Deciding Bit-Vector Formulas with mcSAT

Aleksandar Zeljić
Uppsala University

Philipp Rümmer
Uppsala University

Christoph Wintersteiger
Microsoft Research

SMT Workshop
July 1^{st}, 2016

We present...

- A novel decision procedure for the theory of bit-vectors
  - based on mcSAT[Jovanović, de Moura, VMCAI2013]
  - avoids bit-blasting
  - preserves word-level structure
  - using tailor-made conflict driven learning

A trivial benchmark

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 29980))
(declare-fun y () (_ BitVec 29980))
(assert (and (bvuge x y)
             (bvule (bvadd x (_ bv1 29980)) y)))
```

One model: $x = 111\ldots111, y = 000\ldots000$

A trivial benchmark

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 29980))
(declare-fun y () (_ BitVec 29980))
(assert (and (bvuge x y)
             (bvule (bvadd x (_ bv1 29980)) y)))
```

One model: $x = 111\ldots111, y = 000\ldots000$

Challenging for bit-blasting!

Motivating example

Factorial computation in C

```
unsigned int factorial = 1u;
unsigned int i, n;
for (int i = n; i > 0u; i--) {
  factorial = factorial * i;
}
assert ( n <= 1 || f % 2u == 0u)
```

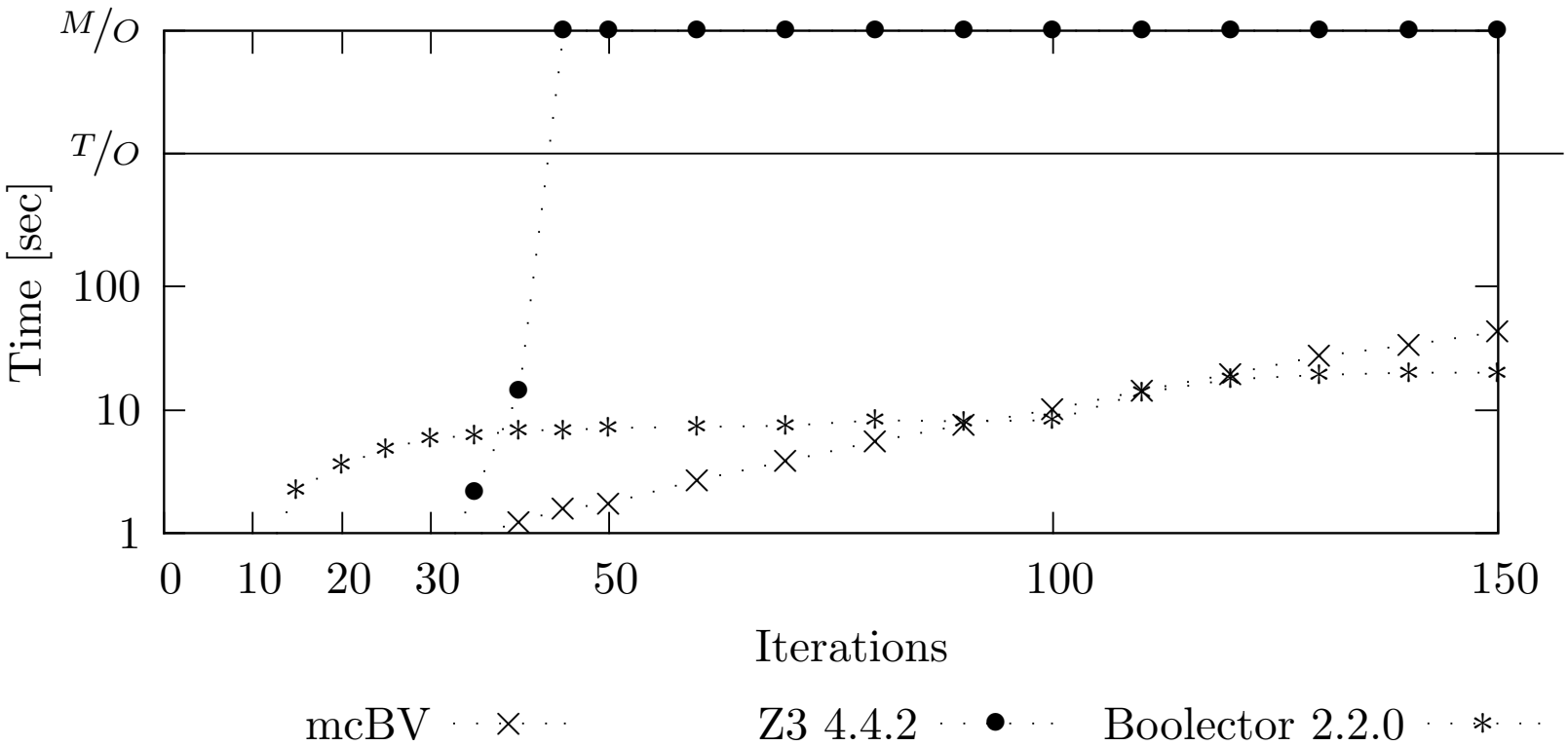
Motivating example

Factorial computation in C

```
unsigned int factorial = 1u;
unsigned int i, n;
for (int i = n; i > 0u; i--) {
  factorial = factorial * i;
}
assert ( n <= 1 || f % 2u == 0u)
```

- Prove that factorial of any number greater than 2 is even.
- Use BMC to generate formulas of increasing complexity

Performance



Example

$$x = y + z \quad \wedge \quad y <_u z \quad \wedge \quad (\quad x \leq_u y + y \quad \vee \quad y \oplus z >_u 1000 \quad)$$

Example

$$x = y + z \;\wedge\; y <_u z \;\wedge\; (\; x \leq_u y + y \;\vee\; y \oplus z >_u 1000 \;)$$

Example

$$y \mapsto 1111$$

$$\boxed{x = y + z} \;\wedge\; \boxed{y <_u z} \;\wedge\; (\; x \leq_u y + y \;\vee\; y \oplus z >_u 1000 \;)$$

Example

$y \mapsto 1111$ $\qquad$ $z \mapsto ?$

$$x = y + z \;\wedge\; y <_u z \;\wedge\; (\; x \leq_u y + y \;\vee\; y \oplus z >_u 1000 \;)$$

Example

$$x = y + z \;\wedge\; y <_u z \;\wedge\; \big(x \leq_u y + y \;\vee\; y \oplus z >_u 1000 \big)$$

$$\wedge\; y \neq 1111$$

Example

$y \mapsto 1110$ $\quad$ $z \mapsto 1111$

$$\boxed{x = y + z} \wedge \boxed{y <_u z} \wedge \left(\boxed{x \leq_u y + y} \vee y \oplus z >_u 1000 \right)$$

$$\wedge \boxed{y \neq 1111}$$

Example

$y \mapsto 1110$ $z \mapsto 1111$ $x \mapsto 1101$

$$\boxed{x = y + z} \wedge \boxed{y <_u z} \wedge \left(\boxed{x \leq_u y + y} \vee y \oplus z >_u 1000 \right)$$

$$\wedge \boxed{y \neq 1111}$$

Example

$\neg(y \geq_u 14) \vee \neg(y <_u z) \vee \neg(x = z + y) \vee \neg(x <_u y + y)$	Valid
$\neg(y \geq_u 0) \vee \neg(y <_u z) \vee \neg(x = z + y) \vee \neg(x <_u y + y)$	$(0, 1, 15)$
$\neg(y \geq_u 7) \vee \neg(y <_u z) \vee \neg(x = z + y) \vee \neg(x <_u y + y)$	$(0, 7, 9)$
$\neg(y \geq_u 10) \vee \neg(y <_u z) \vee \neg(x = z + y) \vee \neg(x <_u y + y)$	Valid
$\neg(y \geq_u 8) \vee \neg(y <_u z) \vee \neg(x = z + y) \vee \neg(x <_u y + y)$	Valid

Example

$$x = y + z \;\wedge\; y <_u z \;\wedge\; (\; x \leq_u y + y \;\vee\; y \oplus z >_u 1000 \;)$$

$$\wedge\; y \neq 1111 \;\wedge$$

$$(\; x \neq y + z \;\vee\; y \not\leq_u z \;\vee\; x \not\leq_u y + y \;\vee\; y \not\geq_u 1000 \;)$$

Example

$$x = y + z \;\wedge\; y <_u z \;\wedge\; (\; x \leq_u y + y \;\vee\; y \oplus z >_u 1000 \;)$$

$$\wedge\; y \neq 1111 \;\wedge$$

$$(\; x \neq y + z \;\vee\; y \not\leq_u z \;\vee\; x \not\leq_u y + y \;\vee\; y \not\geq_u 1000 \;)$$

Example

$y \mapsto 0111$ $z \mapsto 1111$ $x \mapsto 0110$

$$\boxed{x = y + z} \wedge \boxed{y <_u z} \wedge (x \leq_u y + y \vee y \oplus z >_u 1000)$$

$$\wedge \boxed{y \neq 1111} \wedge$$

$$(\boxed{x \neq y + z} \vee \boxed{y \not\leq_u z} \vee x \not\leq_u y + y \vee \boxed{y \not\geq_u 1000})$$

Existing methods

Bit-blasting

- encoding of BVA into propositional logic
- leverages the efficiency of SAT solvers
- scales poorly
- destroys word-level structure

Other approaches:

- preprocessing
- combining lazy and eager solvers
- encoding into arithmetic

About mcSAT

- Allows concrete variable assignments
- Flexible learning (not only in terms of literals on the trail)
- Model is readily available

The partial model philosophy

Literals on the trail are justified by model assignments

mcSAT

- The problem is a collection of clauses C to be satisfied.
- Trail represents the model M being constructed.
- Abstract transition system
- a *search* state — $\langle M, C \rangle$.
- a *conflict* state — $\langle M, C \rangle \models c$

Propositional Search Rules

Propagate				
$\langle M, C\rangle$	$\longrightarrow$	$\langle [M, c \to l], c\rangle$	**if**	$c = (l_1 \vee \ldots \vee l_m \vee l) \in C$ $\forall i : value(l_i, M) = false$ $value(l, M) = undef$
Decide				
$\langle M, C\rangle$	$\longrightarrow$	$\langle [M, l], C\rangle$	**if**	$l \in \boldsymbol{B}, value(l, M) = undef$
Conflict				
$\langle M, C\rangle$	$\longrightarrow$	$\langle M, C\rangle \vdash e$	**if**	$c \in C, value(c, M) = false$
Sat				
$\langle M, C\rangle$	$\longrightarrow$	sat	**if**	M is complete $value(c, M) = true$ for all $c \in C$
Forget				
$\langle M, C\rangle$	$\longrightarrow$	$\langle M, C \backslash \{c\}\rangle$	**if**	$c \in C$ is a learned clause

Propositional Conflict Analysis Rules

Resolve				
$\langle [M, d \to l], C \rangle \vdash c$	$\longrightarrow$	$\langle M, C \rangle \vdash r$	**if**	$\neg l \in c$ $r = resolve(c, d, l)$
Consume				
$\langle [M, d \to l], C \rangle \vdash c$	$\longrightarrow$	$\langle M, C \rangle \vdash c$	**if**	$\neg l \in c$
$\langle [M, l], C \rangle \vdash c$	$\longrightarrow$	$\langle M, C \rangle \vdash c$	**if**	$\neg l \in c$
Backjump				
$\langle [M, N], C \rangle \vdash c$	$\longrightarrow$	$\langle [M, c \to l], C \rangle$	**if**	$c = l_1 \vee \ldots \vee l_m \vee l$ $\forall i : value(L_i, M) = false$ $value(l, M) = undef$ N starts with a decision
Unsat				
$\langle M, C \rangle \vdash false$	$\longrightarrow$	$unsat$		
Learn				
$\langle M, C \rangle \vdash c$	$\longrightarrow$	$\langle M, C \cup \{c\} \rangle \vdash c$	**if**	$c \notin C$

Theory Rules

T-Propagate				
$\langle M, C\rangle$	$\longrightarrow$	$\langle [M, e \to l], C\rangle$	**if**	$l \in \boldsymbol{B},\ value(l, M) = u$ $[M, \neg l]$ is infeasible $e = explain([M, \neg l])$
T-Decide				
$\langle M, C\rangle$	$\longrightarrow$	$\langle [M, \pi(x) \mapsto \alpha], C\rangle$	**if**	x is a (theory) variable $v[M](x) = undef$ $[M, x \mapsto \alpha]$ is consisten
T-Conflict				
$\langle M, C\rangle$	$\longrightarrow$	$\langle M, C\rangle \vdash e$	**if**	M is infeasible $e = explain(M)$
T-Consume				
$\langle [M, x \mapsto \alpha], C\rangle \vdash e$	$\longrightarrow$	$\langle M, C\rangle \vdash e$	**if**	M is infeasible $e = explain(M)$
T-Backjump-Decide				
$\langle [M, x \mapsto \alpha, N], C\rangle \vdash e$	$\longrightarrow$	$\langle [M, l], C\rangle$	**if**	$c = l_1 \vee \ldots \vee l_m \vee l$ $\exists i : value(l_i, M) = und$ $value(l, M) = undef$

Ingredients to instantiate mcSAT

- a finite basis of literals, **B**
  - required for termination
- *explain* function
  - returns a valid clause which evaluates to false under the trail
  - literals are not necessarily on the trail, but must be from **B**
- *value* function
  - used to evaluate literals under a trail
  - affects strength of propagation
- propagation
  - cheap infeasability detection
  - detect model propagations
  - bit-pattern-based
  - interval-based

Model assignments for Bit-Vectors

Partial assignments

$$\pi(x) \mapsto \alpha$$

- mutually consistent

Projection functions π

- complete
- $extract_m^n$ are a natural choice for bit-vectors
- Finite basis **B** remains finite.

Modifying the Calculus

T-Decide

$$\langle M, C\rangle \quad \longrightarrow \quad \langle [M, \pi(x) \mapsto \alpha], C\rangle \quad \textbf{if} \quad \begin{array}{l} x \text{ is a (theory) variable in } C \\ Domain(x, [M, \pi(x) \mapsto \alpha]) \neq \\ \quad Domain(x, M) \\ [M, \pi(x) \mapsto \alpha] \text{ is consistent} \end{array}$$

Representing partial assignments

Bit-patterns

- strings over a 3-letter alphabet $\{0, 1, u\}$
- u stands for 0-or-1
- represents sets of values s.t.

$$matches(x, p) = \bigwedge_{\substack{0 \leq i < k \\ p_i \neq u}} x_i = p_i$$

Partial assignments with bit-patterns

$$extract_2^2(x) \mapsto 11$$

Representing partial assignments

Bit-patterns

- strings over a 3-letter alphabet $\{0, 1, u\}$
- u stands for 0-or-1
- represents sets of values s.t.

$$matches(x, p) = \bigwedge_{\substack{0 \leq i < k \\ p_i \neq u}} x_i = p_i$$

Partial assignments with bit-patterns

$$x \mapsto uu11uuuu$$

Representing partial assignments

Bit-patterns

- strings over a 3-letter alphabet $\{0, 1, u\}$
- u stands for 0-or-1
- represents sets of values s.t.

$$matches(x, p) = \bigwedge_{\substack{0 \leq i < k \\ p_i \neq u}} x_i = p_i$$

Partial assignments with bit-patterns

$$x \mapsto u^2 1^2 u^4$$

Run-length encoding

x	0^3	u^2	1^3	1^2
y	1^3	0^2	0^3	u^2
$x \oplus y$	1^3	u^2	1^3	u^2

Properties:

- Instantiates only relevant bits
- Can represent long bit-vectors
- Allows BCP-like propagation
- Complexity depends on the value representation

How do we capture arithmetic properties?

Interval over-approximation

- Every variable x is associated with an interval:

$$x \in [x_l, x_u]$$

- Relies on interval constraint propagation

Bit-patterns and intervals

- Longest common prefix of lower and upper bound induces a bit-pattern

$$x \in [01000, 01010] \rightarrow match(x, 010uu)$$

- Bit-patterns induce intervals

$$match(x, 010uu) \rightarrow x \in [01000, 01011]$$

Explanations

explain(M) function returns:

- a valid clause
- false under the trail M
- each literal belongs to the **B**

Generalizing explanations

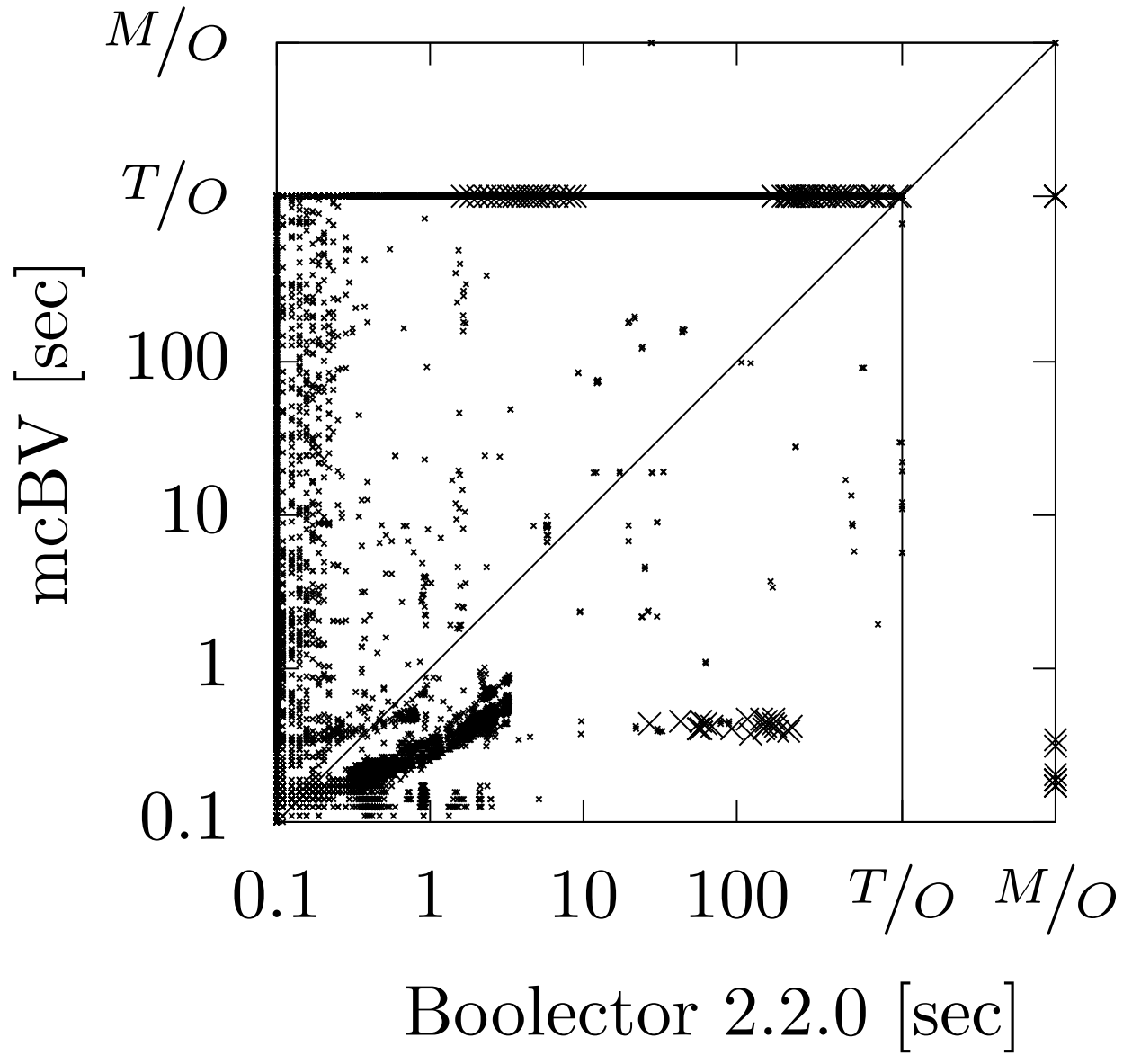
- Start from the conflict clause
- Weaken literals using a binary search over a lattice of literals
- Verify validity using a heuristic procedure *hsat*

Bit-wise generalisation example

$$M = [\ldots, y \mapsto 0^4, x \mapsto 1^1 0^1, \mathsf{extract}_2^2(y) = x]$$

$\neg matches(y, 0000) \vee \neg matches(x, 10) \vee \neg \mathsf{extract}_2^2(y) = x$	Valid
$\neg matches(y, u000) \vee \neg matches(x, 10) \vee \neg \mathsf{extract}_2^2(y) = x$	Valid
$\neg matches(y, uuu0) \vee \neg matches(x, 10) \vee \neg \mathsf{extract}_2^2(y) = x$	$(0010, 10)$
$\neg matches(y, uu00) \vee \neg matches(x, 10) \vee \neg \mathsf{extract}_2^2(y) = x$	Valid
$\neg matches(y, uu0u) \vee \neg matches(x, 10) \vee \neg \mathsf{extract}_2^2(y) = x$	Valid
$\neg matches(y, uu0u) \vee \neg matches(x, u0) \vee \neg \mathsf{extract}_2^2(y) = x$	$(0000, 00)$
$\neg matches(y, uu0u) \vee \neg matches(x, 1u) \vee \neg \mathsf{extract}_2^2(y) = x$	Valid

Evaluation



Evaluation by benchmark sets

Set		Z3 4.4.2	Boolector 2.2.0	**mcBV**	# <
QF_BV	SAT	16260	16793	6679	35
	UNSAT	30748	31534	17025	58
brummayer-biere4	SAT	10	0	10	0
	UNSAT	0	0	0	0
pspace	SAT	0	21	21	21
	UNSAT	15	60	0	0
sage	SAT	8077	8077	6069	0
	UNSAT	18530	18530	16152	29
Sage2	SAT	5104	5649	16	14
	UNSAT	9961	10612	176	29

Conclusions and Future work

- proof of concept for mcSAT
- prototype implementation
- promising results on certain types of problems

Future work:

- implement missing standard techniques
- fine tune the generalization parameters
- look into other approaches to generalization

Implementation available at:
https://github.com/Microsoft/mcBV

Questions?

Thanks for your attention!