Challenges in Bit-Vector Reasoning

Mathias Preiner

SMT Workshop, July 23, 2024

Theory of Fixed-Size Bit-Vectors

Fixed-Size Bit-Vector

- Sequence of bits of a fixed length

Natural Representation of

- ▷ Machine integers in software
- ▷ Hardware registers

. . . and operations on them

Theory of Fixed-Size Bit-Vectors

- Defines semantics of rich set of bit-vector operators
- ▷ Ideal for problems that require bit-precise semantics
- ▷ Used in hardware and software verification: model checking, compiler optimization verification, symbolic execution, . . .

Theory of Fixed-Size Bit-Vectors: Syntax

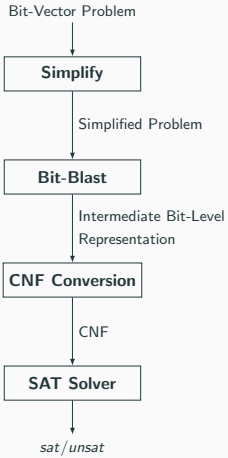
Bit-Vector Operators

- ▷ Predicates: $= \quad <_u \quad \leq_u \quad <_s \quad \leq_s \quad \ldots$
- ▷ Bitwise: $\sim \quad \& \quad | \quad \oplus \quad \ldots$
- ▷ Shifts: $\ll \quad \gg \quad \gg_a$ rotate_left rotate_right
- ▷ Word: $[:] \quad \circ \quad \langle\rangle_u \quad \langle\rangle_s$ repeat
- ▷ Arithmetic: $+ \quad - \quad \cdot \quad \div \quad \text{mod} \quad \div_s \quad \ldots$
- ▶ Arithmetic operations modulo 2^n **(overflow semantics!)**
- ▷ Overflow Predicates: negation, addition, multiplication, ...

State of the Art in Bit-Vector Reasoning

Bit-Blasting

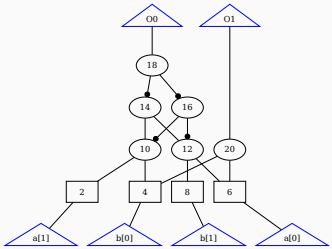
- State of the art for solving quantifier-free bit-vectors
  - Reduces bit-vectors to propositional logic (SAT)
    - Combined with rewriting and preprocessing
  - Employed by the majority of modern SMT solvers: *Bitwuzla*, *Boolector*, *cvc5*, *MathSAT5*, *STP*, *Yices2*, *Z3*, ...
  - Employed eagerly or lazily
  - Efficient in practice
    - Thanks to state-of-the-art SAT solvers
- Increasing bit-width: Significant increase in CNF size



Scalability of Bit-Blasting

- Does not generally scale well with increasing bit-width
- Especially in the presence of arithmetic operators
  - Large and complex Boolean circuits
- Size increase potential bottleneck for SAT solver
- Especially severe for applications that require large bit-vectors
  - **Smart contract verification:** 256 bits, heavy use of arithmetic
  - **Floating-point arithmetic** when word-blasting
  - **Cryptography** (finite fields)

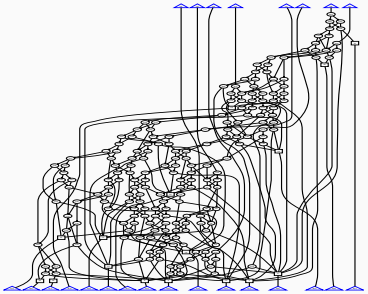
2-bit Multiplier AIG Circuit



Scalability of Bit-Blasting

- Does not generally scale well with increasing bit-width
- ▷ Especially in the presence of arithmetic operators
  - Large and complex Boolean circuits
- Size increase potential bottleneck for SAT solver
- Especially severe for applications that require large bit-vectors
  - **Smart contract verification:** 256 bits, heavy use of arithmetic
  - **Floating-point arithmetic** when word-blasting
  - **Cryptography** (finite fields)

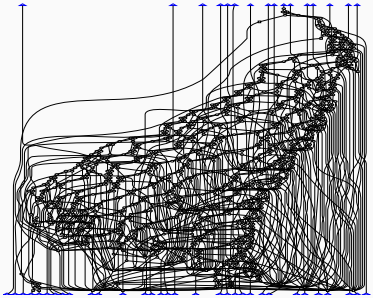
8-bit Multiplier AIG Circuit



Scalability of Bit-Blasting

- Does not generally scale well with increasing bit-width
- ▷ Especially in the presence of arithmetic operators
  - Large and complex Boolean circuits
- Size increase potential bottleneck for SAT solver
- Especially severe for applications that require large bit-vectors
  - **Smart contract verification:** 256 bits, heavy use of arithmetic
  - **Floating-point arithmetic** when word-blasting
  - **Cryptography** (finite fields)

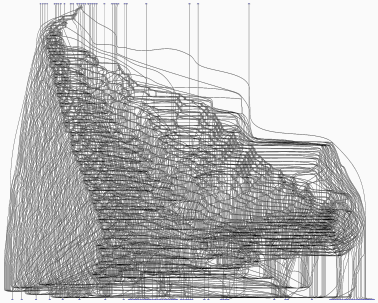
16-bit Multiplier AIG Circuit



Scalability of Bit-Blasting

- Does not generally scale well with increasing bit-width
- ▷ Especially in the presence of arithmetic operators
  - Large and complex Boolean circuits
- Size increase potential bottleneck for SAT solver
- Especially severe for applications that require large bit-vectors
  - **Smart contract verification:** 256 bits, heavy use of arithmetic
  - **Floating-point arithmetic** when word-blasting
  - **Cryptography** (finite fields)

32-bit Multiplier AIG Circuit



Alternative Bit-Vector Approaches

- **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]

Alternative Bit-Vector Approaches

- **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

Alternative Bit-Vector Approaches

- **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

Alternative Bit-Vector Approaches

- **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 procedures for satisfiable instances

Alternative Bit-Vector Approaches

- **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 procedures for satisfiable instances
- **PolySAT** [Rath et al. 2024]
  - CDCL(T) solver in Z3 for non-linear bit-vector polynomials

Alternative Bit-Vector Approaches

- **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 procedures for satisfiable instances
- **PolySAT** [Rath et al. 2024]
  - CDCL(T) solver in Z3 for non-linear bit-vector polynomials
- **Generally not (yet?) competitive with bit-blasting**

SMT-COMP'24 QF_BV Statistics

10,703 benchmarks in 48 benchmark families selected for QF_BV

- 10,560 solved (98.7%)
  - Status: 4,945 sat, 5,615 unsat
- Bit-width range
  - Overall: 1-32768
  - Arithmetic: 1-29980
- 84% with arithmetic
  - » 2.5% pure arithmetic
    (without bitwise/shifts/word)

SMT-COMP'24 QF_BV Statistics

10,703 benchmarks in 48 benchmark families selected for QF_BV

- 10,560 solved (98.7%)
  - Status: 4,945 sat, 5,615 unsat
- Bit-width range
  - Overall: 1-32768
  - Arithmetic: 1-29980
- 84% with arithmetic
  - » 2.5% pure arithmetic (without bitwise/shifts/word)

- 143 unsolved (1.3%) in 20 families
  - Status: 12 sat, 42 unsat, 89 unknown
- Bit-width range
  - Overall: 4-8192
  - Arithmetic: 4-1024
- 140 with arithmetic (97.9%)
  - » 20 pure arithmetic
  - » 120 mixed with bitwise/shifts/word
- 3 without arithmetic (2.1%)

Employed techniques: Bit-Blasting, Int-Blasting* (Bitwuzla, cvc5, SMTInterpol*, STP, Yices2, Z3-alpha)

SMT-COMP'24 QF_BV Statistics

10,703 benchmarks in 48 benchmark families selected for QF_BV

- 10,560 solved (98.7%)
  - Status: 4,945 sat, 5,615 unsat
- Bit-width range
  - Overall: 1-32768
  - Arithmetic: 1-29980
- 84% with arithmetic
  - » 2.5% pure arithmetic (without bitwise/shifts/word)

- 143 unsolved (1.3%) in 20 families
  - Status: 12 sat, 42 unsat, 89 unknown
- Bit-width range
  - Overall: 4-8192
  - Arithmetic: 4-1024
- 140 with arithmetic (97.9%)
  - » 20 pure arithmetic
  - » 120 mixed with bitwise/shifts/word
- 3 without arithmetic (2.1%)

Employed techniques: Bit-Blasting, Int-Blasting* (Bitwuzla, cvc5, SMTInterpol*, STP, Yices2, Z3-alpha)
Local Search (Bitwuzla), Int-Blasting (cvc5), MCSAT (Yices2), PolySAT (Z3) **not able to solve any of 143**

Challenges Discussed in this Talk

- Arithmetic
  - Addition, multiplication, division
  - Arithmetic circuit verification problems
  - Non-linear multiplication
  - More challenging in combination with bitwise/shifts/word operators
  - **Not only challenging for bit-blasting**
- Large bit-vectors
  - **Smart contract verification:** 256 bits, arithmetic heavy
  - **Floating-point arithmetic:** encoded as bit-vectors
  - **Scalability not only an issue for bit-blasting**
    - Int-Blasting: Bitwise operators/shifts
    - Local Search: Too many choices for values
    - MCSAT: Scalability of BDDs with increasing bit-widths, bit-level interpolation

Challenges: Arithmetic

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)
```

Challenges: Arithmetic

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

Challenges: Arithmetic Mixed with Bitwise

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

Challenges: Arithmetic Normalization and Sharing

- Local simplifications may destroy sharing
  - **Example:** $a - b = c \rightsquigarrow b + c = a$
  - Bad if $a - b$ is shared somewhere else, since $b + c$ additionally introduced

Challenges: Arithmetic Normalization and Sharing

- Local simplifications may destroy sharing
  - **Example:** $a - b = c \rightsquigarrow b + c = a$
  - Bad if $a - b$ is shared somewhere else, since $b + c$ additionally introduced
- Speculative simplifications with potentially expensive rewrites
  - **Example:** $a \cdot (b + c) \rightsquigarrow a \cdot b + a \cdot c$
  - Good if $a \cdot b$ and $a \cdot c$ already in the problem, otherwise introduces more multipliers
  - » Need to determine whether problem got easier (e.g. clause count estimation in STP)

Challenges: Arithmetic Normalization and Sharing

- Local simplifications may destroy sharing
  - **Example:** $a - b = c \rightsquigarrow b + c = a$
  - Bad if $a - b$ is shared somewhere else, since $b + c$ additionally introduced
- Speculative simplifications with potentially expensive rewrites
  - **Example:** $a \cdot (b + c) \rightsquigarrow a \cdot b + a \cdot c$
  - Good if $a \cdot b$ and $a \cdot c$ already in the problem, otherwise introduces more multipliers
  - » Need to determine whether problem got easier (e.g. clause count estimation in STP)
- Word-level simplifications may destroy sharing on bit-level
  - **Example:** $c + b + a \rightsquigarrow a + b + c$
  - Bad if $\langle c \rangle_s + \langle b \rangle_u + \langle a \rangle_s$ already in the problem
    - » Subsumes $c + b + a$ on the bit-level
  - » Need to determine whether problem got easier w.r.t. bit-level

Challenges: Large Bit-Vectors, Increasing Bit-Width

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

Solved Benchmarks

Technique	**VBS**	**Bit-Blast**	**Lazy+Layered**	**MCSAT**	**Int-Blast**	**PolySAT**
bw		Bitwuzla	CVC4	Yices2	cvc5	Z3
16	1,500					
32	1,490					
64	1,487					
128	1,488					
256	1,480					
512	1,421					
1,024	1,323					
2,048	1,133					
4,096	1,074					
8,192	993					
	100% $\rightarrow$ 66%					

Limits: 1,200 seconds, 8GB memory

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

Challenges: Large Bit-Vectors, Increasing Bit-Width

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

Solved Benchmarks

Technique	**VBS**	**Bit-Blast**	**Lazy+Layered**	**MCSAT**	**Int-Blast**	**PolySAT**
bw		Bitwuzla	CVC4	Yices2	cvc5	Z3
16	1,500	1,495				
32	1,490	1,459				
64	1,487	1,440				
128	1,488	1,433				
256	1,480	1,388				
512	1,421	1,277				
1,024	1,323	1,065				
2,048	1,133	844				
4,096	1,074	816				
8,192	993	744				
	100% $\rightarrow$ 66%	99% $\rightarrow$ 49%				

Limits: 1,200 seconds, 8GB memory

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

Challenges: Large Bit-Vectors, Increasing Bit-Width

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

Solved Benchmarks

Technique	**VBS**	**Bit-Blast**	**Lazy+Layered**	**MCSAT**	**Int-Blast**	**PolySAT**
bw		Bitwuzla	CVC4	Yices2	cvc5	Z3
16	1,500	1,495	1,458			
32	1,490	1,459	1,390			
64	1,487	1,440	1,368			
128	1,488	1,433	1,308			
256	1,480	1,388	1,232			
512	1,421	1,277	1,162			
1,024	1,323	1,065	774			
2,048	1,133	844	401			
4,096	1,074	816	300			
8,192	993	744	202			
	100% $\rightarrow$ 66%	99% $\rightarrow$ 49%	97% $\rightarrow$ 13%			

Limits: 1,200 seconds, 8GB memory

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

Challenges: Large Bit-Vectors, Increasing Bit-Width

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

Solved Benchmarks

Technique / **bw**	**VBS**	**Bit-Blast** Bitwuzla	**Lazy+Layered** CVC4	**MCSAT** Yices2	**Int-Blast** cvc5	**PolySAT** Z3
16	1,500	1,495	1,458	1,394		
32	1,490	1,459	1,390	1,194		
64	1,487	1,440	1,368	1,112		
128	1,488	1,433	1,308	1,076		
256	1,480	1,388	1,232	987		
512	1,421	1,277	1,162	916		
1,024	1,323	1,065	774	794		
2,048	1,133	844	401	668		
4,096	1,074	816	300	572		
8,192	993	744	202	492		
	100% $\rightarrow$ 66%	99% $\rightarrow$ 49%	97% $\rightarrow$ 13%	93% $\rightarrow$ 33%		

Limits: 1,200 seconds, 8GB memory

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

Challenges: Large Bit-Vectors, Increasing Bit-Width

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

Solved Benchmarks

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

Limits: 1,200 seconds, 8GB memory

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

Challenges: Large Bit-Vectors, Increasing Bit-Width

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

Solved Benchmarks

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

Limits: 1,200 seconds, 8GB memory

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

Improving Bit-Blasting with Abstractions

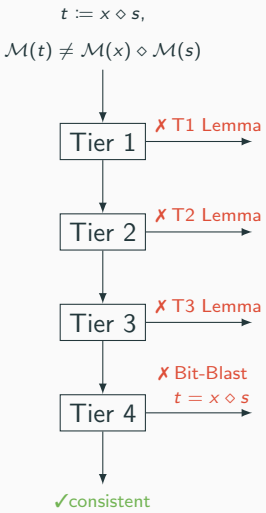
- Joint work with **Aina Niemetz** and **Yoni Zohar**
  - [Niemetz et al. 2024] at CAV'24
- CEGAR-style abstraction-refinement framework for theory BV
  - Implemented in **Bitwuzla** since early 2024
- ▷ Refinement schemes for arithmetic bit-vector operators
  - **Most expensive** operators for bit-blasting $\{\cdot, \div, \mathrm{mod}\}$
- ▷ **4-Tiered** set of refinement lemmas

Bit-Vector Arithmetic Refinement Schemes

CEGAR Loop Refinement Step for Abstracted Term

▷ Checked in order 1-4, if a lemma is violated use as refinement

- **Tier 1** **Hand-Crafted Lemmas** (predefined, 17 lemmas)
  - Describe basic properties of arithmetic operators
- **Tier 2** **Synthesized Lemmas** (predefined, 53 lemmas)
  - Synthesized via syntax-restricted abduction (cvc5, offline)
- **Tier 3** **Value Instantiation Lemmas** (limited fallback)
  - Rules out specific assignment
- **Tier 4** **Bit-Blasting Lemmas** (last resort)
  - Fully bit-blasts abstracted term



Tier 1* and 2 Refinement Lemmas

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	$x \geq_u ((x \mid s)\ \&\ (t \ll 1))$	$36_{\neq 2}$	$x \not\approx (1 - (x \ll (x-t)))$

Challenges: Large Bit-Vectors, Increasing Bit-Width

1,500 benchmarks[2] instantiated with bit-widths 16,. . . ,8192, $\sim$ 85 sat, $\sim$ 1415 unsat

Solved Benchmarks

Technique	VBS	Bit-Blast	Lazy+Layered	MCSAT	Int-Blast	PolySAT	Bit-Blast+Abstr
bw		Bitwuzla	CVC4	Yices2	cvc5	Z3	Bitwuzla
16	1,500	1,495	1,458	1,394	1,116	696	
32	1,492	1,459	1,390	1,194	1,102	672	
64	1,489	1,440	1,368	1,112	1,077	668	
128	1,490	1,433	1,308	1,076	1,017	648	
256	1,485	1,388	1,232	987	916	637	
512	1,476	1,277	1,162	916	788	620	
1,024	1,464	1,065	774	794	613	608	
2,048	1,397	844	401	668	528	576	
4,096	1,338	816	300	572	428	562	
8,192	1,314	744	202	492	389	552	
	100% $\rightarrow$ 88%	99% $\rightarrow$ 49%	97% $\rightarrow$ 13%	93% $\rightarrow$ 33%	74% $\rightarrow$ 26%	46% $\rightarrow$ 37%	

Limits: 1,200 seconds, 8GB memory

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

Challenges: Large Bit-Vectors, Increasing Bit-Width

1,500 benchmarks[2] instantiated with bit-widths 16,...,8192, $\sim$ 85 sat, $\sim$ 1415 unsat

Solved Benchmarks

Technique	**VBS**	**Bit-Blast**	**Lazy+Layered**	**MCSAT**	**Int-Blast**	**PolySAT**	**Bit-Blast+Abstr**
bw		Bitwuzla	CVC4	Yices2	cvc5	Z3	Bitwuzla
16	1,500	1,495	1,458	1,394	1,116	696	**1,494**
32	1,492	1,459	1,390	1,194	1,102	672	**1,458**
64	1,489	1,440	1,368	1,112	1,077	668	**1,454**
128	1,490	1,433	1,308	1,076	1,017	648	**1,458**
256	1,485	1,388	1,232	987	916	637	**1,456**
512	1,476	1,277	1,162	916	788	620	**1,449**
1,024	1,464	1,065	774	794	613	608	**1,434**
2,048	1,397	844	401	668	528	576	**1,365**
4,096	1,338	816	300	572	428	562	**1,300**
8,192	1,314	744	202	492	389	552	**1,274**
	100% $\rightarrow$ 88%	99% $\rightarrow$ 49%	97% $\rightarrow$ 13%	93% $\rightarrow$ 33%	74% $\rightarrow$ 26%	46% $\rightarrow$ 37%	99% $\rightarrow$ 85%

Limits: 1,200 seconds, 8GB memory

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

Evaluation: Bit-Blast+Abstr

- **Benchmarks**
  - **Smart contract verification**
    - » *certora*$_1$, *certora*$_2$ (Certora Prover)
    - » *ethereum* (hevm, Ethereum Foundation)
    - 256 bit bit-vectors
    - Theories: BV, UF, Arrays (+quantifiers)
  - **Crafted benchmarks**
    - » *syrew*
    - Controlled set to evaluate effectiveness
    - Equivalence checks for each operator
    - Enumerated by SyGuS (cvc5) for $bw = 4$
    - Instantiated for $bw = 16, 32, \ldots, 8192$
    - Theories: BV
  - **Translation validation of ZK proofs**
    - » *ff*
    - 510 bit bit-vectors
    - Theories: BV
  - **SMT-LIB**
    - » *smtlib*
    - All quantifier-free and quantified logics supported by Bitwuzla (24 in total)
    - Theories: BV, FP, UF, Arrays (+ quantifiers)

Evaluation: Bit-Blast+Abstr

- **Configurations**
  - ABSTR-T (Bitwuzla 0.3.2 + term abstraction)
  - ABSTR-TA (ABSTR-T + assertion abstraction)
  - Bitwuzla (version 0.3.2)
  - cvc5 (version 1.1.0)
  - cvc5-ib (Int-Blasting)
  - cvc5-ff (Finite Field solver)
  - Z3 (version 4.12.4)
- **Setup**
  - Limits: 1200 seconds, 8GB memory
  - Abstract $\{\cdot, \div, \mathsf{mod}\}$ of size $\geq$32

Evaluation: Bit-Blast+Abstr

Benchmarks	Solver	Solved	Timeout	Memout	Time [s]	Memory [GB]
certora$_1$ (850)	ABSTR-TA	573	231	46	448k	2,492
	ABSTR-T	258	155	437	760k	4,807
	Bitwuzla	111	86	653	915k	6,182
certora$_2$ (1,138)	ABSTR-TA	866	264	8	370k	1,024
	ABSTR-T	866	263	9	384k	1,402
	Bitwuzla	843	266	29	439k	2,944
ethereum (3,173)	ABSTR-T	3,173	0	0	407	11
	Bitwuzla	3,173	0	0	720	29

Evaluation: Bit-Blast+Abstr

Benchmarks	Solver	Solved	Timeout	Memout	Time [s]	Memory [GB]
certora$_1$ (850)	ABSTR-TA	573	231	46	448k	2,492
	ABSTR-T	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
	ABSTR-T	866	263	9	384k	1,402
	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)	ABSTR-T	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: Bit-Blast+Abstr

Benchmarks	Solver	Solved	Timeout	Memout	Time [s]	Memory [GB]
syrew (15,000)	Abstr-t	14,142	583	276	1,225k	4,409
	Bitwuzla	11,961	744	2,296	3,955k	23,483
ff (1,224)	Abstr-t	480	729	15	913k	2,762
	Bitwuzla	223	71	930	1,211k	8,360
smtlib (155,269)	Abstr-t	148,554	1,944	152	8,770k	8,566
	Bitwuzla	148,492	1,966	193	8,748k	8,953

Evaluation: Bit-Blast+Abstr

Benchmarks	Solver	Solved	Timeout	Memout	Time [s]	Memory [GB]	
syrew (15,000)	ABSTR-T	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	36 unique
	ABSTR-T	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)	ABSTR-T	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	

Evaluation: Improvements for Floating-Point Arithmetic in Set smtlib

- ▷ Bitwuzla employs word-blasting for FP logics
  - ◦ Reduction of FP to bit-vectors with SymFPU [Brain et al. 2019]
- ▷ Operations on unpacked formats require full precision arithmetic
  - ◦ Float32 `fp.mul`: 48-bit `bvmul`
  - ◦ Float64 `fp.mul`: 106-bit `bvmul`

Bit-Blast vs. **Bit-Blast+Abstr**

(Bitwuzla)

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: Abstraction Statistics over all Benchmark Sets

- **80% solved did not require bit-blasting**
  - » 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% solved required bit-blasting of abstracted terms
  - » Overall, 37% of multiplications, 13% of divisions, 2% of remainders bit-blasted
- On average 37 refinement iterations (median 4)

Takeaways

- Arithmetic is challenging for state-of-the-art approaches
  - » Arithmetic usually in combination with bitwise/shifts/word operators
  - » Requires better word-level reasoning
- Large bit-vectors don't scale well
  - » Scalability not only an issue for bit-blasting
  - » Performance of current word-level approaches suffers too
- Overall, bit-blasting still the best-performing approach
  - » Full arithmetic circuits often not required, abstraction very useful

Challenging Topics not Discussed

- ▷ Quantified bit-vectors
- ▷ Bit-vector interpolation
- ▷ Bit-vector proofs

Takeaways

- Arithmetic is challenging for state-of-the-art approaches
  - » Arithmetic usually in combination with bitwise/shifts/word operators
  - » Requires better word-level reasoning
- Large bit-vectors don't scale well
  - » Scalability not only an issue for bit-blasting
  - » Performance of current word-level approaches suffers too
- Overall, bit-blasting still the best-performing approach
  - » Full arithmetic circuits often not required, abstraction very useful

Challenging Topics not Discussed

- ▷ Quantified bit-vectors
- ▷ Bit-vector interpolation
- ▷ Bit-vector proofs

Thank you!

Check out Aina's talk on
"Scalable Bit-Blasting with Abstractions"
at **CAV on Thursday @ 9:00am**

Evaluation: Abstraction Statistics on 24,033 Solved Instances

Terms		Refinement Tier				
Operator	*Abstracted*	*1*	*2*	*3*	*4*	*Total*
·	367,101	579,369	67,221	650,086	134,525	1,431,201
÷	55,461	126,223	109,137	73,019	7,024	315,403
mod	62,328	161,270	5,614	30,350	1,326	198,560

References i

- 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.
- A. Niemetz, M. Preiner, Y. Zohar. *Scalable Bit-Blasting with Abstractions.* to appear in CAV'24, 2024.
- D. Kroening, O. Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition.*, Springer, 2016.
- 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.
- 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 ii

- 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.
- A. Zeljic, C. Wintersteiger, P. Rümmer. *Deciding Bit-Vector Formulas with mcSAT.* In Proc. of SAT'16, pages 249–266, Springer, 2016.
- Y. Zohar, A. Irfan, M. Mann, A. Niemetz, A. Nötzli, M. Preiner, A. Reynolds, C. Barrett, C.e Tinelli. *Bit-Precise Reasoning via Int-Blasting.* In Proc. of VMCAI'22, pages 496–518, Springer, 2022.
- 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. Niemetz, M. Preiner. *Ternary Propagation-Based Local Search for more Bit-Precise Reasoning.* In Proc. of FMCAD'20, pages 214–224, IEEE, 2020.

References iii

- A. Fröhlich, A. Biere, C. Wintersteiger, Y. Hamadi. In Proc. of AAAI'15, pages 1136–1143, AAAI Press, 2015.
- 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.
- 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.
- 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.
- M. Brain, F. Schanda, Y. Sun. *Building Better Bit-Blasting for Floating-Point Problems.* In Proc. of TACAS'19, 2019.

References iv

 M. Brain, L. Hadarean, D. Kroening, R. Martins. *Automatic Generation of Propagation Complete SAT Encodings.* In Proc. of VMCAI'16, 2016.