$
$\wedge \neg(a \wedge b)$

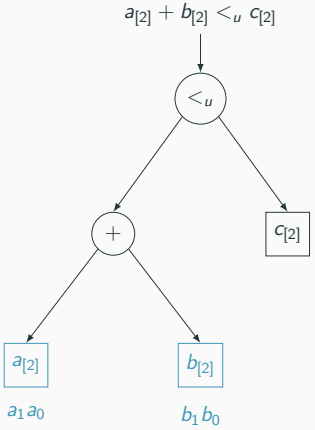
Bit-Blasting: Intermediate Representation

2. Translation into Intermediate Bit-Level Representation



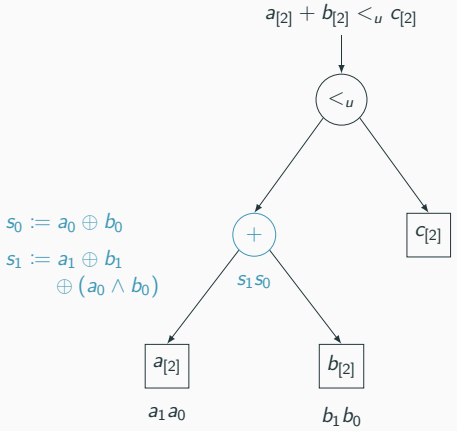
Bit-Blasting: Intermediate Representation

2. Translation into Intermediate Bit-Level Representation



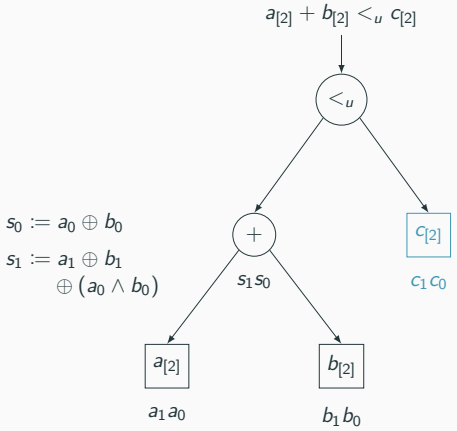
Bit-Blasting: Intermediate Representation

2. Translation into Intermediate Bit-Level Representation



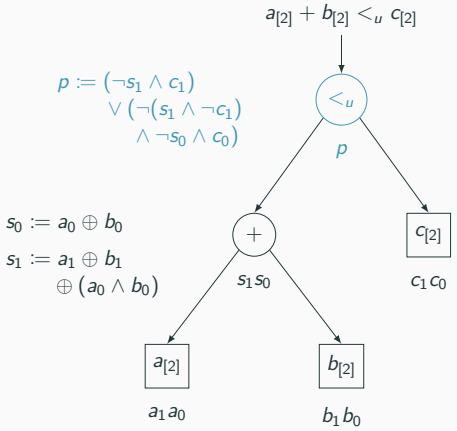
Bit-Blasting: Intermediate Representation

2. Translation into Intermediate Bit-Level Representation



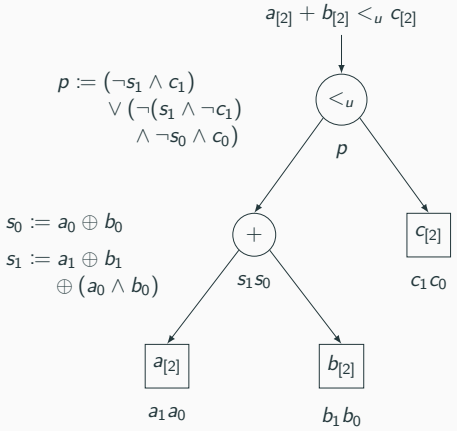
Bit-Blasting: Intermediate Representation

2. Translation into Intermediate Bit-Level Representation



Bit-Blasting: Intermediate Representation

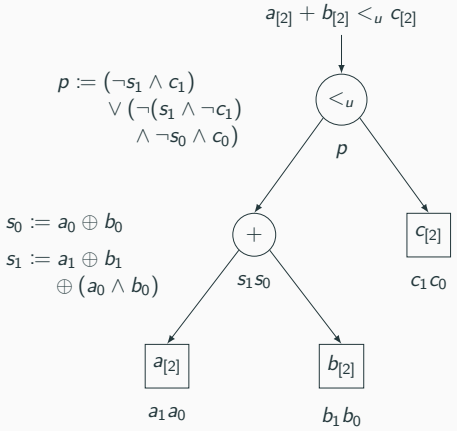
2. Translation into Intermediate Bit-Level Representation



$$
\begin{aligned}
& p \\
\wedge\, p \;&\leftrightarrow ((\neg s_1 \vee c_1) \\
&\quad \vee\ (\neg(s_1 \wedge \neg c_1) \\
&\quad\quad \wedge \neg s_0 \wedge c_0)) \\
\wedge\, s_0 \;&\leftrightarrow (a_0 \oplus b_0) \\
\wedge\, s_1 \;&\leftrightarrow (a_1 \oplus b_1 \\
&\quad \oplus (a_0 \wedge b_0))
\end{aligned}
$$

Bit-Blasting: Intermediate Representation

2. Translation into Intermediate Bit-Level Representation



$$
\begin{aligned}
& p \\
\wedge\, p \;&\leftrightarrow ((\neg s_1 \vee c_1) \\
&\quad \vee\ (\neg(s_1 \wedge \neg c_1) \\
&\quad\ \ \wedge \neg s_0 \wedge c_0)) \\
\wedge\, s_0 \;&\leftrightarrow (a_0 \oplus b_0) \\
\wedge\, s_1 \;&\leftrightarrow (a_1 \oplus b_1 \\
&\quad \oplus (a_0 \wedge b_0))
\end{aligned}
$$

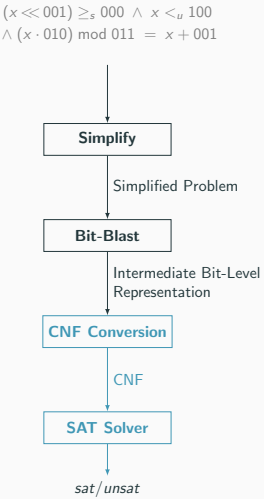
» translate to CNF via **Tseitin** transformation

Bit-Blasting: CNF Conversion

3. CNF Conversion via Tseitin Transformation

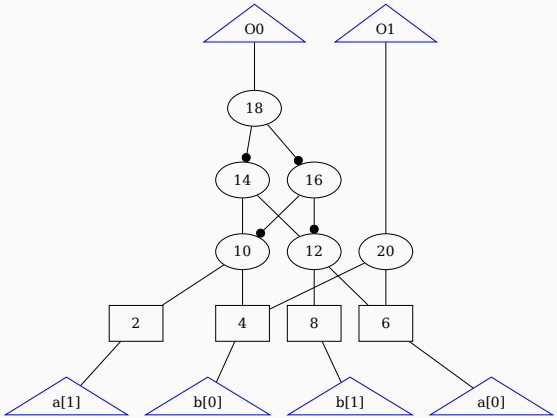
- introduce fresh variable for each AND-gate:
  - » $x \leftrightarrow a \wedge b$
- transform into CNF:
  - » $x \leftrightarrow a \wedge b \quad \rightsquigarrow \quad (\neg x \vee a) \wedge (\neg x \vee b) \wedge (x \vee \neg a \vee \neg b)$

4. Send CNF to SAT Solver



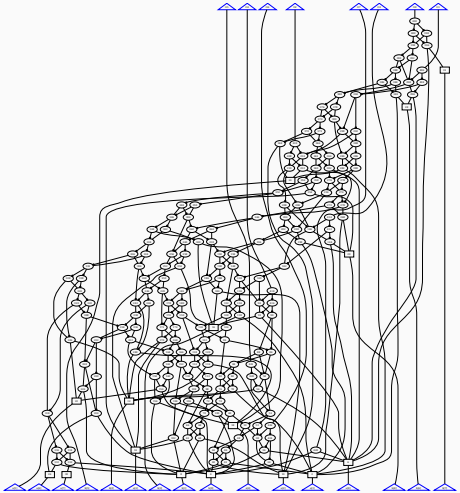
Scalability of Bit-Blasting

2-bit Multiplier (AIG circuit)



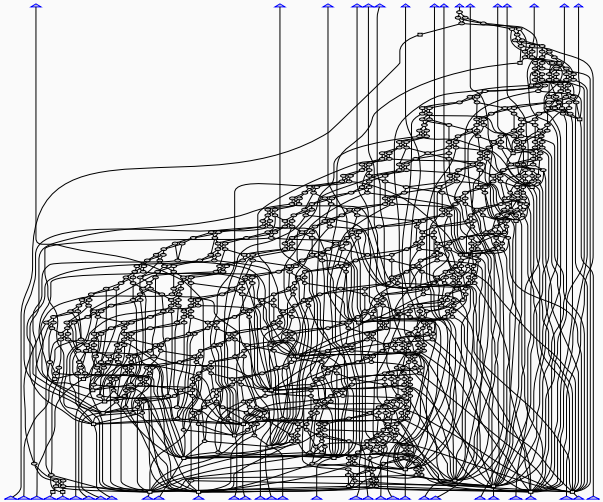
Scalability of Bit-Blasting

8-bit Multiplier (AIG circuit)



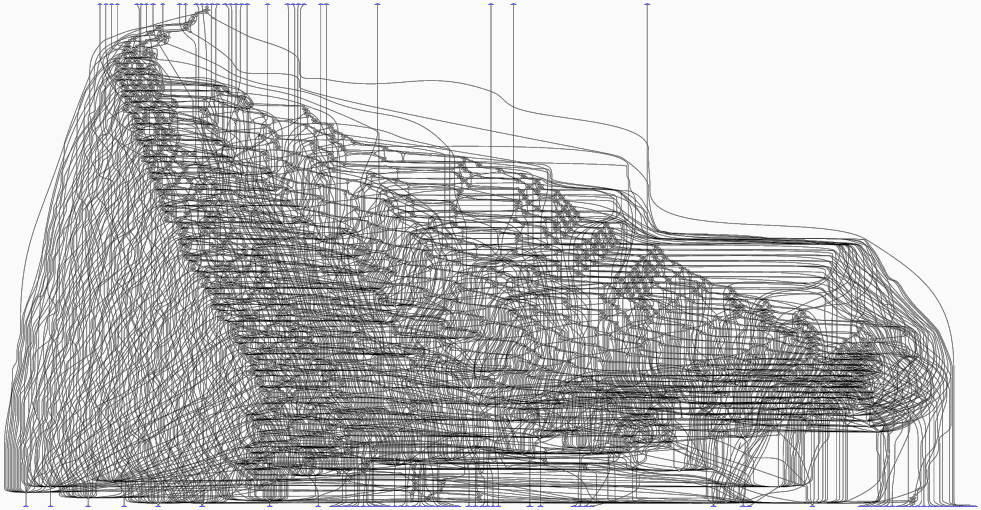
Scalability of Bit-Blasting

16-bit Multiplier (AIG circuit)



Scalability of Bit-Blasting

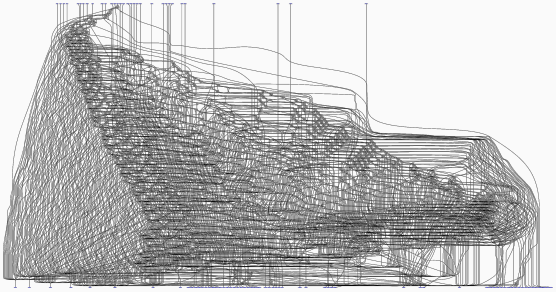
32-bit Multiplier (AIG circuit)



Scalability of Bit-Blasting

- **does not generally scale well** with **increasing** bit-width
- especially with **arithmetic** operators
  - » large and complex Boolean circuits
- potential **bottleneck** for SAT solver
  - » in practice already as low as 32 bits

32-bit Multiplier (AIG circuit)



- especially severe for applications with **large bit-vectors**
  - **smart contract verification:** 256 bits, heavy use of $\{\cdot, \div, \text{mod}\}$
  - **floating-point arithmetic** when word-blasting: 106 bit `bvmul` for Float64 `fp.mul`
  - **cryptography**

How to tackle these scalability issues?

- **Alternative approaches that do not (mainly) rely on bit-blasting**

How to tackle these scalability issues?

- **Alternative approaches that do not (mainly) rely on bit-blasting**
  - **Int-Blasting**: translation to (non-)linear integers
    - Partial int-blasting to linear arithmetic [Bozzano et al. 2006, Rümmer 2008]
    - Full int-blasting to non-linear arithmetic [Zohar et al. 2022]

How to tackle these scalability issues?

- **Alternative approaches that do not (mainly) rely on bit-blasting**
  - **Int-Blasting**: translation to (non-)linear integers
    - Partial int-blasting to linear arithmetic [Bozzano et al. 2006, Rümmer 2008]
    - Full int-blasting to non-linear arithmetic [Zohar et al. 2022]
  - **Layered** approach [Bruttomesso et al. 2007, Hadarean et al. 2014]
    - Layer of cheap (but incomplete) procedures, lazy bit-blasting as fallback

How to tackle these scalability issues?

- **Alternative approaches that do not (mainly) rely on bit-blasting**
  - **Int-Blasting**: translation to (non-)linear integers
    - Partial int-blasting to linear arithmetic [Bozzano et al. 2006, Rümmer 2008]
    - Full int-blasting to non-linear arithmetic [Zohar et al. 2022]
  - **Layered** approach [Bruttomesso et al. 2007, Hadarean et al. 2014]
    - Layer of cheap (but incomplete) procedures, lazy bit-blasting as fallback
  - **MCSAT** for bit-vectors [Zeljic et al. 2016, Graham-Lengrand et al. 2020]
    - Word-level explanations for supported fragments, bit-level as fallback

How to tackle these scalability issues?

- **Alternative approaches that do not (mainly) rely on bit-blasting**
  - **Int-Blasting**: translation to (non-)linear integers
    - Partial int-blasting to linear arithmetic [Bozzano et al. 2006, Rümmer 2008]
    - Full int-blasting to non-linear arithmetic [Zohar et al. 2022]
  - **Layered** approach [Bruttomesso et al. 2007, Hadarean et al. 2014]
    - Layer of cheap (but incomplete) procedures, lazy bit-blasting as fallback
  - **MCSAT** for bit-vectors [Zeljic et al. 2016, Graham-Lengrand et al. 2020]
    - Word-level explanations for supported fragments, bit-level as fallback
  - **Local Search** [Fröhlich et al. 2015, Niemetz et al. 2016, Niemetz et al. 2020]
    - Fast, non-deterministic procedures for satisfiable instances

How to tackle these scalability issues?

- **Alternative approaches that do not (mainly) rely on bit-blasting**
  - **Int-Blasting**: translation to (non-)linear integers
    - Partial int-blasting to linear arithmetic [Bozzano et al. 2006, Rümmer 2008]
    - Full int-blasting to non-linear arithmetic [Zohar et al. 2022]
  - **Layered** approach [Bruttomesso et al. 2007, Hadarean et al. 2014]
    - Layer of cheap (but incomplete) procedures, lazy bit-blasting as fallback
  - **MCSAT** for bit-vectors [Zeljic et al. 2016, Graham-Lengrand et al. 2020]
    - Word-level explanations for supported fragments, bit-level as fallback
  - **Local Search** [Fröhlich et al. 2015, Niemetz et al. 2016, Niemetz et al. 2020]
    - Fast, non-deterministic procedures for satisfiable instances
  - **PolySAT** [Rath et al. 2024]
    - CDCL(T) solver in Z3 for non-linear bit-vector polynomials

Every Technique Has Scalability Issues

1,500 benchmarks* instantiated with bit-widths 16,...,8192, $\sim$ 85 sat, $\sim$ 1415 unsat

Solved Benchmarks

	Bit-Blast	**Lazy+Layered**	**MCSAT**	**Int-Blast**	**PolySAT**	**VBS**
bw	Bitwuzla	CVC4	Yices2	cvc5	Z3	
16	1,495					
32	1,459					
64	1,440					
128	1,433					
256	1,388					
512	1,277					
1,024	1,065					
2,048	844					
4,096	816					
8,192	744					
	99% $\rightarrow$ 49%					

Limits: 1,200 seconds, 8GB memory

* *syrew* benchmarks from [Niemetz et al. 2024]. 500 term and formula equivalence checks enumerated with cvc5's SyGuS solver using SyGuS grammar $\{0, 1, x, s, t, \approx, \not\approx, <_u, \leq_u, \sim, \&, \ll, \gg, \diamond\}$ for $\diamond \in \{\cdot, \div, \mathrm{mod}\}$.

Every Technique Has Scalability Issues

1,500 benchmarks* instantiated with bit-widths 16,...,8192, $\sim$ 85 sat, $\sim$ 1415 unsat

Solved Benchmarks

bw	Bit-Blast Bitwuzla	Lazy+Layered CVC4	MCSAT Yices2	Int-Blast cvc5	PolySAT Z3	VBS
16	1,495	1,458	1,394	1,116	696	
32	1,459	1,390	1,194	1,102	672	
64	1,440	1,368	1,112	1,077	668	
128	1,433	1,308	1,076	1,017	648	
256	1,388	1,232	987	916	637	
512	1,277	1,162	916	788	620	
1,024	1,065	774	794	613	608	
2,048	844	401	668	528	576	
4,096	816	300	572	428	562	
8,192	744	202	492	389	552	
	99% $\rightarrow$ 49%	97% $\rightarrow$ 13%	93% $\rightarrow$ 33%	74% $\rightarrow$ 26%	46% $\rightarrow$ 37%	

Limits: 1,200 seconds, 8GB memory

* *syrew* benchmarks from [Niemetz et al. 2024]. 500 term and formula equivalence checks enumerated with cvc5's SyGuS solver using SyGuS grammar $\{0, 1, x, s, t, \approx, \not\approx, <_u, \leq_u, \sim, \&, \ll, \gg, \diamond\}$ for $\diamond \in \{\cdot, \div, \mathrm{mod}\}$.

Every Technique Has Scalability Issues

1,500 benchmarks* instantiated with bit-widths 16,...,8192, $\sim$ 85 sat, $\sim$ 1415 unsat

Solved Benchmarks

bw	Bit-Blast Bitwuzla	Lazy+Layered CVC4	MCSAT Yices2	Int-Blast cvc5	PolySAT Z3	VBS
16	1,495	1,458	1,394	1,116	696	**1,500**
32	1,459	1,390	1,194	1,102	672	**1,490**
64	1,440	1,368	1,112	1,077	668	**1,487**
128	1,433	1,308	1,076	1,017	648	**1,488**
256	1,388	1,232	987	916	637	**1,480**
512	1,277	1,162	916	788	620	**1,421**
1,024	1,065	774	794	613	608	**1,323**
2,048	844	401	668	528	576	**1,133**
4,096	816	300	572	428	562	**1,074**
8,192	744	202	492	389	552	**993**
	99% $\rightarrow$ 49%	97% $\rightarrow$ 13%	93% $\rightarrow$ 33%	74% $\rightarrow$ 26%	46% $\rightarrow$ 37%	**100%** $\rightarrow$ **66%**

Limits: 1,200 seconds, 8GB memory

* *syrew* benchmarks from [Niemetz et al. 2024]. 500 term and formula equivalence checks enumerated with cvc5's SyGuS solver using SyGuS grammar $\{0, 1, x, s, t, \approx, \not\approx, <_u, \leq_u, \sim, \&, \ll, \gg, \diamond\}$ for $\diamond \in \{\cdot, \div, \mathrm{mod}\}$.

How to tackle these scalability issues?

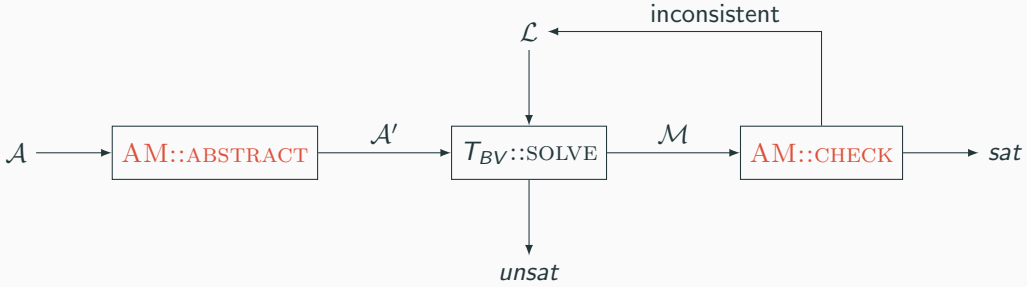
- **Improve scalability of bit-blasting itself**
  - combination of **under- and over-approximation** [Bryant et al. 2007]
    - under-approximation via restricting value ranges of inputs
    - over-approximation via ITE elimination, assertion abstraction and abstracting $x \cdot y$ with partially interpreted function $\lambda x.\lambda y.ite(x \approx 0 \vee y \approx 0, 0, ite(x = 1, y, ite(y \approx 1, x, f(x, y))))$
  - similar **under-approximation** techniques in [Brummayer 2009, Jonás et al. 2020]
  - **CEGAR-style framework** abstracting $\{\cdot, \div, \text{mod}\}$ [Niemetz et al. 2024]

Scalable Bit-Blasting with Abstractions*

- a **CEGAR-style** abstraction-refinement framework for **theory BV**
  - » Counter Example Guided Abstraction Refinement (CEGAR)
  - » based on **bit-blasting**
    - implemented in our SMT solver **Bitwuzla**
    - significantly **improves** performance on **large** bit-vectors
- **abstraction refinement scheme** for bit-vector operators $\{\cdot, \div, \mathrm{mod}\}$
  - » **most expensive** operators for bit-blasting
    - 70 lemmas total
- abduction-based framework for **lemma synthesis**
- lemma **scoring** scheme

* [Niemetz et al. 2024] joint work with M. Preiner and Y. Zohar

Abstraction-Refinement Loop



- abstract each relevant $\{\cdot, \div, \text{mod}\}$ term with **fresh constant**
- **over-approximation**

- **check consistency** with true semantics of operators
  - » consistent: ✓
  - » inconsistent: refine abstraction

4-Tiered Refinement Scheme

- **CEGAR Loop Refinement Step for Abstracted Term**

  For each abstracted term $t := x \diamond s$

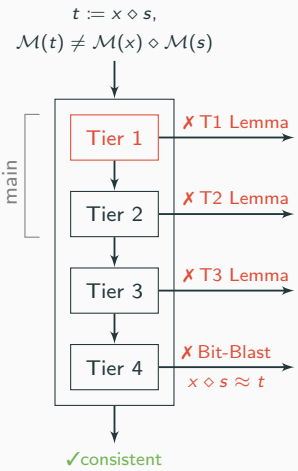
  1. check if $\mathcal{M}(t) \neq \mathcal{M}(x) \diamond \mathcal{M}(s)$

     » e.g., $t := x \cdot s$ $\mathcal{M} = \{x_{[32]} = 3, s_{[32]} = 6, t_{[32]} = 1\}$

  2. check if a lemma in tiers 1–3 evaluates to false under $\mathcal{M}$

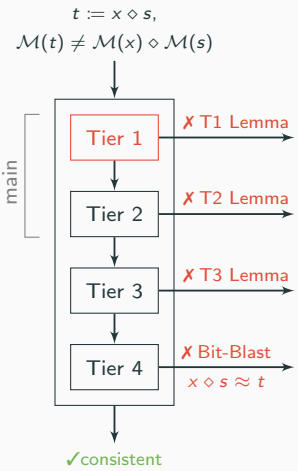
     » tiers processed in order 1-4

  3. use first violated lemma as refinement



4-Tiered Refinement Scheme

- **Tier 1** **Hand-Crafted Lemmas**
  - properties described by **invertibility condition** IC
    - Given $x \diamond s \approx t$, is there a value for x given s and t?
    - $\exists x.\, x \diamond s \approx t \Leftrightarrow IC[s, t]$

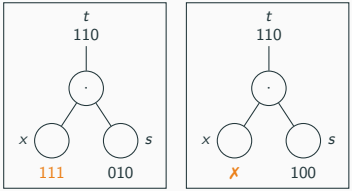


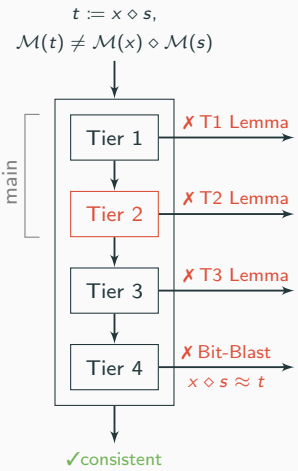
4-Tiered Refinement Scheme

▶ Tier 1 Hand-Crafted Lemmas

▷ properties described by **invertibility condition** IC

- Given $x \diamond s \approx t$, is there a value for x given s and t?
- $\exists x.\, x \diamond s \approx t \Leftrightarrow IC[s,t]$
- e.g., $x \cdot s$ has at least as many trailing zeros as x or s
  - » $((-s \mid s)\ \&\ t) \approx t$
  - » $((-x \mid x)\ \&\ t) \approx t$





4-Tiered Refinement Scheme

▶ Tier 1 Hand-Crafted Lemmas

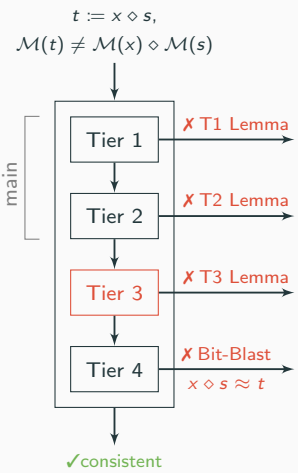
▷ properties described by **invertibility condition** IC

- Given $x \diamond s \approx t$, is there a value for x given s and t?
- $\exists x.\, x \diamond s \approx t \Leftrightarrow IC[s,t]$
- e.g., $x \cdot s$ has at least as many trailing zeros as x or s
  - » $((-s \mid s)\ \&\ t) \approx t$
  - » $((-x \mid x)\ \&\ t) \approx t$

▷ **basic** properties of the abstracted operator

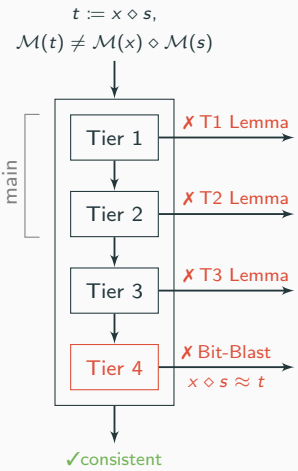
- e.g., abstraction t for $x \cdot s$ is odd iff x and s are odd
  - » $t[0] \approx (x[0]\ \&\ s[0])$

▷ **verified** up to bit-width 512 (17 lemmas)



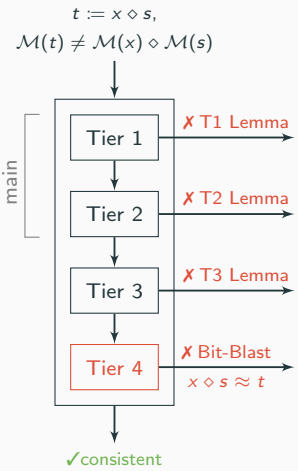
4-Tiered Refinement Scheme

- **Tier 2 Synthesized Lemmas**
  - synthesized via **syntax-restricted abduction** with **cvc5** (offline)
    - **abduction:** A, B, find C s.t. $A \wedge C \Rightarrow B$ valid, $A \wedge C$ sat
    - find **lemma** ℓ such that $\top \wedge \neg\ell \Rightarrow x \diamond s \not\approx t$
  - e.g., for $x \cdot s \approx t$
    - $x \not\approx (1 \oplus (x \ll (s \oplus t)))$
  - lemmas **filtered** based on **lemma score**
  - **with respect to hand-crafted (tier 1) lemmas**
  - **verified** up to bit-width 512 (53 lemmas)



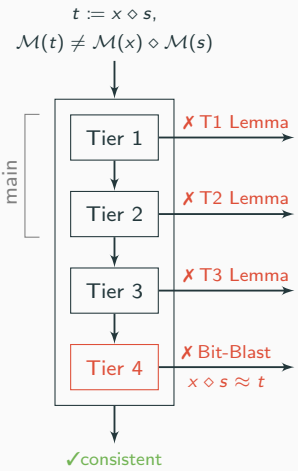
4-Tiered Refinement Scheme

- **Tier 3** **Value Instantiation** Lemmas
  - rule out **current inconsistent model value**
  - only added if none in tiers 1–2 are violated
  - e.g., for $x \cdot s$ and $\mathcal{M} = \{x_{[32]} = 3, s_{[32]} = 6, t_{[32]} = 1\}$, we add lemma $(x = 3 \wedge s = 6) \Rightarrow t = 18$
  - **limited** fallback strategy
    - » limited to $w/8$ instantiations per term
    - » e.g., 4 instantiations for 32 bit



4-Tiered Refinement Scheme

- **Tier 4** **Bit-blasting** lemmas
  - **last resort**
  - add lemma to **enforce bit-blasting** of the abstracted term
  - e.g., lemma $t \approx x \cdot s$ for $x \cdot s$



Tier 1* and 2 Refinement Lemmas ($t \coloneqq x \diamond s$)

bvmul

1*	$s \approx 2^i \Rightarrow t \approx x \ll i$	$11_{>1}$	$t \not\approx (1 \mid \sim(x \oplus s))$
2*	$s \approx -2^i \Rightarrow t \approx -x \ll i$	$12_{>1}$	$t \not\approx (\sim 1 \mid (x \oplus s))$
3*	$((-s \mid s) \,\&\, t) \approx t$	13	$x \not\approx ((x \ll (s+t)) - 1)$
4*	$t[0] \approx (x[0] \,\&\, s[0])$	14	$x \not\approx (1 - (x \ll (s-t)))$
$5_{>1}$	$s \not\approx \sim(t \mid (1 \,\&\, (x \mid s)))$	15	$s \not\approx (1 + (s \ll (t-x)))$
$6_{>1}$	$(x \,\&\, t) \not\approx (s \mid \sim t)$	16	$s \not\approx (1 - (s \ll (t-x)))$
$7_{>1}$	$t \not\approx ((s \mid 1) \ll (t \ll x))$	17	$s \not\approx (1 + (s \ll (x-t)))$
8	$s \approx (s \ll (x \,\&\, (1 \gg t)))$	$18_{>1}$	$t \not\approx (1 \mid (x + s))$
$9_{\neq 2}$	$t \geq_u (1 \,\&\, ((x \,\&\, s) \gg 1))$	19	$x \not\approx \sim(x \ll (s+t))$
10	$x \not\approx (1 \oplus (x \ll (s \oplus t)))$		

bvurem

1*	$s \approx 2^i \Rightarrow t \approx (0_{[\kappa(x)-i]} \circ x[i-1:0])$	9	$x \geq_u (t \mid (x \,\&\, s))$
2*	$s \not\approx 0 \Rightarrow t \leq_u s$	10	$1 \not\approx (t \,\&\, \sim(x \mid s))$
3*	$x \approx 0 \Rightarrow t \approx 0$	11	$t \not\approx (\sim x \mid -s)$
4*	$s \approx 0 \Rightarrow t \approx x$	12	$(t \,\&\, (x \mid s)) \geq_u (t \,\&\, 1)$
5*	$s \approx x \Rightarrow t \approx 0$	$13_{>2}$	$x \not\approx (-x \mid -\sim t)$
6*	$x <_u s \Rightarrow t \approx x$	14	$(x + -s) \geq_u t$
7*	$\sim -s \geq_u t$	15	$(-s \oplus (x \mid s)) \geq_u t$
8	$x \approx (x \,\&\, (s \mid (t \mid -s)))$		

bvudiv

1*	$s \approx 2^i \Rightarrow t \approx x \gg i$	19	$(x \gg t) \not\approx (s \mid t)$
2*	$(s \approx x \wedge s \not\approx 0) \Rightarrow t \approx 1$	20	$s \not\approx \sim(s \gg (t \gg 1))$
3*	$s \approx 0 \Rightarrow t \approx \sim 0$	$21_{>1}$	$x \not\approx \sim(x \,\&\, (t \ll 1))$
4*	$(x \approx 0 \wedge s \not\approx 0) \Rightarrow t \approx 0$	22	$t \geq_u ((x \ll 1) \gg s)$
5*	$s \not\approx 0 \Rightarrow t \leq_u x$	23	$x \geq_u (s \ll \sim(x \mid t))$
6*	$(s \approx \sim 0 \wedge x \not\approx \sim 0) \Rightarrow t \approx 0$	24	$x \geq_u (t \ll \sim(x \mid s))$
7	$x \geq_u -(-s \,\&\, -t)$	25	$x \geq_u (t \oplus (t \gg (s \gg 1)))$
8	$-(s \mid 1) \geq_u t$	26	$x \geq_u (s \oplus (s \gg (t \gg 1)))$
9	$t \not\approx -(s \,\&\, \sim x)$	27	$x \geq_u (s \ll \sim(x \oplus t))$
10	$(s \mid t) \not\approx (x \,\&\, \sim 1)$	28	$x \geq_u (t \ll \sim(x \oplus s))$
11	$(s \mid 1) \not\approx (x \,\&\, \sim t)$	29	$x \not\approx (t + (s \mid (x + s)))$
12	$(x \,\&\, -t) \geq_u (s \,\&\, t)$	$30_{>2}$	$x \not\approx (t + (1 + (1 \ll x)))$
13	$s \geq_u (x \gg t)$	31	$s \geq_u ((x + t) \gg t)$
14	$x \geq_u ((s \gg (s \ll t)) \ll 1)$	$32_{>1}$	$x \not\approx (t + (t + (x \mid s)))$
15	$x \geq_u ((t \ll 1) \gg (t \ll s))$	33	$(s \oplus (x \mid t)) \geq_u (t \oplus 1)$
16	$t \geq_u ((x \gg s) \ll 1)$	34	$t \geq_u (x \gg (s-1))$
17	$x \geq_u ((x \mid t) \,\&\, (s \ll 1))$	35	$(s-1) \geq_u (x \gg t)$
18	$x \geq_u ((x \mid s) \,\&\, (t \ll 1))$	$36_{\neq 2}$	$x \not\approx (1 - (x \ll (x-t)))$

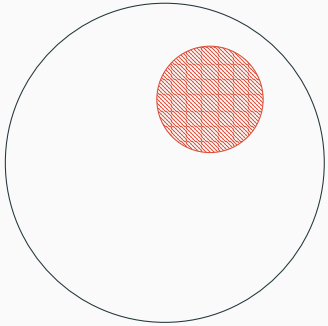
Lemma Score

- **Metric for quality of a lemma**

▷ Given abstracted term $x \diamond s$ and lemma $\ell[x, s, t]$ s.t. $x \diamond s \approx t \Rightarrow \ell$, then
$\textsc{Score}(\ell, w) :=$ num. triplets (v^x, v^s, v^t) where $\ell[v^x, v^s, v^t] = \top$.

Example. $x \cdot s$ with $w = 4$

- Worst score: $2^4 \times 2^4 \times 2^4 = 4096$ $(\ell = \top)$
- Best score: $2^4 \times 2^4 = 256$ $(\ell = x \cdot s \approx t)$



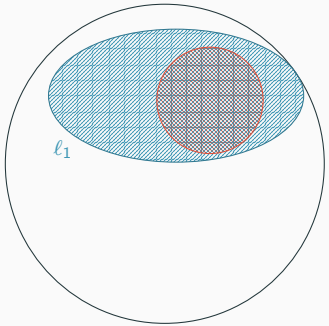
Lemma Score

- **Metric for quality of a lemma**

▷ Given abstracted term $x \diamond s$ and lemma $\ell[x, s, t]$ s.t. $x \diamond s \approx t \Rightarrow \ell$, then
$\textsc{Score}(\ell, w) :=$ num. triplets (v^x, v^s, v^t) where $\ell[v^x, v^s, v^t] = \top$.

Example. $x \cdot s$ with $w = 4$

- Worst score: $2^4 \times 2^4 \times 2^4 = 4096$ $(\ell = \top)$
- Best score: $2^4 \times 2^4 = 256$ $(\ell = x \cdot s \approx t)$



ℓ_1: 2416

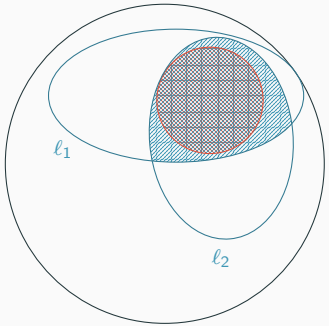
Lemma Score

- **Metric for quality of a lemma**

▷ Given abstracted term $x \diamond s$ and lemma $\ell[x, s, t]$ s.t. $x \diamond s \approx t \Rightarrow \ell$, then
$\textsc{Score}(\ell, w) :=$ num. triplets (v^x, v^s, v^t) where $\ell[v^x, v^s, v^t] = \top$.

Example. $x \cdot s$ with $w = 4$

- Worst score: $2^4 \times 2^4 \times 2^4 = 4096$ $(\ell = \top)$
- Best score: $2^4 \times 2^4 = 256$ $(\ell = x \cdot s \approx t)$



ℓ_1: 2416
ℓ_2: 2791

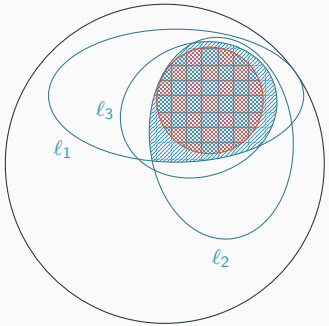
Lemma Score

- **Metric for quality of a lemma**

▷ Given abstracted term $x \diamond s$ and lemma $\ell[x, s, t]$ s.t. $x \diamond s \approx t \Rightarrow \ell$, then
$\textsc{Score}(\ell, w) :=$ num. triplets (v^x, v^s, v^t) where $\ell[v^x, v^s, v^t] = \top$.

Example. $x \cdot s$ with $w = 4$

- Worst score: $2^4 \times 2^4 \times 2^4 = 4096$ ($\ell = \top$)
- Best score: $2^4 \times 2^4 = 256$ ($\ell = x \cdot s \approx t$)



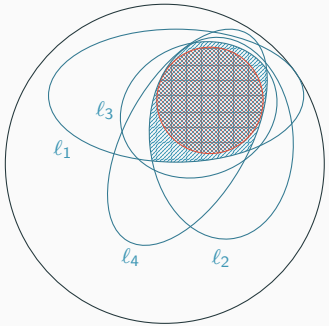
Lemma Score

- **Metric for quality of a lemma**

▷ Given abstracted term $x \diamond s$ and lemma $\ell[x, s, t]$ s.t. $x \diamond s \approx t \Rightarrow \ell$, then
$\textsc{Score}(\ell, w) :=$ num. triplets (v^x, v^s, v^t) where $\ell[v^x, v^s, v^t] = \top$.

Example. $x \cdot s$ with $w = 4$

- Worst score: $2^4 \times 2^4 \times 2^4 = 4096$ ($\ell = \top$)
- Best score: $2^4 \times 2^4 = 256$ ($\ell = x \cdot s \approx t$)
- **Score for hand-crafted $\{\ell_1, \ldots, \ell_4\}$: 704**
  - » **rule out 88% of incorrect triplets**



ℓ_1: 2416
ℓ_2: 2791
ℓ_3: 1961
ℓ_4: 2048

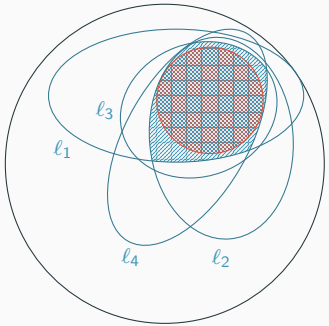
Lemma Score

- **Metric for quality of a lemma**

▷ Given abstracted term $x \diamond s$ and lemma $\ell[x, s, t]$ s.t. $x \diamond s \approx t \Rightarrow \ell$, then
$\textsc{Score}(\ell, w) :=$ num. triplets (v^x, v^s, v^t) where $\ell[v^x, v^s, v^t] = \top$.

Example. $x \cdot s$ with $w = 4$

- Worst score: $2^4 \times 2^4 \times 2^4 = 4096$ ($\ell = \top$)
- Best score: $2^4 \times 2^4 = 256$ ($\ell = x \cdot s \approx t$)
- **Score for hand-crafted $\{\ell_1, \ldots, \ell_4\}$: 704**
  - » **rule out 88% of incorrect triplets**
- **Overall scores** (tiers 1+2, w = 4)
  - » $x \cdot s$ 490 ≅ 94% (19 lemmas)
  - » $x \div s$ 396 ≅ 96% (36 lemmas)
  - » $x \bmod s$ 400 ≅ 96% (15 lemmas)



ℓ_1: 2416
ℓ_2: 2791
ℓ_3: 1961
ℓ_4: 2048

Every Technique Has Scalability Issues

1,500 benchmarks* instantiated with bit-widths 16,. . . ,8192, $\sim$ 85 sat, $\sim$ 1415 unsat

Solved Benchmarks

bw	Bit-Blast Bitwuzla	Lazy+Layered CVC4	MCSAT Yices2	Int-Blast cvc5	PolySAT Z3	VBS	Bit-Blast+Abstr Bitwuzla
16	1,495	1,458	1,394	1,116	696	**1,500**	
32	1,459	1,390	1,194	1,102	672	**1,490**	
64	1,440	1,368	1,112	1,077	668	**1,487**	
128	1,433	1,308	1,076	1,017	648	**1,488**	
256	1,388	1,232	987	916	637	**1,480**	
512	1,277	1,162	916	788	620	**1,421**	
1,024	1,065	774	794	613	608	**1,323**	
2,048	844	401	668	528	576	**1,133**	
4,096	816	300	572	428	562	**1,074**	
8,192	744	202	492	389	552	**993**	
	99% $\rightarrow$ 49%	97% $\rightarrow$ 13%	93% $\rightarrow$ 33%	74% $\rightarrow$ 26%	46% $\rightarrow$ 37%	**100%** $\rightarrow$ **66%**	

Limits: 1,200 seconds, 8GB memory

* *syrew* benchmarks from [Niemetz et al. 2024]. 500 term and formula equivalence checks enumerated with cvc5's SyGuS solver using SyGuS grammar $\{0, 1, x, s, t, \approx, \not\approx, <_u, \leq_u, \sim, \&, \ll, \gg, \diamond\}$ for $\diamond \in \{\cdot, \div, \mathrm{mod}\}$.

Every Technique Has Scalability Issues

1,500 benchmarks* instantiated with bit-widths 16,. . . ,8192, $\sim$ 85 sat, $\sim$ 1415 unsat

Solved Benchmarks

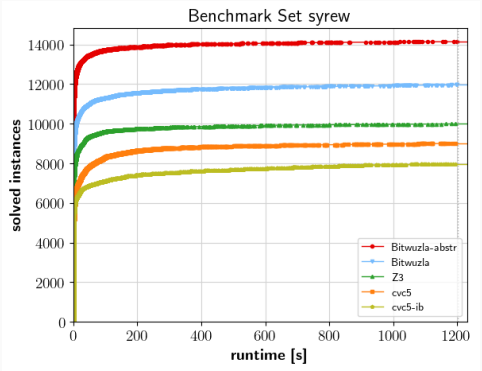
bw	Bit-Blast Bitwuzla	Lazy+Layered CVC4	MCSAT Yices2	Int-Blast cvc5	PolySAT Z3	VBS	Bit-Blast+Abstr Bitwuzla
16	1,495	1,458	1,394	1,116	696	**1,500**	**1,495**
32	1,459	1,390	1,194	1,102	672	**1,490**	**1,459**
64	1,440	1,368	1,112	1,077	668	**1,487**	**1,454**
128	1,433	1,308	1,076	1,017	648	**1,488**	**1,458**
256	1,388	1,232	987	916	637	**1,480**	**1,456**
512	1,277	1,162	916	788	620	**1,421**	**1,449**
1,024	1,065	774	794	613	608	**1,323**	**1,434**
2,048	844	401	668	528	576	**1,133**	**1,365**
4,096	816	300	572	428	562	**1,074**	**1,300**
8,192	744	202	492	389	552	**993**	**1,274**
	99% $\rightarrow$ 49%	97% $\rightarrow$ 13%	93% $\rightarrow$ 33%	74% $\rightarrow$ 26%	46% $\rightarrow$ 37%	**100% $\rightarrow$ 66%**	99% $\rightarrow$ 85%

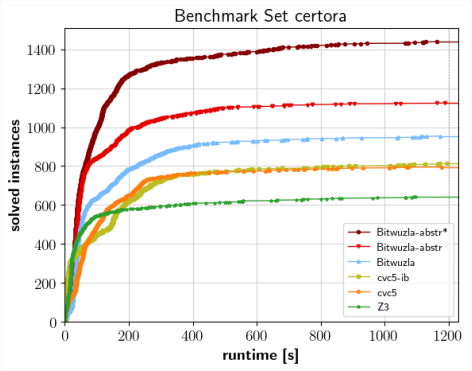
Limits: 1,200 seconds, 8GB memory

100% $\rightarrow$ 88%

* *syrew* benchmarks from [Niemetz et al. 2024]. 500 term and formula equivalence checks enumerated with cvc5's SyGuS solver using SyGuS grammar $\{0, 1, x, s, t, \approx, \not\approx, <_u, \leq_u, \sim, \&, \ll, \gg, \diamond\}$ for $\diamond \in \{\cdot, \div, \mathrm{mod}\}$.

Evaluation





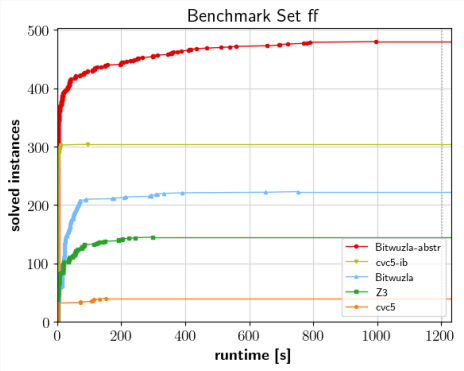
» **500 term and formula equivalence checks**

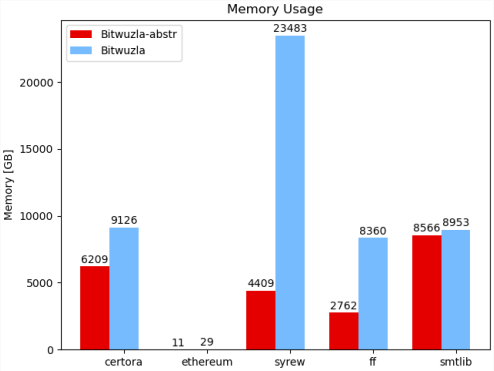
- enumerated by SyGuS (cvc5) for $w = 4$
- instantiated for $w = \{16, 32, \ldots, 8192\}$
- 1500 benchmarks, $\sim$ 85 sat, $\sim$ 1415 unsat

» **Smart contract verification**

- 256 bit bit-vectors
- heavy use of $\{\cdot, \div, \text{mod}\}$

Evaluation





» **Translation validation of ZK proofs**

- 510 bit bit-vectors

Evaluation: Abstraction Statistics over all Benchmark Sets

- **80% solved without bit-blasting of abstracted terms**
  - » Tier 1-3 refinement lemmas were sufficent to solve the problem

Family	Solved	Abstr. Only	T1	T2	T3	T4
certora$_1$	255	**86.7%**	86.3%	78.0%	60.4%	13.3%
certora$_2$	395	**97.2%**	85.3%	46.3%	50.4%	2.8%
ethereum	24	**100%**	41.7%	29.2%	16.7%	0.0%
syrew	7,355	**91.9%**	98.1%	23.5%	12.3%	8.1%
ff	480	**95.2%**	84.6%	38.1%	34.8%	4.8%
smtlib	15,524	**73.9%**	64.3%	27.5%	36.5%	26.1%
Total	24,033	**80.4%**	75.6%	27.3%	29.5%	19.6%

- 20% required bit-blasting of abstracted terms
  - » Only 37% of multiplication, 13% of division, 2% of remainder terms bit-blasted

- ▷ On average 37 refinement iterations (median 4)

What is still challenging?

Example: Commutativity of Multiplication

$a \cdot b \neq b \cdot a$

```
(set-logic QF_BV)
(declare-const a (_ BitVec 13))
(declare-const b (_ BitVec 13))
(assert (distinct (bvmul a b) (bvmul b a)))
(check-sat)
```

bw	CNF v/c	Solve [s]	DRAT Size
5	142/282	< 0.00	6.2K
6	216/467	0.03	56K
7	305/697	0.16	593K
8	406/972	0.6	2.2M
9	528/1292	3.4	9.6M
10	662/1657	18	40M
11	811/2067	95	171M
12	975/2522	436	459M
13	1152/3022	2188	1.8G

What is still challenging?

Example: Commutativity of Multiplication

$a \cdot b \neq b \cdot a$

```
(set-logic QF_BV)
(declare-const a (_ BitVec 13))
(declare-const b (_ BitVec 13))
(assert (distinct (bvmul a b) (bvmul b a)))
(check-sat)
```

bw	CNF v/c	Solve [s]	DRAT Size
5	142/282	< 0.00	6.2K
6	216/467	0.03	56K
7	305/697	0.16	593K
8	406/972	0.6	2.2M
9	528/1292	3.4	9.6M
10	662/1657	18	40M
11	811/2067	95	171M
12	975/2522	436	459M
13	1152/3022	2188	1.8G

- **Arithmetic circuit verification very hard for SAT solvers**
  - » results indicate **exponential size resolution proofs** [Biere'16]
- This example: **trivial for SMT** solvers with **commutativity rewrite rule**

What is still challenging?

Example: Multiplication + Bitwise

$$a \cdot b \neq (a \,\&\, \sim b) \cdot (\sim a \,\&\, b) + (a \,\&\, b) \cdot (a \mid b)$$

```
(set-logic QF_BV)
(declare-const a (_ BitVec 16))
(declare-const b (_ BitVec 16))
(assert
 (distinct
  (bvmul a b)
  (bvadd
   (bvmul (bvand a (bvnot b)) (bvand (bvnot a) b))
   (bvmul (bvand a b) (bvor a b)))
  )
)
(check-sat)
```

- Worse SAT solver behavior as before
- Very hard for SMT solvers
  - Insufficient word-level reasoning
- Enumeration faster than SMT solvers
  - For 16-bit example (left)
  - SMT solvers time out after 60min
    - Bit-Blast (Bitwuzla)
    - Int-Blast (cvc5)
    - Lazy Bit-Blast+Layered (CVC4)
    - MCSAT (Yices2)
    - PolySAT (Z3)
  - C++ program enumerates in ∼13s

Takeaways

- **Large bit-vectors** and/with **arithmetic** **challenging** for state-of-the-art approaches
  - » arithmetic usually **in combination** with bitwise/shift/word operators
  - » scalability **not only** an issue for bit-blasting
- Bit-Blasting still **best-performing** approach
  - » alternative approaches **complementary**
- **CEGAR-style** abstraction-refinement for **theory BV** based on **bit-blasting**
  - » **significantly improves scalability of bit-blasting**
  - » **for majority of solved instances full arithmetic circuit not required**

References

- A. Niemetz, M. Preiner, Y. Zohar. *Scalable Bit-Blasting with Abstractions.* In Proc. of CAV'24, pages 178–200, Springer, 2024.
- J. Rath, C. Eisenhofer, D. Kaufmann, N. Bjørner, L. Kovács *PolySAT: Word-level Bit-vector Reasoning in Z3.* `https://arxiv.org/abs/2406.04696`, 2024.
- Y. Zohar, A. Irfan, M. Mann, A. Niemetz, A. Nötzli, M. Preiner, A. Reynolds, C. Barrett, and C. Tinelli. *Bit-Precise Reasoning via Int-Blasting.* In Proc. of VMCAI'22, pages 496–518, Springer, 2022.
- A. Niemetz, M. Preiner. *Ternary Propagation-Based Local Search for more Bit-Precise Reasoning.* In Proc. of FMCAD'20, pages 214–224, IEEE, 2020.
- M. Jonás and J. Strejcek. *Speeding up Quantified Bit-Vector SMT Solvers by Bit-Width Reductions and Extensions.* In Proc. of SAT'20, pages 378–393, Springer, 2020.

References


S. Graham-Lengrand, D. Jovanovic, B. Dutertre. *Solving Bitvectors with MCSAT: Explanations from Bits and Pieces.* In Proc. of IJCAR'20, pages 103–121, Springer, 2020.


M. Brain, F. Schanda, Y. Sun. *Building Better Bit-Blasting for Floating-Point Problems.* In Proc. of TACAS'19, 2019.


A. Zeljic, C. Wintersteiger, P. Rümmer. *Deciding Bit-Vector Formulas with mcSAT.* In Proc. of SAT'16, pages 249–266, Springer, 2016.


A. Niemetz, M. Preiner, A. Biere. *Precise and Complete Propagation Based Local Search for Satisfiability Modulo Theories.* In Proc. of CAV'16, pages 199–217, Springer, 2016.


A. Biere. *Weaknesses of CDCL Solvers.* Fields Institute Workshop on Theoretical Foundations for SAT Solving, `http://www.fields.utoronto.ca/talks/weaknesses-cdcl-solvers`, August 2016.

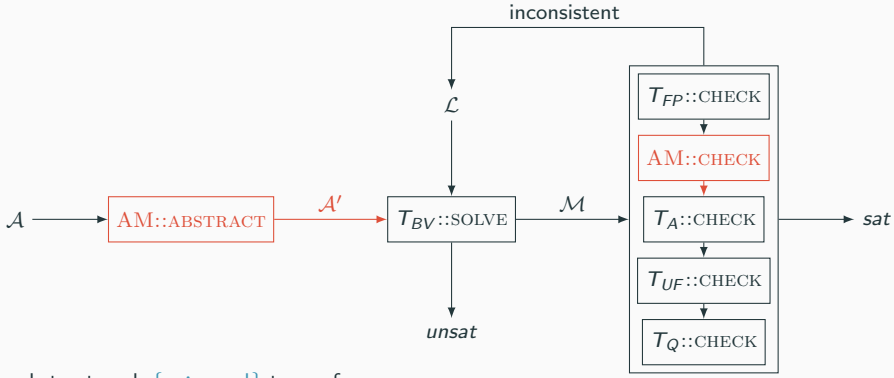
References

- A. Fröhlich, A. Biere, C. Wintersteiger, Y. Hamadi. In Proc. of AAAI'15, pages 1136–1143, AAAI Press, 2015.
- L. Hadarean, K. Bansal, D. Jovanovic, C. W. Barrett and C. Tinelli *A Tale of Two Solvers: Eager and Lazy Approaches to Bit-Vectors.* In Proc. of CAV'14, pages 680–695, Springer, 2014.
- R. Brummayer. *Efficient SMT Solving for Bit-Vectors and the Extensional Theory of Arrays.* PhD Thesis, Johannes Kepler University, 2009.
- R. E. Bryant, D. Kroening, J. Ouaknine, S. A. Seshia, O. Strichman and B. A. Brady.*Deciding Bit-Vector Arithmetic with Abstraction* In Proc. of TACAS'07, pages 358–372, Springer, 2007.
- P. Rümmer. *A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic.* In Proc. of LPAR'08, pages 274–289, Springer, 2008.

References

- R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, Z. Hanna, A. Nadel, A. Palti, R. Sebastiani. *A Lazy and Layered SMT($\mathcal{BV}$) Solver for Hard Industrial Verification Problems.* In Proc. of CAV'07, pages 547–560, Springer, 2007.
- M. Bozzano, R.Bruttomesso, A.Cimatti, A. Franzén, Z. Hanna, Z. Khasidashvili, A. Palti, R. Sebastiani. *Encoding RTL Constructs for MathSAT: a Preliminary Report.* ENTCS, vol. 144, pages 3–14, Springer, 2006.
- A. Mishchenko, S. Chatterjee, and R. Brayton. *DAG-Aware AIG Rewriting - A Fresh Look at Combinational Logic Synthesis.* In Proc. DAC'06, 2006.
- R. Brummayer and A. Biere *Local Two-Level And-Inverter Graph Minimization without Blowup.* In Proc. of MEMICS'06, 2006.

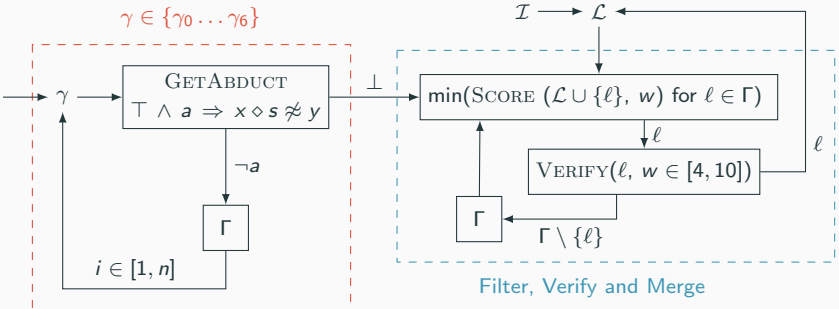
Bitwuzla: Lemmas on Demand Loop with Abstraction Module



- abstract each $\{\cdot, \div, \text{mod}\}$ term of size ≥ 32 with **fresh constant**
- optional: **assertion abstraction** [?]
  - » interleaved with term abstraction
  - » effective if unsat core very small

- order **not arbitrary**
  - » T_{FP} word-blasted to T_{BV}
  - » T_A, T_{UF} and T_Q require consistent T_{BV} abstraction

Lemma Synthesis



- **verified lemmas for bit-widths [1, 512]**
  - Bitwuzla, cvc5, Yices, Z3
  - 8 hours time limit, 8 GB memory limit
  - 16,896 benchmark, 6348 CPU hours

Evaluation: FP logics

- **SMT-LIB**
  - » *smtlib*
  - all quantifier-free and quantified logics supported by Bitwuzla (24 in total)
  - combinations of BV, FP, UF, Arrays
- **FP Logics**
  - » Bitwuzla uses word-blasting for FP logics
    - Reduction of FP to bit-vectors with SymFPU [Brain et al. 2019b]
  - » Operations on unpacked formats require full precision arithmetic
    - Float32 `fp.mul`: 48-bit `bvmul`
    - Float64 `fp.mul`: 106-bit `bvmul`

Bitwuzla vs. Bitwuzla-abstr

Logic	Solved (Diff)	Time [s] (Common)
FP	+5	−16%
BVFP	0	−45%
QF_ABVFP	+1	−33%
QF_ABVFPLRA	0	−23%
QF_BVFP	+1	−45%
QF_BVFPLRA	+9	−46%
QF_FP	+23	−13%
QF_FPLRA	+1	−7%
QF_BV/float	+11	−16%

Evaluation

Benchmarks	Solver	Solved	Timeout	Memout	Time [s]	Mem [GB]
certora$_1$ (850)	Abstr-ta	573	231	46	448k	2,492
	Abstr-a	386	140	324	681k	5,201
	Bitwuzla-abstr	258	155	437	760k	4,807
	cvc5-ib	147	674	0	879k	667
	Bitwuzla	111	86	653	915k	6,182
	cvc5	90	113	610	923k	6,064
	Z3	30	447	373	989k	4,944
certora$_2$ (1,138)	Abstr-ta	866	264	8	370k	1,024
	Bitwuzla-abstr	866	263	9	384k	1,402
	Abstr-a	844	269	25	433k	2,661
	Bitwuzla	843	266	29	439k	2,944
	cvc5	705	223	210	603k	4,027
	cvc5-ib	666	472	0	643k	106
	Z3	612	492	34	679k	1,866
ethereum (3,173)	Bitwuzla-abstr	3,173	0	0	407	11
	Bitwuzla	3,173	0	0	720	29
	Z3	3,169	4	0	6k	107
	cvc5	3,158	0	1	18k	36
	cvc5-ib	3,141	20	0	39k	21

Evaluation

Benchmarks	Solver	Solved	Timeout	Memout	Time [s]	Mem [GB]
syrew (15,000)	Bitwuzla-abstr	14,142	583	276	1,225k	4,409
	Bitwuzla	11,961	744	2,296	3,955k	23,483
	Z3	9,992	833	4,175	6,198k	39,506
	cvc5	9,003	797	5,200	7,498k	48,421
	cvc5-ib	7,974	5,137	1,632	8,836k	19,850
ff (1,224)	cvc5-ff	973	129	122	313k	1,364
	Bitwuzla-abstr	480	729	15	913k	2,762
	cvc5-ib	304	822	98	1,104k	1,074
	Bitwuzla	223	71	930	1,211k	8,360
	Z3	145	56	1,023	1,299k	8,893
	cvc5	40	0	1,184	1,422k	9,523
smtlib (155,269)	Bitwuzla-abstr	148,554	1,944	152	8,770k	8,566
	Bitwuzla	148,492	1,966	193	8,748k	8,953
	Z3	145,121	4,846	565	13,528k	18,278
	cvc5	144,829	3,775	285	13,513k	11,029
	cvc5-ib	127,144	24,479	194	39,647k	15,233

36 unique

Limits: 1200 seconds CPU time limit, 8GB memory limit