Solving bitvectors with MCSAT: explanations from bits and pieces

Stéphane Graham-Lengrand, Dejan Jovanović, Bruno Dutertre

SRI International

IJCAR, July 2020

tl;dl (Too Long; Didn't Listen)

- MCSAT (Model-Constructing Satisfiability)
  is a scheme for SMT-solving (Satisfiability-Modulo-Theories),
  alternative to DPLL($\mathcal{T}$).

- To apply the scheme to a particular theory $\mathcal{T}$,
  you need a form of *interpolation* mechanism for $\mathcal{T}$.

- Designing an efficient mechanism for the full theory of bitvectors is difficult. So we do it for 2 fragments of the theory:
  - Equality + concatenation and extraction of bitvectors
  - A fragment of bitvector arithmetic

  Outside these fragments we use a less efficient, but generic, procedure.

- The approach is implemented in SRI's SMT-solver Yices.

- We experimented it on the SMTLib benchmarks.

1. Overview of MCSAT

The model-constructing approach to SMT-solving 1/2

MCSAT introduced in [dMJ13, JBdM13, Jov17],
inspired by multiple contributions including Conflict Resolution [KTV09] and specific decision procedures for theories such as non-linear arithmetic [JdM12].

The model-constructing approach to SMT-solving 1/2

MCSAT introduced in [dMJ13, JBdM13, Jov17], inspired by multiple contributions including Conflict Resolution [KTV09] and specific decision procedures for theories such as non-linear arithmetic [JdM12].

MCSAT tailored to theories with a standard model used for evaluating constraints (example: arithmetic)

Evaluation is a key aspect of MCSAT

The model-constructing approach to SMT-solving 1/2

MCSAT introduced in [dMJ13, JBdM13, Jov17], inspired by multiple contributions including Conflict Resolution [KTV09] and specific decision procedures for theories such as non-linear arithmetic [JdM12].

MCSAT tailored to theories with a standard model used for evaluating constraints (example: arithmetic)

Evaluation is a key aspect of MCSAT

Solving satisfiability problem

(set of constraints on variables $x_1, \ldots, x_n$)

$=$ finding values for variables $x_1, \ldots, x_n$

(so that constraints evaluate to true)

The model-constructing approach to SMT-solving 1/2

MCSAT introduced in [dMJ13, JBdM13, Jov17], inspired by multiple contributions including Conflict Resolution [KTV09] and specific decision procedures for theories such as non-linear arithmetic [JdM12].

MCSAT tailored to theories with a standard model used for evaluating constraints (example: arithmetic)

Evaluation is a key aspect of MCSAT

Solving satisfiability problem

(set of constraints on variables $x_1, \ldots, x_n$)

$=$ finding values for variables $x_1, \ldots, x_n$

(so that constraints evaluate to true)

MCSAT offers:

- a template for decision procedures
- an integration of such procedures with Boolean reasoning
- new possibilities for combining theories [JBdM13, BGLS19]

The model-constructing approach to SMT-solving 1/2

MCSAT introduced in [dMJ13, JBdM13, Jov17],
inspired by multiple contributions including Conflict Resolution [KTV09] and specific decision procedures for theories such as non-linear arithmetic [JdM12].

MCSAT tailored to theories with a standard model used for evaluating constraints (example: arithmetic)

Evaluation is a key aspect of MCSAT

Solving satisfiability problem

(set of constraints on variables $x_1, \ldots, x_n$)

$=$ finding values for variables $x_1, \ldots, x_n$

(so that constraints evaluate to true)

MCSAT offers:

- a template for decision procedures
- an integration of such procedures with Boolean reasoning
- new possibilities for combining theories [JBdM13, BGLS19]

The template is a generalisation of how CDCL works, the core calculus of SAT-solvers.

Run $=$ alternation of search phases and conflict analysis phases

The model-constructing approach to SMT-solving 2/2

- Like CDCL's trail assigns Boolean values to Boolean variables, MCSAT's trail assigns
  - Boolean values to theory atoms; these constitute *theory contraints*
  - model values to first-order variables (e.g., $x \leftarrow 3/4$)

The model-constructing approach to SMT-solving 2/2

- Like CDCL's trail assigns Boolean values to Boolean variables, MCSAT's trail assigns
  - Boolean values to theory atoms; these constitute *theory contraints*
  - model values to first-order variables (e.g., $x \leftarrow 3/4$)

- As in CDCL, MCSAT successively guesses assignments...
  ...while maintaining the invariant that
  no constraint evaluates to false according to the assignments;

The model-constructing approach to SMT-solving 2/2

- Like CDCL's trail assigns Boolean values to Boolean variables, MCSAT's trail assigns
  - Boolean values to theory atoms; these constitute *theory contraints*
  - model values to first-order variables (e.g., $x \leftarrow 3/4$)

- As in CDCL, MCSAT successively guesses assignments...
  ...while maintaining the invariant that
  no constraint evaluates to false according to the assignments;

- To pick a value for variable y after $x_1, \ldots, x_n$ were assigned values $v_1, \ldots, v_n$, simply worry about constraints over variables $x_1, \ldots, x_n, y$
  (i.e. constraints that have become unit in y)

The model-constructing approach to SMT-solving 2/2

- Like CDCL's trail assigns Boolean values to Boolean variables, MCSAT's trail assigns
  - Boolean values to theory atoms; these constitute *theory contraints*
  - model values to first-order variables (e.g., $x \leftarrow 3/4$)

- As in CDCL, MCSAT successively guesses assignments…
  …while maintaining the invariant that
  no constraint evaluates to false according to the assignments;

- To pick a value for variable y after $x_1, \ldots, x_n$ were assigned values $v_1, \ldots, v_n$, simply worry about constraints over variables $x_1, \ldots, x_n, y$
  (i.e. constraints that have become unit in y)

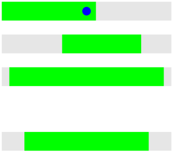
- If all variables get values while maintaining invariant: **SAT**.
  illustration on the next slide.

Search phase (satisfiable case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i

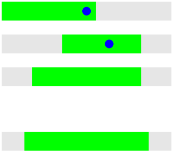
Search phase (satisfiable case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i



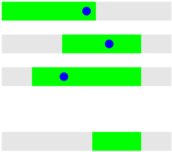
Search phase (satisfiable case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i

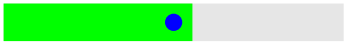


Search phase (satisfiable case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i

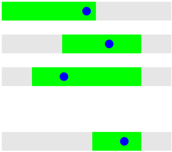


Search phase (satisfiable case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
...			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i

Search phase (satisfiable case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i



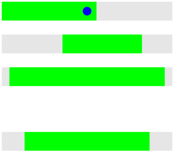
SAT

Search phase (conflict case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i

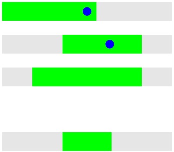
Search phase (conflict case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i



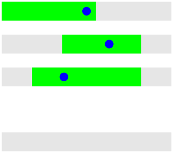
Search phase (conflict case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C^1_1, \ldots, C^1_j, \ldots$		x_1
$\{x_1, x_2\}$	$C^2_1, C^2_2, \ldots, C^2_j, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C^3_1, C^3_2, \ldots, C^3_j, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C^i_1, C^i_2, \ldots, C^i_{42}, \ldots, C^i_j, \ldots$		x_i



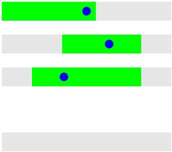
Search phase (conflict case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i



Search phase (conflict case)

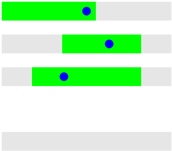
Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		x_i



Conflict

Search phase (conflict case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \ldots, C_j^1, \ldots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \ldots, C_j^2, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \ldots, C_j^3, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C_1^i, C_2^i, \ldots, C_{42}^i, \ldots, C_j^i, \ldots$		$x_i{=}y$



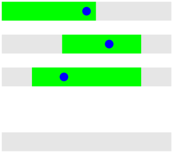
Conflict

If at any point the invariant cannot be maintained, it means:

- Some variables $x_1, \ldots, x_n$ have already been assigned values $v_1, \ldots, v_n$ (here $n = i{-}1$): this constitutes a partial model $\mathfrak{M}$;
- No value can be assigned to $y = x_i$ to extend $\mathfrak{M}$ into a model of the constraints $\{C_1, \ldots, C_m\}$ unit in y: $\mathfrak{M}$ falsifies $\exists y(C_1 \wedge \cdots \wedge C_m)$, denoted $\mathfrak{M} \not\models \exists y A$, where A is $C_1 \wedge \cdots \wedge C_m$.

Search phase (conflict case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C^1_1, \ldots, C^1_j, \ldots$		x_1
$\{x_1, x_2\}$	$C^2_1, C^2_2, \ldots, C^2_j, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C^3_1, C^3_2, \ldots, C^3_j, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C^i_1, C^i_2, \ldots, C^i_{42}, \ldots, C^i_j, \ldots$		$x_i = y$



Conflict

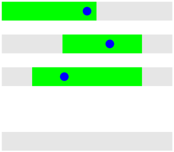
If at any point the invariant cannot be maintained, it means:

- Some variables $x_1, \ldots, x_n$ have already been assigned values $v_1, \ldots, v_n$ (here $n = i-1$): this constitutes a partial model $\mathfrak{M}$;
- No value can be assigned to $y = x_i$ to extend $\mathfrak{M}$ into a model of the constraints $\{C_1, \ldots, C_m\}$ unit in y: $\mathfrak{M}$ falsifies $\exists y(C_1 \wedge \cdots \wedge C_m)$, denoted $\mathfrak{M} \not\models \exists y A$, where A is $C_1 \wedge \cdots \wedge C_m$.

Backtrack and try new values $v'_1, \ldots, v'_n$ to assign to $x_1, \ldots, x_n$ (i.e. try another $\mathfrak{M}'$)

Search phase (conflict case)

Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C^1_1, \ldots, C^1_j, \ldots$		x_1
$\{x_1, x_2\}$	$C^2_1, C^2_2, \ldots, C^2_j, \ldots$		x_2
$\{x_1, x_2, x_3\}$	$C^3_1, C^3_2, \ldots, C^3_j, \ldots$		x_3
$\ldots$			
$\{x_1, \ldots, x_i\}$	$C^i_1, C^i_2, \ldots, C^i_{42}, \ldots, C^i_j, \ldots$		$x_i{=}y$



Conflict

If at any point the invariant cannot be maintained, it means:

- Some variables $x_1, \ldots, x_n$ have already been assigned values $v_1, \ldots, v_n$ (here $n = i{-}1$): this constitutes a partial model $\mathfrak{M}$;
- No value can be assigned to $y = x_i$ to extend $\mathfrak{M}$ into a model of the constraints $\{C_1, \ldots, C_m\}$ unit in y: $\mathfrak{M}$ falsifies $\exists y(C_1 \wedge \cdots \wedge C_m)$, denoted $\mathfrak{M} \not\models \exists y A$, where A is $C_1 \wedge \cdots \wedge C_m$.

Backtrack and try new values $v'_1, \ldots, v'_n$ to assign to $x_1, \ldots, x_n$ (i.e. try another $\mathfrak{M}'$)

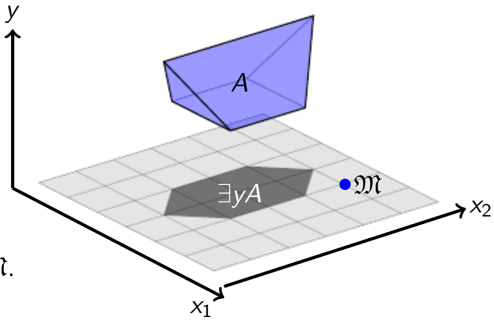
To avoid picking the same values (i.e. the same $\mathfrak{M}$) or another model $\mathfrak{M}'$ that fails "for the same reason" $\mathfrak{M}$ fails, we generalise $\mathfrak{M}$ into a class of failing models and characterise this class by a *conflict explanation*.

Conflict explanation

The conflict explanation is a quantifier-free B (with $\mathrm{fv}(B) \subseteq \{\overrightarrow{x}\}$) over-approximating $\exists y A$:

- $\mathcal{T} \models (\exists y A) \Rightarrow B$
- $\mathfrak{M} \not\models B$

B is an *interpolant* of $\exists y A$ at $\mathfrak{M}$.

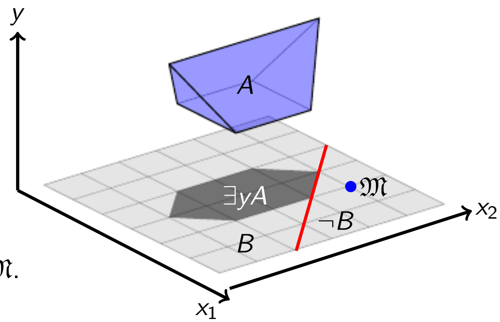


Conflict explanation

The conflict explanation is a quantifier-free B (with $\mathrm{fv}(B) \subseteq \{\overrightarrow{x}\}$) over-approximating $\exists y A$:

- $\mathcal{T} \models (\exists y A) \Rightarrow B$
- $\mathfrak{M} \not\models B$

B is an *interpolant* of $\exists y A$ at $\mathfrak{M}$.

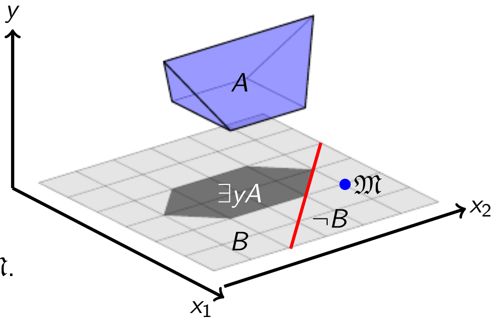


Conflict explanation

The conflict explanation is a quantifier-free B (with $\mathrm{fv}(B) \subseteq \{\overrightarrow{x}\}$) over-approximating $\exists y A$:

- $\mathcal{T} \models (\exists y A) \Rightarrow B$
- $\mathfrak{M} \not\models B$

B is an *interpolant* of $\exists y A$ at $\mathfrak{M}$.



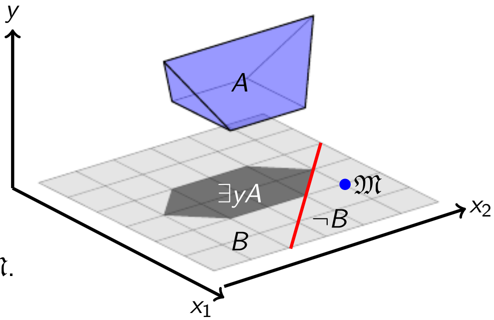
MCSAT considers the theory lemma $A \Rightarrow B$ that rules out not only $\mathfrak{M}$ but a set of similar models (we impose that B be a clause, so $A \Rightarrow B$ is a clause).

Conflict explanation

The conflict explanation is a quantifier-free B (with $\mathrm{fv}(B) \subseteq \{\overrightarrow{x}\}$) over-approximating $\exists y A$:

- $\mathcal{T} \models (\exists y A) \Rightarrow B$
- $\mathfrak{M} \not\models B$

B is an *interpolant* of $\exists y A$ at $\mathfrak{M}$.



MCSAT considers the theory lemma $A \Rightarrow B$ that rules out not only $\mathfrak{M}$ but a set of similar models (we impose that B be a clause, so $A \Rightarrow B$ is a clause).

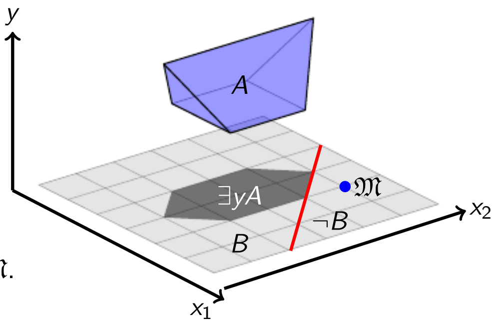
If some of the constraints in the conflict result from Boolean propagation, it performs Boolean conflict analysis on A (Boolean resolutions).

Conflict explanation

The conflict explanation
is a quantifier-free B
(with $\mathrm{fv}(B) \subseteq \{\vec{x}\}$)
over-approximating $\exists y A$:

- $\mathcal{T} \models (\exists y A) \Rightarrow B$
- $\mathfrak{M} \not\models B$

B is an *interpolant* of $\exists y A$ at $\mathfrak{M}$.



MCSAT considers the theory lemma $A \Rightarrow B$
that rules out not only $\mathfrak{M}$ but a set of similar models
(we impose that B be a clause, so $A \Rightarrow B$ is a clause).

If some of the constraints in the conflict result from Boolean propagation,
it performs Boolean conflict analysis on A (Boolean resolutions).

It backtracks to a point where $A \Rightarrow B$ is no longer violated,
e.g., B no longer evaluates (to false).

MCSAT theories

For a theory $\mathcal{T}$ to be turned into an MCSAT "plugin", we need:

- an efficient way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;
- such an explanation mechanism

MCSAT theories

For a theory $\mathcal{T}$ to be turned into an MCSAT "plugin", we need:

- an efficient way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;
- such an explanation mechanism, producing interpolants as clauses

MCSAT theories

For a theory $\mathcal{T}$ to be turned into an MCSAT "plugin", we need:

- an efficient way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;
- such an explanation mechanism,
  producing interpolants as clauses ,
  satisfying some suitable conditions for termination.

MCSAT theories

For a theory $\mathcal{T}$ to be turned into an MCSAT "plugin", we need:

- an efficient way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints;
- such an explanation mechanism,
  producing interpolants as clauses ,
  satisfying some suitable conditions for termination.

MCSAT framework is implemented in **Yices** (SRI's main SMT-solver), with plugins for Boolean, non-linear arithmetic, EUF (can be mixed), ... and now bitvectors.

2. The bitvector theory in MCSAT

Bitvectors

Traditional approach to bitvectors in SMT-solving:
Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

Bitvectors

Traditional approach to bitvectors in SMT-solving:
Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

In this paper:
our approach to turn the bitvector theory into an MCSAT plugin.

Bitvectors

Traditional approach to bitvectors in SMT-solving:
Bitvector formulae can be encoded into Boolean logic (one Boolean variable for each bit of each variable): bit blasting.

In this paper:
our approach to turn the bitvector theory into an MCSAT plugin.

On the whole SMTlib bitvector benchmarks,
MCSAT does not perform as well as long established bitblasting solvers (comparison later in this talk),
but there is a decent subset of instances where it performs better. . .

A trivial example

```
(set-info :smt-lib-version 2.6)
(set-logic QF_BV)
(set-info :source |
We verify that (x < y) -> (x + 1 <= y)
...
|)
(set-info :status unsat)
(declare-fun x () (_ BitVec 29980))
(declare-fun y () (_ BitVec 29980))
(assert (bvult x y))
(assert (bvugt (bvadd x (_ bv1 29980)) y))
(check-sat)
(exit)
```

The best 2 solvers of the SMT-comp 2019 (which use bitblasting) cannot solve this.

An MCSAT plugin for bitvectors - encoding domains

We need a nice way of representing domains of feasible values,
and how they are affected (i.e. reduced) by unit constraints:

An MCSAT plugin for bitvectors - encoding domains

We need a nice way of representing domains of feasible values, and how they are affected (i.e. reduced) by unit constraints:

Binary Decision Diagrams (BDD)

encode functions $\{0,1\}^n \rightarrow \{0,1\}$.
When considered over the bits of a n-bit bitvector variable y, a BDD can encode any set of bitvector values for y.

An MCSAT plugin for bitvectors - encoding domains

We need a nice way of representing domains of feasible values,
and how they are affected (i.e. reduced) by unit constraints:

Binary Decision Diagrams (BDD)

encode functions $\{0,1\}^n \to \{0,1\}$.
When considered over the bits of a n-bit bitvector variable y,
a BDD can encode any set of bitvector values for y.

Updating the set of feasible values when a constraint becomes unit
corresponds to computing a conjunction of 2 BDDs.

An MCSAT plugin for bitvectors - explanation mechanism

We need an explanation mechanism producing clausal interpolants (satisfying some suitable conditions for termination – easy here);

If $\exists y(C_1 \wedge \cdots \wedge C_m)$ evaluates to false in $\mathfrak{M} = \{x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n\}$ (i.e., if $v_1, \ldots, v_n$ are the values picked for $x_1, \ldots, x_n$, and $C_1, \ldots, C_m$ are the constraints that leave no feasible values for y)

- **Naive explanation mechanism**: Take
  $\neg B \;=\; x_1 \simeq v_1 \wedge \cdots \wedge x_n \simeq v_n$

An MCSAT plugin for bitvectors - explanation mechanism

We need an explanation mechanism producing clausal interpolants (satisfying some suitable conditions for termination – easy here);

If $\exists y(C_1 \wedge \cdots \wedge C_m)$ evaluates to false in $\mathfrak{M} = \{x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n\}$ (i.e., if $v_1, \ldots, v_n$ are the values picked for $x_1, \ldots, x_n$, and $C_1, \ldots, C_m$ are the constraints that leave no feasible values for y)

- **Naive explanation mechanism**: Take
  $\neg B \;=\; x_1 \simeq v_1 \wedge \cdots \wedge x_n \simeq v_n$ (only rules out $\mathfrak{M}$)

An MCSAT plugin for bitvectors - explanation mechanism

We need an explanation mechanism producing clausal interpolants (satisfying some suitable conditions for termination – easy here);

If $\exists y(C_1 \wedge \cdots \wedge C_m)$ evaluates to false in $\mathfrak{M} = \{x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n\}$ (i.e., if $v_1, \ldots, v_n$ are the values picked for $x_1, \ldots, x_n$, and $C_1, \ldots, C_m$ are the constraints that leave no feasible values for y)

- **Naive explanation mechanism**: Take
  $\neg B \;=\; x_1 \simeq v_1 \wedge \cdots \wedge x_n \simeq v_n$ (only rules out $\mathfrak{M}$)

- **Default explanation mechanism**:
  Bitblast the unsat formula $C_1 \wedge \cdots \wedge C_m \wedge x_1 \simeq v_1 \wedge \cdots \wedge x_n \simeq v_n$,
  and get an *unsat core* identifying the bits of $x_1, \ldots, x_n$ that mattered.

An MCSAT plugin for bitvectors - explanation mechanism

We need an explanation mechanism producing clausal interpolants (satisfying some suitable conditions for termination – easy here);

If $\exists y(C_1 \wedge \cdots \wedge C_m)$ evaluates to false in $\mathfrak{M} = \{x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n\}$ (i.e., if $v_1, \ldots, v_n$ are the values picked for $x_1, \ldots, x_n$, and $C_1, \ldots, C_m$ are the constraints that leave no feasible values for y)

- **Naive explanation mechanism**: Take
  $\neg B \ = \quad x_1 \simeq v_1 \wedge \cdots \wedge x_n \simeq v_n$ (only rules out $\mathfrak{M}$)

- **Default explanation mechanism**:
  Bitblast the unsat formula $C_1 \wedge \cdots \wedge C_m \wedge x_1 \simeq v_1 \wedge \cdots \wedge x_n \simeq v_n$, and get an *unsat core* identifying the bits of $x_1, \ldots, x_n$ that mattered.

  Better than the naive mechanism, but still inefficient:
  Many bit-level explanations may be needed to capture a property that could be expressed at the word level.

Word-level explanations

The difficulty is the diversity of word-level bitvector operations.

Word-level explanations

The difficulty is the diversity of word-level bitvector operations.

In the paper we identify 2 fragments of the bitvector theory for which we design nice explanation mechanisms.

Word-level explanations

The difficulty is the diversity of word-level bitvector operations.

In the paper we identify 2 fragments of the bitvector theory for which we design nice explanation mechanisms.

To produce an interpolant for $\exists y(C_1 \wedge \cdots \wedge C_m)$ at model $\mathfrak{M}$,

- we get a *conflict core* without redundant constraints, using the QuickXplain algorithm on BDDs; then
- we aggressively rewrite the remaining constraints...

...in the hope that they fit into one of these two fragments.

Word-level explanations

The difficulty is the diversity of word-level bitvector operations.

In the paper we identify 2 fragments of the bitvector theory for which we design nice explanation mechanisms.

To produce an interpolant for $\exists y(C_1 \wedge \cdots \wedge C_m)$ at model $\mathfrak{M}$,

- we get a *conflict core* without redundant constraints, using the QuickXplain algorithm on BDDs; then
- we aggressively rewrite the remaining constraints...

...in the hope that they fit into one of these two fragments.

If they don't, we use bitblasting + unsat core for an explanation.

Word-level explanations

The difficulty is the diversity of word-level bitvector operations.

In the paper we identify 2 fragments of the bitvector theory for which we design nice explanation mechanisms.

To produce an interpolant for $\exists y(C_1 \wedge \cdots \wedge C_m)$ at model $\mathfrak{M}$,

- we get a *conflict core* without redundant constraints, using the QuickXplain algorithm on BDDs; then
- we aggressively rewrite the remaining constraints...

...in the hope that they fit into one of these two fragments.

If they don't, we use bitblasting + unsat core for an explanation.

The fragments:

- **Equality with concat + extract**
- **A fragment of linear bitvector arithmetic**

Equality with concat + extract

$$
\begin{array}{llll}
\text{Constraints} & C & ::= & t \simeq t \mid t \not\simeq t \\
\text{Terms} & t & ::= & e \mid y[h{:}l] \mid t \circ t
\end{array}
$$

where e ranges over *evaluable terms*, i.e., terms without variable y (their free variables $x_1, \ldots, x_n$ have values in the current model $\mathfrak{M}$)

Equality with concat + extract

$$\begin{array}{llll} \text{Constraints} & C & ::= & t \simeq t \mid t \not\simeq t \\ \text{Terms} & t & ::= & e \mid y[h{:}l] \mid t \circ t \end{array}$$

where e ranges over *evaluable terms*, i.e., terms without variable y (their free variables $x_1, \ldots, x_n$ have values in the current model $\mathfrak{M}$)

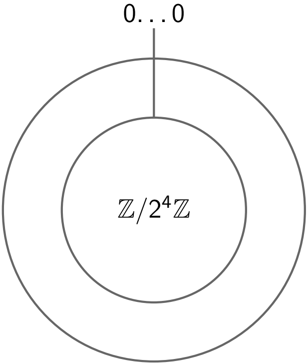
Explanation mechanism given in the paper, utilising *slicing* and model-aware E-graph.

A fragment of bitvector arithmetic - concrete example

$\mathfrak{M} = \{x_1 \leftarrow 1100, x_2 \leftarrow 1101, x_3 \leftarrow 0000\}$

Constraint C_1: $\neg(y \simeq x_1)$
Constraint C_2: $(x_1 \leq^u x_3 + y)$
Constraint C_3: $\neg(y - x_2 \leq^u x_3 + y)$

Space of values for y (feasible ones in white, forbidden ones in red):



A fragment of bitvector arithmetic - concrete example

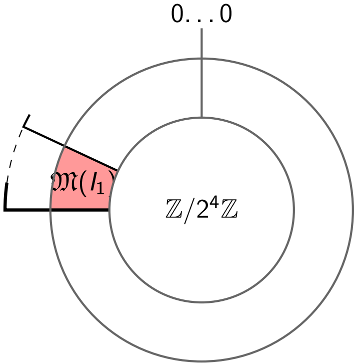
$\mathfrak{M} = \{x_1 \leftarrow 1100, x_2 \leftarrow 1101, x_3 \leftarrow 0000\}$

Constraint C_1: $\neg(y \simeq x_1)$ forbids values in interval I_1: $[x_1; x_1 + 1[$

Constraint C_2: $(x_1 \leq^u x_3 + y)$

Constraint C_3: $\neg(y - x_2 \leq^u x_3 + y)$

Space of values for y (feasible ones in white, forbidden ones in red):



A fragment of bitvector arithmetic - concrete example

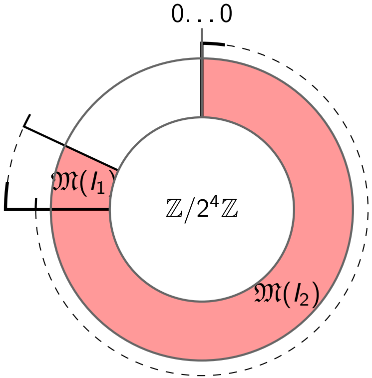
$\mathfrak{M} = \{x_1 \leftarrow 1100, x_2 \leftarrow 1101, x_3 \leftarrow 0000\}$

Constraint C_1: $\neg(y \simeq x_1)$ forbids values in interval I_1: $[x_1; x_1 + 1[$

Constraint C_2: $(x_1 \leq^u x_3 + y)$ forbids values in interval I_2: $[-x_3; x_1 - x_3[$

Constraint C_3: $\neg(y - x_2 \leq^u x_3 + y)$

Space of values for y (feasible ones in white, forbidden ones in red):

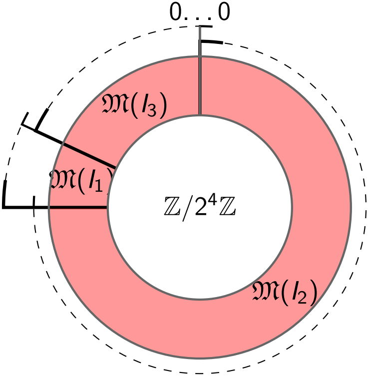


A fragment of bitvector arithmetic - concrete example

$\mathfrak{M} = \{x_1 \leftarrow 1100, x_2 \leftarrow 1101, x_3 \leftarrow 0000\}$

Constraint C_1:	$\neg(y \simeq x_1)$	forbids values in interval I_1:	$[x_1; x_1 + 1[$
Constraint C_2:	$(x_1 \leq^u x_3 + y)$	forbids values in interval I_2:	$[-x_3; x_1 - x_3[$
Constraint C_3:	$\neg(y - x_2 \leq^u x_3 + y)$	forbids values in interval I_3:	$[x_2; -x_3[$

Space of values for y (feasible ones in white, forbidden ones in red):

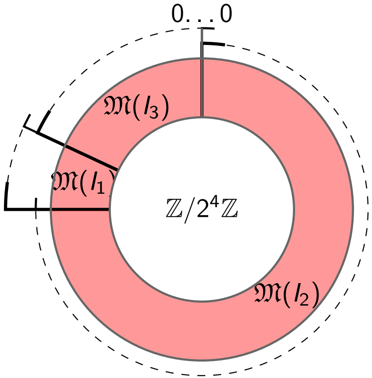


A fragment of bitvector arithmetic - concrete example

$\mathfrak{M} = \{x_1 \leftarrow 1100, x_2 \leftarrow 1101, x_3 \leftarrow 0000\}$

Constraint C_1:	$\neg(y \simeq x_1)$	forbids values in interval I_1: $[x_1; x_1 + 1[$
Constraint C_2:	$(x_1 \leq^u x_3 + y)$	forbids values in interval I_2: $[-x_3; x_1 - x_3[$
Constraint C_3:	$\neg(y - x_2 \leq^u x_3 + y)$	forbids values in interval I_3: $[x_2; -x_3[$

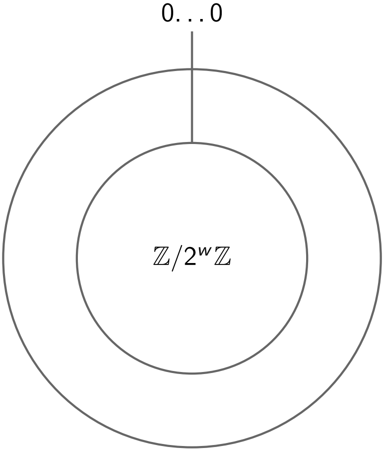
Space of values for y (feasible ones in white, forbidden ones in red):



The explanation is $(x_1{+}1) \in I_3 \ \wedge (-x_3) \in I_2 \wedge (x_1 - x_3) \in I_1$

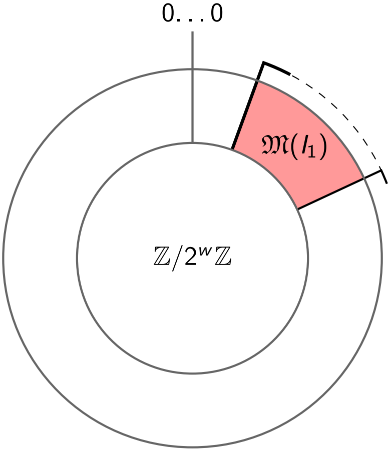
More generally on bitwidth w

Each constraint C_i forbids an interval I_i with interpretation $\mathfrak{M}(I_i)$ and upper bound u_i.



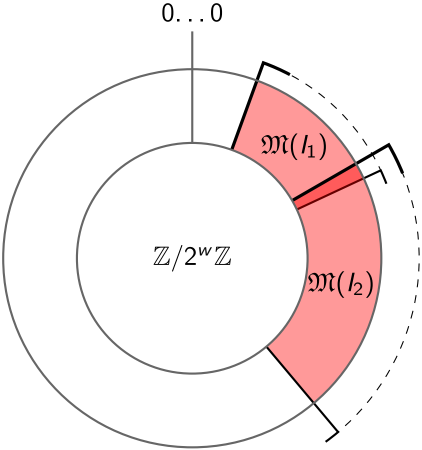
More generally on bitwidth w

Each constraint C_i forbids an interval I_i with interpretation $\mathfrak{M}(I_i)$ and upper bound u_i.



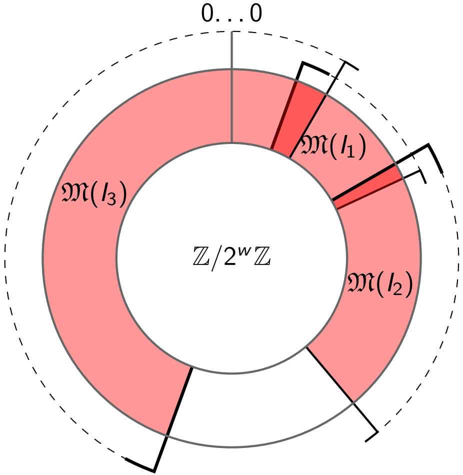
More generally on bitwidth w

Each constraint C_i forbids an interval I_i with interpretation $\mathfrak{M}(I_i)$ and upper bound u_i.



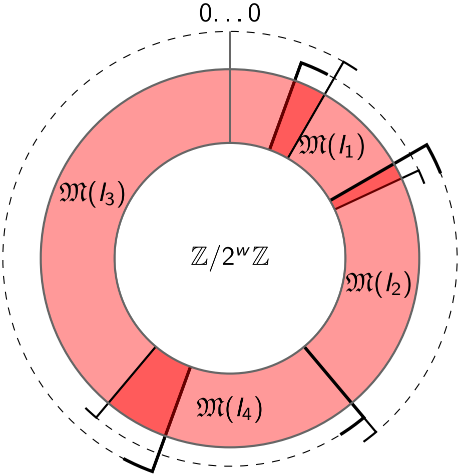
More generally on bitwidth w

Each constraint C_i forbids an interval I_i with interpretation $\mathfrak{M}(I_i)$ and upper bound u_i.



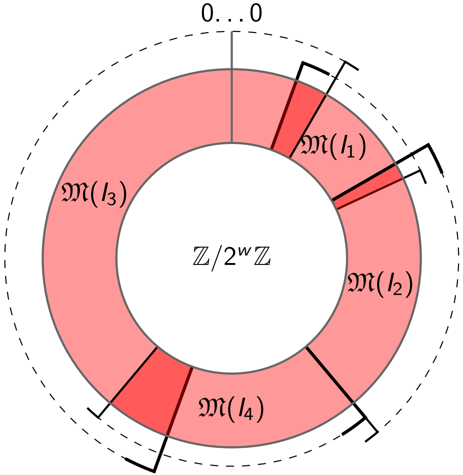
More generally on bitwidth w

Each constraint C_i forbids an interval I_i with interpretation $\mathfrak{M}(I_i)$ and upper bound u_i.



More generally on bitwidth w

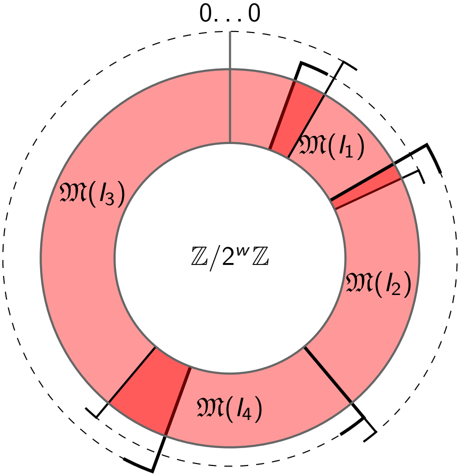
Each constraint C_i forbids an interval I_i with interpretation $\mathfrak{M}(I_i)$ and upper bound u_i.



All values in $\mathbb{Z}/2^w\mathbb{Z}$ end up being forbidden because:
$\mathfrak{M}(u_1) \in \mathfrak{M}(I_2)$ and $\mathfrak{M}(u_2) \in \mathfrak{M}(I_4)$ and $\mathfrak{M}(u_4) \in \mathfrak{M}(I_3)$ and $\mathfrak{M}(u_3) \in \mathfrak{M}(I_1)$

More generally on bitwidth w

Each constraint C_i forbids an interval I_i with interpretation $\mathfrak{M}(I_i)$ and upper bound u_i.



All values in $\mathbb{Z}/2^w\mathbb{Z}$ end up being forbidden because:
$\mathfrak{M}(u_1) \in \mathfrak{M}(I_2)$ and $\mathfrak{M}(u_2) \in \mathfrak{M}(I_4)$ and $\mathfrak{M}(u_4) \in \mathfrak{M}(I_3)$ and $\mathfrak{M}(u_3) \in \mathfrak{M}(I_1)$
The explanation is $(u_1 \in I_2) \wedge (u_2 \in I_4) \wedge (u_4 \in I_3) \wedge (u_3 \in I_1)$

Things to do in practice

- For each constraint C_i, compute the forbidden interval I_i
  (there are exactly 12 cases to consider – see Table 1 in the paper)

Things to do in practice

- For each constraint C_i, compute the forbidden interval I_i (there are exactly 12 cases to consider – see Table 1 in the paper)
- From the set $\{I_1, \ldots, I_m\}$ of intervals corresponding to constraints $C_1, \ldots, C_m$,
  extract a sequence $I_{\pi(1)}, \ldots, I_{\pi(q)}$ covering $\mathbb{Z}/2^w\mathbb{Z}$ in model $\mathfrak{M}$,
  two consecutive intervals being hooked together.

Things to do in practice

- For each constraint C_i, compute the forbidden interval I_i (there are exactly 12 cases to consider – see Table 1 in the paper)
- From the set $\{I_1, \ldots, I_m\}$ of intervals corresponding to constraints $C_1, \ldots, C_m$,
  extract a sequence $I_{\pi(1)}, \ldots, I_{\pi(q)}$ covering $\mathbb{Z}/2^w\mathbb{Z}$ in model $\mathfrak{M}$,
  two consecutive intervals being hooked together.
  In the example, the sequence is I_1, I_3, I_2

Things to do in practice

- For each constraint C_i, compute the forbidden interval I_i (there are exactly 12 cases to consider – see Table 1 in the paper)

- From the set $\{I_1, \ldots, I_m\}$ of intervals corresponding to constraints $C_1, \ldots, C_m$,
  extract a sequence $I_{\pi(1)}, \ldots, I_{\pi(q)}$ covering $\mathbb{Z}/2^w\mathbb{Z}$ in model $\mathfrak{M}$, two consecutive intervals being hooked together.
  In the example, the sequence is I_1, I_3, I_2

- Express constraints "$a \in [d; u[$" in the language of linear bv-arithmetic: $a{-}d <^u u{-}d$

Things to do in practice

- For each constraint C_i, compute the forbidden interval I_i (there are exactly 12 cases to consider – see Table 1 in the paper)

- From the set $\{I_1, \ldots, I_m\}$ of intervals corresponding to constraints $C_1, \ldots, C_m$,
  extract a sequence $I_{\pi(1)}, \ldots, I_{\pi(q)}$ covering $\mathbb{Z}/2^w\mathbb{Z}$ in model $\mathfrak{M}$, two consecutive intervals being hooked together.
  In the example, the sequence is I_1, I_3, I_2

- Express constraints "$a \in [d; u[$" in the language of linear bv-arithmetic: $a - d <^u u - d$
  In the example, the explanation
  $$(x_1 + 1) \in I_3 \ \wedge (-x_3) \in I_2 \wedge (x_1 - x_3) \in I_1$$
  is expressed as
  $$(x_1 + 1 - x_2 <^u -x_3 - x_2) \wedge (0 <^u x_1) \wedge (-x_3 <^u 1)$$

Extending the method - by enhancing the algorithm

The 12 cases of constraints turning into forbidden intervals capture the following grammar:

$$
\begin{array}{lll}
\text{Constraints} & C & ::= a \;\mid\; \neg a \\
\text{Atoms} & a & ::= e_1 + y \leq^u e_2 + y \;\mid\; e_1 \leq^u e_2 + y \;\mid\; e_1 + y \leq^u e_2
\end{array}
$$

where e_1, e_2 range over *evaluable terms*.

Extending the method - by enhancing the algorithm

We can extend the grammar into:

$$
\begin{array}{lll}
\text{Constraints} & C & ::= a \;\mid\; \neg a \\
\text{Atoms} & a & ::= e_1 + t \leq^u e_2 + t \;\mid\; e_1 \leq^u e_2 + t \;\mid\; e_1 + t \leq^u e_2 \\
\text{Terms} & t & ::= y[h{:}]
\end{array}
$$

where e_1, e_2 range over *evaluable terms*.

Generalization 1: with lower-bit extraction, leading to multiple bitwidths that the technique has to support (Algorithm 3 in the paper).

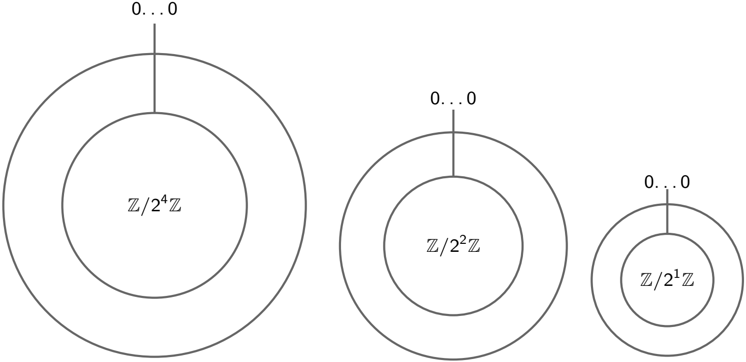
Extending the method - by enhancing the algorithm

We can extend the grammar into:

$$
\begin{array}{lll}
\text{Constraints} & C & ::= a \mid \neg a \\
\text{Atoms} & a & ::= e_1 + t \leq^u e_2 + t \mid e_1 \leq^u e_2 + t \mid e_1 + t \leq^u e_2 \\
\text{Terms} & t & ::= y[h{:}]
\end{array}
$$

where e_1, e_2 range over *evaluable terms*.

Generalization 1: with lower-bit extraction, leading to multiple bitwidths that the technique has to support (Algorithm 3 in the paper).



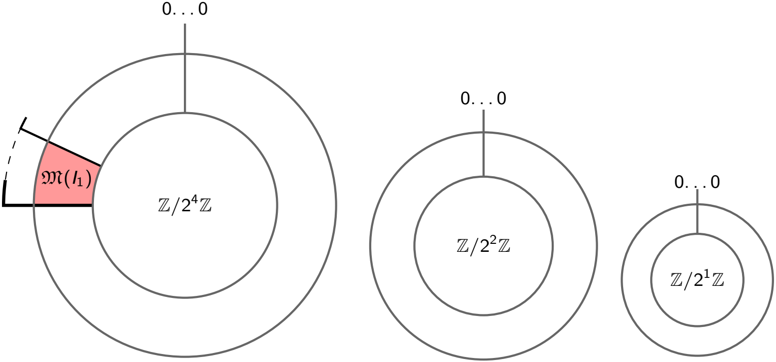
Extending the method - by enhancing the algorithm

We can extend the grammar into:

Constraints	C	$::= a \mid \neg a$
Atoms	a	$::= e_1 + t \leq^u e_2 + t \mid e_1 \leq^u e_2 + t \mid e_1 + t \leq^u e_2$
Terms	t	$::= y[h{:}]$

where e_1, e_2 range over *evaluable terms*.

Generalization 1: with lower-bit extraction, leading to multiple bitwidths that the technique has to support (Algorithm 3 in the paper).



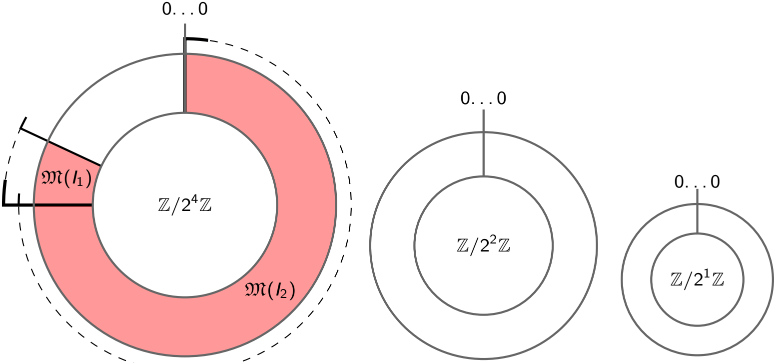
Extending the method - by enhancing the algorithm

We can extend the grammar into:

Constraints	C	$::= a \mid \neg a$
Atoms	a	$::= e_1 + t \leq^u e_2 + t \mid e_1 \leq^u e_2 + t \mid e_1 + t \leq^u e_2$
Terms	t	$::= y[h{:}]$

where e_1, e_2 range over *evaluable terms*.

Generalization 1: with lower-bit extraction, leading to multiple bitwidths that the technique has to support (Algorithm 3 in the paper).



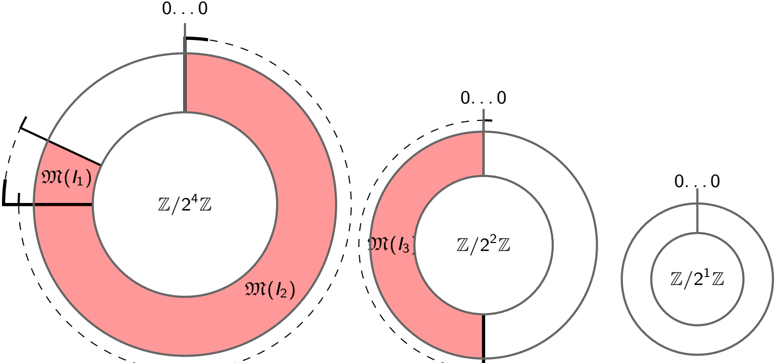
Extending the method - by enhancing the algorithm

We can extend the grammar into:

Constraints	C	$::= a \mid \neg a$
Atoms	a	$::= e_1 + t \leq^u e_2 + t \mid e_1 \leq^u e_2 + t \mid e_1 + t \leq^u e_2$
Terms	t	$::= y[h{:}]$

where e_1, e_2 range over *evaluable terms*.

Generalization 1: with lower-bit extraction, leading to multiple bitwidths that the technique has to support (Algorithm 3 in the paper).



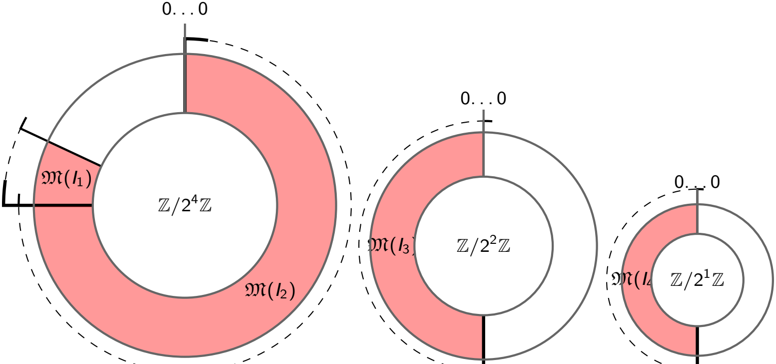
Extending the method - by enhancing the algorithm

We can extend the grammar into:

Constraints	C	$::= a \mid \neg a$
Atoms	a	$::= e_1 + t \leq^u e_2 + t \mid e_1 \leq^u e_2 + t \mid e_1 + t \leq^u e_2$
Terms	t	$::= y[h{:}]$

where e_1, e_2 range over *evaluable terms*.

Generalization 1: with lower-bit extraction, leading to multiple bitwidths that the technique has to support (Algorithm 3 in the paper).



Extending the method - by enhancing the algorithm

We can extend the grammar into:

Constraints	C	$::= a \mid \neg a$
Atoms	a	$::= e_1 + t \leq^u e_2 + t \mid e_1 \leq^u e_2 + t \mid e_1 + t \leq^u e_2$
Terms	t	$::= y[h{:}] \mid t[{:}l] \mid t + e_1 \mid -t \mid 0_k \circ t \mid t \circ 0_k$

where e_1, e_2 range over *evaluable terms*.

Generalization 2:
also with nestings of upper-bit extraction, addition of evaluable terms, negation, and concatenations with 0s (or with evaluable terms).
See Figure 1 in the paper.

Extending the method - by adding rewrites

We normalise the constraints in the conflict with the following rules (Figure 3 in the paper):

$u_1 <^s u_2$	$\rightsquigarrow \neg(u_2 \leq^s u_1)$			$u_1 \leq^s u_2$	$\rightsquigarrow u_1{+}2^{\lvert u_1\rvert-1} \leq^u u_2{+}2^{\lvert u_2\rvert-1}$	
$u_1 <^u u_2$	$\rightsquigarrow \neg(u_2 \leq^u u_1)$			$u_1 \simeq u_2$	$\rightsquigarrow u_1 - u_2 \leq^u 0$	
$u[h{:}l]$	$\rightsquigarrow u[h{:}][{:}l]$			$u[{:}l][h{:}]$	$\rightsquigarrow u[h{+}l{:}][{:}l]$	
$(u_1 \circ u_2)[{:}l]$	$\rightsquigarrow u_1[{:}l{-}\lvert u_2\rvert]$	if $\lvert u_2\rvert \leq l$		$(u_1 \circ u_2)[h{:}]$	$\rightsquigarrow u_2[h{:}]$	if $h \leq \lvert u_2\rvert$
$(u_1 \circ u_2)[{:}l]$	$\rightsquigarrow u_1 \circ u_2[{:}l]$	if not		$(u_1 \circ u_2)[h{:}]$	$\rightsquigarrow u_1[h{-}\lvert u_2\rvert{:}] \circ u_2$	if not
$2^n \times u$	$\rightsquigarrow u[\lvert u\rvert{-}n{:}] \circ 0_n$	$(n < \lvert u\rvert)$		$(u_1{+}u_2)[h{:}]$	$\rightsquigarrow u_1[h{:}] + u_2[h{:}]$	
$\mathrm{bvnot}(u)$	$\rightsquigarrow -(u+1)$			$(u_1 \times u_2)[h{:}]$	$\rightsquigarrow u_1[h{:}] \times u_2[h{:}]$	
$\pm\text{-extend}_k(u)$	$\rightsquigarrow (0_k \circ (u{+}2^{\lvert u\rvert-1})){-}(0_k \circ 2^{\lvert u\rvert-1})$			$(-u)[h{:}]$	$\rightsquigarrow -u[h{:}]$	
$u_1 \circ u_2$	$\rightsquigarrow (u_1 \circ 0_{\lvert u_2\rvert}) + (0_{\lvert u_1\rvert} \circ u_2)$					

Extending the method - by adding rewrites

We normalise the constraints in the conflict with the following rules (Figure 3 in the paper):

$u_1 <^s u_2 \rightsquigarrow \neg(u_2 \leq^s u_1)$			$u_1 \leq^s u_2 \rightsquigarrow u_1{+}2^{\lvert u_1\rvert-1} \leq^u u_2{+}2^{\lvert u_2\rvert-1}$		
$u_1 <^u u_2 \rightsquigarrow \neg(u_2 \leq^u u_1)$			$u_1 \simeq u_2 \rightsquigarrow u_1 - u_2 \leq^u 0$		
$u[h{:}l]$	$\rightsquigarrow u[h{:}][{:}l]$		$u[{:}l][h{:}]$	$\rightsquigarrow u[h{+}l{:}][{:}l]$	
$(u_1{\circ}u_2)[{:}l]$	$\rightsquigarrow u_1[{:}l{-}\lvert u_2\rvert]$	if $\lvert u_2\rvert \leq l$	$(u_1{\circ}u_2)[h{:}]$	$\rightsquigarrow u_2[h{:}]$	if $h \leq \lvert u_2\rvert$
$(u_1{\circ}u_2)[{:}l]$	$\rightsquigarrow u_1 \circ u_2[{:}l]$	if not	$(u_1{\circ}u_2)[h{:}]$	$\rightsquigarrow u_1[h{-}\lvert u_2\rvert{:}] \circ u_2$	if not
$2^n \times u$	$\rightsquigarrow u[\lvert u\rvert{-}n{:}] \circ 0_n$	$(n < \lvert u\rvert)$	$(u_1{+}u_2)[h{:}]$	$\rightsquigarrow u_1[h{:}] + u_2[h{:}]$	
$\mathrm{bvnot}(u)$	$\rightsquigarrow -(u+1)$		$(u_1 \times u_2)[h{:}]$	$\rightsquigarrow u_1[h{:}] \times u_2[h{:}]$	
$\pm\text{-extend}_k(u)$	$\rightsquigarrow (0_k{\circ}(u{+}2^{\lvert u\rvert-1})){-}(0_k{\circ}2^{\lvert u\rvert-1})$		$(-u)[h{:}]$	$\rightsquigarrow -u[h{:}]$	
$u_1{\circ}u_2$	$\rightsquigarrow (u_1{\circ}0_{\lvert u_2\rvert}) + (0_{\lvert u_1\rvert}{\circ}u_2)$				

This allows the plugin to cover (at least) the following grammar:

$$\begin{array}{lll} \text{Atoms} & a ::= & e_1 + t \lessdot e_2 + t \;\mid\; e_1 \lessdot e_2 + t \;\mid\; e_1 + t \lessdot e_2 \;\mid\; e_1 \lessdot e_2 \\ \text{Terms} & t ::= & t[h{:}l] \;\mid\; t + e_1 \;\mid\; -t \;\mid\; e_1 \circ t \;\mid\; t \circ e_1 \;\mid\; \pm\text{-extend}_k(t) \end{array}$$

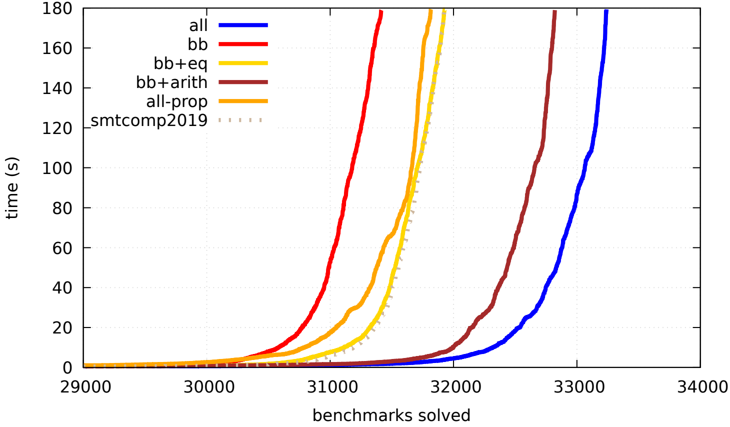
where $\lessdot$ is any comparison symbol in $\{\leq^u, <^u, \leq^s, <^s, \simeq\}$, and terms can also involve arbitrary extracts, sign-extensions, etc

3. Experimentation on the SMTLib benchmarks

Effects of explanation mechanisms and propagation

...on the 41,547 instances in SMTLib (QF_BV)

Timeout is 3 minutes



bb	only the bitblasting baseline
bb+eq	baseline + concat-extract explanation mechanism
bb+arith	baseline + arithmetic explanation mechanism
all	baseline + both mechanisms
all-prop	same as all but with no propagation of bitvector values

Numbers

Total number of instances solved by all: 33,236
(14,174 solved by pure preprocessing + 19,062 using MCSAT)

- 14,313 are solved without ever calling the default bitblasting baseline ($\simeq$ half of the benchmarks are entirely within the two fragments)
- 4,749 instances are solved by a combination of the three explainers.

Numbers

Total number of instances solved by all: 33,236
(14,174 solved by pure preprocessing + 19,062 using MCSAT)

- 14,313 are solved without ever calling the default bitblasting baseline ($\simeq$ half of the benchmarks are entirely within the two fragments)
- 4,749 instances are solved by a combination of the three explainers.

With the same 3-minute timeout,

- Yices+CadiCal solves 40,962 instances
- Boolector+CadiCal solves 40,763 instances

using bitblasting.

Numbers

Total number of instances solved by all: 33,236
(14,174 solved by pure preprocessing + 19,062 using MCSAT)

- 14,313 are solved without ever calling the default bitblasting baseline ($\simeq$ half of the benchmarks are entirely within the two fragments)
- 4,749 instances are solved by a combination of the three explainers.

With the same 3-minute timeout,

- Yices+CadiCal solves 40,962 instances
- Boolector+CadiCal solves 40,763 instances

using bitblasting.

MCSAT not as good on the whole, but in the paper we identify classes of instances where MCSAT is better, e.g.,
arithmetic explanation mechanism is insensitive to big bitwidths.
For instance, MCSAT could solve 794 instances for which Boolector+CadiCal timed out.

4. Conclusion

Related work

MCSAT approach to bitvectors first explored in [ZWR16], using

- bitvector intervals and masks to represent domains;
- eager propagation mechanisms instead of interpolation-based conflict-explanations.

Our numbers on SMTLib seem to improve on [ZWR16] quite a bit.

Related work

MCSAT approach to bitvectors first explored in [ZWR16], using

- bitvector intervals and masks to represent domains;
- eager propagation mechanisms instead of interpolation-based conflict-explanations.

Our numbers on SMTLib seem to improve on [ZWR16] quite a bit.

Our work extends preliminary work [GLJ17, GLJ19]. Main improvements:

- Use of arbitrary evaluable terms to extend the scopes of the 2 fragments;
- Generalization 2 of the arithmetic explanation mechanism;
- Normalization of conflicts by rewrite rules
- Experimentation

Future work and MCSAT beyond ground SMT-solving

- Extend the fragments little by little, e.g., handling a bigger fragment of bitvector arithmetic, e.g., with arbitrary coefficients for the conflict variable y in polynomials.

Future work and MCSAT beyond ground SMT-solving

- Extend the fragments little by little, e.g., handling a bigger fragment of bitvector arithmetic, e.g., with arbitrary coefficients for the conflict variable y in polynomials.

- Explore whether techniques used for quantified bitvector solving can help MCSAT
  - *invertibility conditions* [NPR+18]
  - other techniques inspired by *quantifier elimination*, e.g., [JC16]

Future work and MCSAT beyond ground SMT-solving

- Extend the fragments little by little, e.g., handling a bigger fragment of bitvector arithmetic, e.g., with arbitrary coefficients for the conflict variable y in polynomials.

- Explore whether techniques used for quantified bitvector solving can help MCSAT
  - *invertibility conditions* [NPR+18]
  - other techniques inspired by *quantifier elimination*, e.g., [JC16]

- Even if MCSAT ends up not performing as well as bitblasting on ground instances, it may still be interesting to produce word-level explanations of conflicts.
  Two applications of these MCSAT explanations currently investigated at SRI:
  - *General interpolation problems* in the bitvector theory
  - Solving *quantified problems* in the bitvector theory

Questions?

M. P. Bonacina, S. Graham-Lengrand, and N. Shankar.
Conflict-driven satisfiability for theory combination: Transition system and completeness.
J. of Automated Reasoning, 64(3):579–609, 2019.

L. M. de Moura and D. Jovanovic.
A model-constructing satisfiability calculus.
In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *Proc. of the 14th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'13)*, volume 7737 of *LNCS*, pages 1–12. Springer-Verlag, 2013.

S. Graham-Lengrand and D. Jovanović.
An MCSAT treatment of bit-vectors.
In M. Brain and L. Hadarean, editors, *15th Int. Work. on Satisfiability Modulo Theories (SMT 2017)*, 2017.

S. Graham-Lengrand and D. Jovanović.
Interpolating bit-vector arithmetic constraints in MCSAT.
In N. Sharygina and J. Hendrix, editors, *17th Int. Work. on Satisfiability Modulo Theories (SMT 2019)*, 2019.

D. Jovanović, C. Barrett, and L. de Moura.
The design and implementation of the model constructing satisfiability calculus.
In *Proc. of the 13th Int. Conf. on Formal Methods In Computer-Aided Design (FMCAD'13)*. FMCAD Inc., 2013.
Portland, Oregon

A. K. John and S. Chakraborty.
A layered algorithm for quantifier elimination from linear modular constraints.
Formal Methods Syst. Des., 49(3):272–323, 2016.

D. Jovanović and L. de Moura.
Solving non-linear arithmetic.
In B. Gramlich, D. Miller, and U. Sattler, editors, *Proc. of the 6th Int. Joint Conf. on Automated Reasoning (IJCAR'12)*, volume 7364 of *LNCS*, pages 339–354. Springer-Verlag, 2012.

D. Jovanović.
Solving nonlinear integer arithmetic with MCSAT.
In A. Bouajjani and D. Monniaux, editors, *Proc. of the 18th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'17)*, volume 10145 of *LNCS*, pages 330–346. Springer-Verlag, 2017.

K. Korovin, N. Tsiskaridze, and A. Voronkov.
Conflict resolution.
In I. P. Gent, editor, *Proc. of the Fifteenth Int. Conf. on Principles and Practice of Constraint Programming (CP)*, volume 5732 of *LNCS*, pages 509–523. Springer-Verlag, 2009.

A. Niemetz, M. Preiner, A. Reynolds, C. W. Barrett, and C. Tinelli.
Solving quantified bit-vectors using invertibility conditions.
In H. Chockler and G. Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 236–255. Springer, 2018.

A. Zeljic, C. M. Wintersteiger, and P. Rümmer.
Deciding bit-vector formulas with mcsat.
In N. Creignou and D. L. Berre, editors, *Proc. of the 19th Int. Conf. on Theory and Applications of Satisfiability Testing (RTA'06)*, volume 9710 of *LNCS*, pages 249–266. Springer-Verlag, 2016.