An MCSAT treatment of Bit-Vectors (work-in-progress)

Stéphane Graham-Lengrand and Dejan Jovanović
CNRS - SRI International

SMT workshop, 23rd July 2017

The model-constructing approach to SMT-solving

MCSAT introduced in [dMJ13, JBdM13, Jov17],
following work on specific decision procedures for theories such as non-linear arithmetic [JdM12].

The model-constructing approach to SMT-solving

MCSAT introduced in [dMJ13, JBdM13, Jov17],
following work on specific decision procedures for theories such as non-linear arithmetic [JdM12].

MCSAT offers:

- a template for decision procedures
- an integration of such procedures with Boolean reasoning
- new possibilities for combining procedures [JBdM13, BGLS17]

The model-constructing approach to SMT-solving

MCSAT introduced in [dMJ13, JBdM13, Jov17],
following work on specific decision procedures for theories such as non-linear arithmetic [JdM12].

MCSAT offers:

- a template for decision procedures
- an integration of such procedures with Boolean reasoning
- new possibilities for combining procedures [JBdM13, BGLS17]

The template is a generalisation of how CDCL works.
Run = alternation of search phases and conflict analysis phases
Boolean theory can be given the same status as other theories.

The model-constructing approach to SMT-solving

MCSAT introduced in [dMJ13, JBdM13, Jov17],
following work on specific decision procedures for theories such as non-linear arithmetic [JdM12].

MCSAT offers:

- a template for decision procedures
- an integration of such procedures with Boolean reasoning
- new possibilities for combining procedures [JBdM13, BGLS17]

The template is a generalisation of how CDCL works.
Run = alternation of search phases and conflict analysis phases
Boolean theory can be given the same status as other theories.

Terms and literals are created that do not belong to the input problem.

Search phase

1. For each variable awaiting a value, track the set of feasible values (those not yet ruled out by the current constraints)

Search phase

1. For each variable awaiting a value, track the set of feasible values (those not yet ruled out by the current constraints)
2. For one of the variables, pick a value out of that set.

Search phase

1. For each variable awaiting a value, track the set of feasible values (those not yet ruled out by the current constraints)
2. For one of the variables, pick a value out of that set.
3. With that choice, some of the constraints may become **unit**: all of their free variables are assigned a value but one.

Search phase

1. For each variable awaiting a value, track the set of feasible values (those not yet ruled out by the current constraints)
2. For one of the variables, pick a value out of that set.
3. With that choice, some of the constraints may become **unit**: all of their free variables are assigned a value but one.
4. See whether and how those new constraints unit in y restrict the feasible values for y, and update that set

Search phase

1. For each variable awaiting a value, track the set of feasible values (those not yet ruled out by the current constraints)
2. For one of the variables, pick a value out of that set.
3. With that choice, some of the constraints may become **unit**: all of their free variables are assigned a value but one.
4. See whether and how those new constraints unit in y restrict the feasible values for y, and update that set
5. Repeat until
   all variables are assigned values & all constraints are satisfied
   or one of these sets becomes empty: there is a conflict

Search phase

1. For each variable awaiting a value, track the set of feasible values (those not yet ruled out by the current constraints)
2. For one of the variables, pick a value out of that set.
3. With that choice, some of the constraints may become **unit**: all of their free variables are assigned a value but one.
4. See whether and how those new constraints unit in y restrict the feasible values for y, and update that set
5. Repeat until
   all variables are assigned values & all constraints are satisfied
   or one of these sets becomes empty: there is a conflict

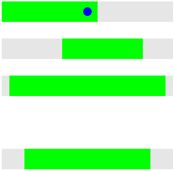
Simple process:
we only look at what the constraints say once they become unit.
Until then, we simply maintain for each constraint a watch list of variables, to detect when they become unit (as in CDCL).

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

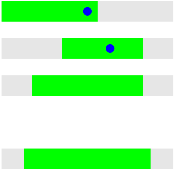
Search phase (satisfiable case)

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



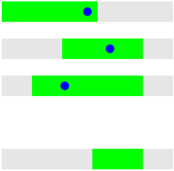
Search phase (satisfiable case)

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



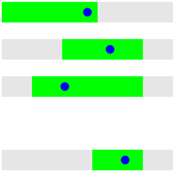
Search phase (satisfiable case)

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



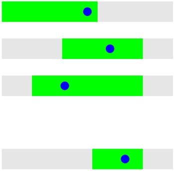
Search phase (satisfiable case)

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

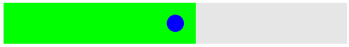


SAT

Search phase (conflict case)

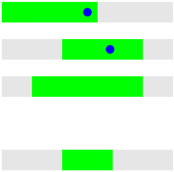
Free var within	Constraints (unit ones in red)	Feasible set	Var
$\{x_1\}$	$C_1^1, \dots, C_j^1, \dots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \dots, C_j^2, \dots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \dots, C_j^3, \dots$		x_3
$\dots$			
$\{x_1, \dots, x_i\}$	$C_1^i, C_2^i, \dots, C_{42}^i, \dots, C_j^i,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

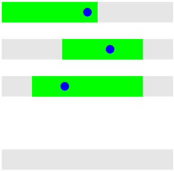
Search phase (conflict case)

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



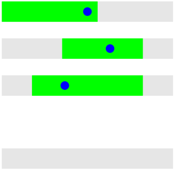
Search phase (conflict case)

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
$\{x_1\}$	$C_1^1, \dots, C_j^1, \dots$		x_1
$\{x_1, x_2\}$	$C_1^2, C_2^2, \dots, C_j^2, \dots$		x_2
$\{x_1, x_2, x_3\}$	$C_1^3, C_2^3, \dots, C_j^3, \dots$		x_3
$\dots$			
$\{x_1, \dots, x_i\}$	$C_1^i, C_2^i, \dots, C_{42}^i, \dots, C_j^i, \dots$		x_i



Conflict

Implementing the set of feasible values for y

This has to be a data-structure with operations for

1. updating the set whenever a new constraint becomes unit in y,
2. detecting when the set becomes empty, and
3. proposing a value from the feasible set.

Implementing the set of feasible values for y

This has to be a data-structure with operations for

1. updating the set whenever a new constraint becomes unit in y,
2. detecting when the set becomes empty, and
3. proposing a value from the feasible set.

This is theory-dependent:

- For LRA, this can be an interval

Implementing the set of feasible values for y

This has to be a data-structure with operations for

1. updating the set whenever a new constraint becomes unit in y,
2. detecting when the set becomes empty, and
3. proposing a value from the feasible set.

This is theory-dependent:

- For LRA, this can be an interval
- For bit-vectors, [ZWR16] use the combination of
  - an interval, e.g. $[0000, 0010]$ (understanding bitvectors in arithmetic modulo)
  - and a pattern imposing the value of some of the bits, e.g. ???1

Alternative proposition:

Use a Binary Decision Diagram (BDD) over the bits of y.

Alternative proposition:

Use a Binary Decision Diagram (BDD) over the bits of y.
BDD provide unique rep. for functions from $\{0,1\}^n$ to $\{0,1\}$.
Can exactly represent the set of feasible values for bv-variable y of length n, describing e.g. the dependencies between bits.

Alternative proposition:

Use a Binary Decision Diagram (BDD) over the bits of y.
BDD provide unique rep. for functions from $\{0,1\}^n$ to $\{0,1\}$.
Can exactly represent the set of feasible values for bv-variable y of length n, describing e.g. the dependencies between bits.

Imagine a constraint $(y <_u x)$
over two bv-variables of length 4.

Alternative proposition:

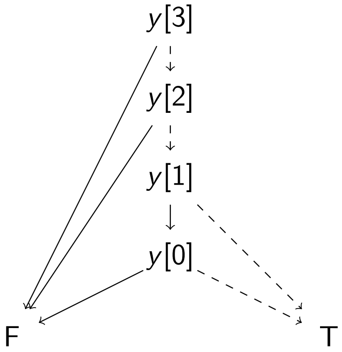
Use a Binary Decision Diagram (BDD) over the bits of y.
BDD provide unique rep. for functions from $\{0, 1\}^n$ to $\{0, 1\}$.
Can exactly represent the set of feasible values for bv-variable y of length n, describing e.g. the dependencies between bits.

Imagine a constraint $(y <_u x)$
over two bv-variables of length 4.
Setting x to 0011 makes it unit in y.

Alternative proposition:

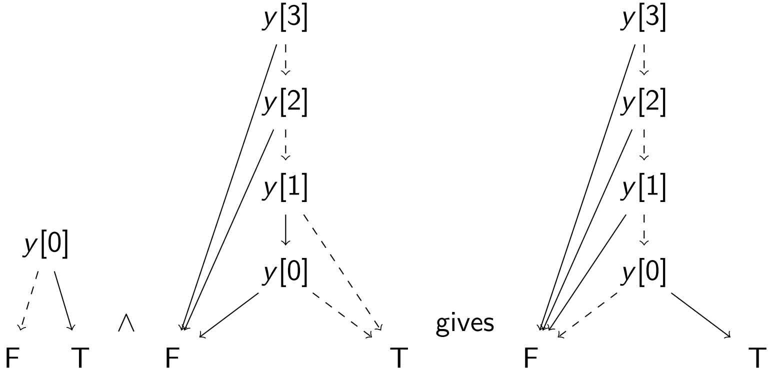
Use a Binary Decision Diagram (BDD) over the bits of y.
BDD provide unique rep. for functions from $\{0, 1\}^n$ to $\{0, 1\}$.
Can exactly represent the set of feasible values for bv-variable y of length n, describing e.g. the dependencies between bits.

Imagine a constraint $(y <_u x)$
over two bv-variables of length 4.
Setting x to 0011 makes it unit in y.
Update the BDD for y,
replacing it by its conjunction with



Using BDD

Imagine that we already knew y must satisfy pattern ???1, then



When the BDD becomes F, we have detected a conflict.

Conflict explanation in MCSAT

At this point, we have a conjunction of constraints:

$$\mathcal{A}(\vec{x}, y) = \quad A_1 \wedge \ldots \wedge A_m$$

as well as some attempted assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$ forming a partial model $\mathcal{M}$, and making $A_1, \ldots, A_m$ unit in y; and

Conflict explanation in MCSAT

At this point, we have a conjunction of constraints:

$$\mathcal{A}(\vec{x}, y) = \quad A_1 \wedge \ldots \wedge A_m$$

as well as some attempted assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$ forming a partial model $\mathcal{M}$, and making $A_1, \ldots, A_m$ unit in y; and $\mathcal{M}$ cannot be extended with a value for y in a way that satisfies $\mathcal{A}(\vec{x}, y)$.

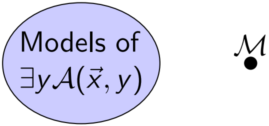
Conflict explanation in MCSAT

At this point, we have a conjunction of constraints:

$$\mathcal{A}(\vec{x}, y) = \quad A_1 \wedge \ldots \wedge A_m$$

as well as some attempted assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$ forming a partial model $\mathcal{M}$, and making $A_1, \ldots, A_m$ unit in y; and $\mathcal{M}$ cannot be extended with a value for y in a way that satisfies $\mathcal{A}(\vec{x}, y)$.

In other words, $\mathcal{M}$ falsifies $\exists y \mathcal{A}(\vec{x}, y)$.

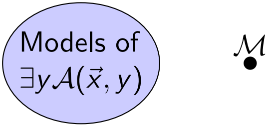


Conflict explanation in MCSAT

At this point, we have a conjunction of constraints:

$$\mathcal{A}(\vec{x}, y) = \quad A_1 \wedge \ldots \wedge A_m$$

as well as some attempted assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$ forming a partial model $\mathcal{M}$, and making $A_1, \ldots, A_m$ unit in y; and $\mathcal{M}$ cannot be extended with a value for y in a way that satisfies $\mathcal{A}(\vec{x}, y)$.
In other words, $\mathcal{M}$ falsifies $\exists y \mathcal{A}(\vec{x}, y)$.



Conflict explanation explains why that is.
Technically, by producing an interpolating clause $I(\vec{x})$ such that

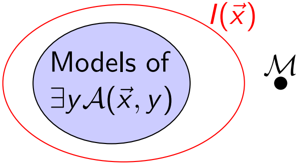
- $\mathcal{A}(\vec{x}, y) \Rightarrow I(\vec{x})$ is valid (or equivalently $(\exists y \mathcal{A}(\vec{x}, y)) \Rightarrow I(\vec{x})$)
- $\mathcal{M}$ falsifies $I(\vec{x})$

Conflict explanation in MCSAT

At this point, we have a conjunction of constraints:

$$\mathcal{A}(\vec{x}, y) = \quad A_1 \wedge \ldots \wedge A_m$$

as well as some attempted assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$ forming a partial model $\mathcal{M}$, and making $A_1, \ldots, A_m$ unit in y; and $\mathcal{M}$ cannot be extended with a value for y in a way that satisfies $\mathcal{A}(\vec{x}, y)$.
In other words, $\mathcal{M}$ falsifies $\exists y \mathcal{A}(\vec{x}, y)$.



Conflict explanation explains why that is.
Technically, by producing an interpolating clause $I(\vec{x})$ such that

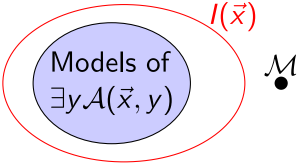
- $\mathcal{A}(\vec{x}, y) \Rightarrow I(\vec{x})$ is valid (or equivalently $(\exists y \mathcal{A}(\vec{x}, y)) \Rightarrow I(\vec{x})$)
- $\mathcal{M}$ falsifies $I(\vec{x})$

Conflict explanation in MCSAT

At this point, we have a conjunction of constraints:

$$\mathcal{A}(\vec{x}, y) = \quad A_1 \wedge \ldots \wedge A_m$$

as well as some attempted assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$ forming a partial model $\mathcal{M}$, and making $A_1, \ldots, A_m$ unit in y; and $\mathcal{M}$ cannot be extended with a value for y in a way that satisfies $\mathcal{A}(\vec{x}, y)$.
In other words, $\mathcal{M}$ falsifies $\exists y \mathcal{A}(\vec{x}, y)$.



Conflict explanation explains why that is.
Technically, by producing an interpolating clause $I(\vec{x})$ such that

- $\mathcal{A}(\vec{x}, y) \Rightarrow I(\vec{x})$ is valid (or equivalently $(\exists y \mathcal{A}(\vec{x}, y)) \Rightarrow I(\vec{x})$)
- $\mathcal{M}$ falsifies $I(\vec{x})$

Then we can analyse the conflict described by the conflict clause $\mathcal{A}(\vec{x}, y) \Rightarrow I(\vec{x})$, almost as it would be done by CDCL.

Conflict explanation for bit-vectors

An inefficient interpolant generation method:
If values can be expressed in the language (as in BV), we could take $x_1 \not\simeq v_1 \vee \ldots \vee x_n \not\simeq v_n$ as interpolant, simply ruling out model $\mathcal{M}$.

Conflict explanation for bit-vectors

An inefficient interpolant generation method:
If values can be expressed in the language (as in BV),
we could take $x_1 \not\simeq v_1 \vee \ldots \vee x_n \not\simeq v_n$ as interpolant,
simply ruling out model $\mathcal{M}$.
In BV, this would even terminate,
and provide a complete (though impractical) procedure.

Conflict explanation for bit-vectors

An inefficient interpolant generation method:
If values can be expressed in the language (as in BV),
we could take $x_1 \not\simeq v_1 \vee \ldots \vee x_n \not\simeq v_n$ as interpolant,
simply ruling out model $\mathcal{M}$.
In BV, this would even terminate,
and provide a complete (though impractical) procedure.

A default interpolant generation method for BV:
Bit-blast the conflict, i.e. turn

$$A_1 \wedge \ldots \wedge A_m \wedge (x_1 \simeq v_1) \wedge \ldots \wedge (x_n \simeq v_n)$$

into a CNF: $C_{constraints} \wedge C_{model}$ (C_{model} are unit clauses)

Conflict explanation for bit-vectors

An inefficient interpolant generation method:
If values can be expressed in the language (as in BV),
we could take $x_1 \not\simeq v_1 \vee \ldots \vee x_n \not\simeq v_n$ as interpolant,
simply ruling out model $\mathcal{M}$.
In BV, this would even terminate,
and provide a complete (though impractical) procedure.

A default interpolant generation method for BV:
Bit-blast the conflict, i.e. turn

$$A_1 \wedge \ldots \wedge A_m \wedge (x_1 \simeq v_1) \wedge \ldots \wedge (x_n \simeq v_n)$$

into a CNF: $C_{constraints} \wedge C_{model}$ (C_{model} are unit clauses)

Solve and extract an unsat core $C^{core}_{constraints} \wedge C^{core}_{model}$,
where C^{core}_{model} represent "the bits assigned by $\mathcal{M}$ that mattered"

Conflict explanation for bit-vectors

An inefficient interpolant generation method:
If values can be expressed in the language (as in BV),
we could take $x_1 \not\simeq v_1 \vee \ldots \vee x_n \not\simeq v_n$ as interpolant,
simply ruling out model $\mathcal{M}$.
In BV, this would even terminate,
and provide a complete (though impractical) procedure.

A default interpolant generation method for BV:
Bit-blast the conflict, i.e. turn

$$A_1 \wedge \ldots \wedge A_m \wedge (x_1 \simeq v_1) \wedge \ldots \wedge (x_n \simeq v_n)$$

into a CNF: $C_{constraints} \wedge C_{model}$ (C_{model} are unit clauses)

Solve and extract an unsat core $C^{core}_{constraints} \wedge C^{core}_{model}$,
where C^{core}_{model} represent "the bits assigned by $\mathcal{M}$ that mattered"

and take $I(\vec{x})$ to be the negation of C^{core}_{model}

Conflict explanation for bit-vectors

It is good to have a default mechanism that can always apply.

Conflict explanation for bit-vectors

It is good to have a default mechanism that can always apply.
Note however that the generated interpolant is at the bit level, and having an interpolant in (or closer to) the word level is desirable.

Conflict explanation for bit-vectors

It is good to have a default mechanism that can always apply.
Note however that the generated interpolant is at the bit level, and having an interpolant in (or closer to) the word level is desirable.
It seems difficult to design a conflict explanation mechanism

- that generate interpolants at the word level,
- that would work for a conjunction of bit-vector constraints with arbitrarily diverse bit-vector operations.

Conflict explanation for bit-vectors

It is good to have a default mechanism that can always apply.
Note however that the generated interpolant is at the bit level, and having an interpolant in (or closer to) the word level is desirable.
It seems difficult to design a conflict explanation mechanism

- that generate interpolants at the word level,
- that would work for a conjunction of bit-vector constraints with arbitrarily diverse bit-vector operations.

However if the constraints $\mathcal{A}(\vec{x}, y)$ conflicting with $\mathcal{M}$ live in a sub-theory of BV, then a specialised explanation mechanism may be used to provide better explanations than bit blasting.

Conflict explanation for bit-vectors

It is good to have a default mechanism that can always apply.
Note however that the generated interpolant is at the bit level, and having an interpolant in (or closer to) the word level is desirable.
It seems difficult to design a conflict explanation mechanism

- that generate interpolants at the word level,
- that would work for a conjunction of bit-vector constraints with arbitrarily diverse bit-vector operations.

However if the constraints $\mathcal{A}(\vec{x}, y)$ conflicting with $\mathcal{M}$ live in a sub-theory of BV, then a specialised explanation mechanism may be used to provide better explanations than bit blasting.

An example of specialised conflict explanation mechanism

Core of BV:

$$
\begin{array}{lcl}
A & ::= & t \simeq u \mid t \not\simeq u \\
t, u & ::= & x \mid c \mid t[h{:}l] \mid t \circ u
\end{array}
$$

An example of specialised conflict explanation mechanism

Core of BV:

$$\begin{array}{lcl} A & ::= & t \simeq u \mid t \not\simeq u \\ t, u & ::= & x \mid c \mid t[h{:}l] \mid t \circ u \end{array}$$

$\mathcal{A}(\vec{x}, y)$ made of

- a set of equalities $E = \{a_i \simeq b_i\}_{i \in \mathfrak{E}}$, and
- a set of disequalities $D = \{a_i \not\simeq b_i\}_{i \in \mathfrak{D}}$.

An example of specialised conflict explanation mechanism

Core of BV:

$$\begin{array}{lcl} A & ::= & t \simeq u \mid t \not\simeq u \\ t, u & ::= & x \mid c \mid t[h{:}l] \mid t \circ u \end{array}$$

$\mathcal{A}(\vec{x}, y)$ made of

- a set of equalities $E = \{a_i \simeq b_i\}_{i \in \mathfrak{E}}$, and
- a set of disequalities $D = \{a_i \not\simeq b_i\}_{i \in \mathfrak{D}}$.

First task for conflict explanation is to slice terms in E and D into their coarsest-base slicing [CMR97, BS09]:

An example of specialised conflict explanation mechanism

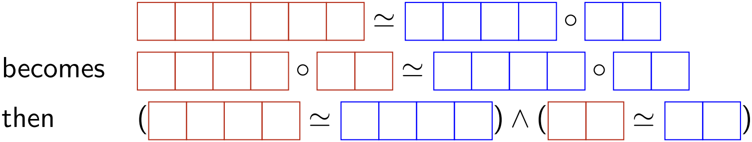
Core of BV:

$$\begin{array}{lcl} A & ::= & t \simeq u \mid t \not\simeq u \\ t, u & ::= & x \mid c \mid t[h{:}l] \mid t \circ u \end{array}$$

$\mathcal{A}(\vec{x}, y)$ made of

- a set of equalities $E = \{a_i \simeq b_i\}_{i \in \mathfrak{E}}$, and
- a set of disequalities $D = \{a_i \not\simeq b_i\}_{i \in \mathfrak{D}}$.

First task for conflict explanation is to slice terms in E and D into their coarsest-base slicing [CMR97, BS09]:



An example of specialised conflict explanation mechanism

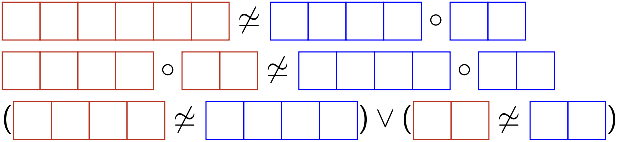
Core of BV:

$$\begin{array}{lcl} A & ::= & t \simeq u \mid t \not\simeq u \\ t, u & ::= & x \mid c \mid t[h{:}l] \mid t \circ u \end{array}$$

$\mathcal{A}(\vec{x}, y)$ made of

- a set of equalities $E = \{a_i \simeq b_i\}_{i \in \mathfrak{E}}$, and
- a set of disequalities $D = \{a_i \not\simeq b_i\}_{i \in \mathfrak{D}}$.

First task for conflict explanation is to slice terms in E and D into their coarsest-base slicing [CMR97, BS09]:



becomes

then

An example of specialised conflict explanation mechanism

Core of BV:

$$\begin{array}{lcl} A & ::= & t \simeq u \mid t \not\simeq u \\ t, u & ::= & x \mid c \mid t[h{:}l] \mid t \circ u \end{array}$$

$\mathcal{A}(\vec{x}, y)$ made of

- a set of equalities $E = \{a_i \simeq b_i\}_{i \in \mathfrak{E}}$, and
- a set of disequalities $D = \{a_i \not\simeq b_i\}_{i \in \mathfrak{D}}$.

First task for conflict explanation is to slice terms in E and D into their coarsest-base slicing [CMR97, BS09]:

No overlap of slices:

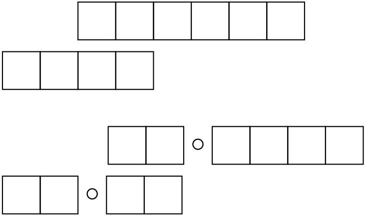
$x_i[5{:}0]$

$x_i[8{:}4]$

becomes

$x_i[5{:}4] \circ x_i[3{:}0]$

$x_i[8{:}6] \circ x_i[5{:}4]$



An example of specialised conflict explanation mechanism

Core of BV:

$$
\begin{array}{lcl}
A & ::= & t \simeq u \mid t \not\simeq u \\
t, u & ::= & x \mid c \mid t[h{:}l] \mid t \circ u
\end{array}
$$

$\mathcal{A}(\vec{x}, y)$ made of

- a set of equalities $E = \{a_i \simeq b_i\}_{i \in \mathfrak{E}}$, and
- a set of disequalities $D = \{a_i \not\simeq b_i\}_{i \in \mathfrak{D}}$.

First task for conflict explanation is to slice terms in E and D into their coarsest-base slicing [CMR97, BS09]:

No overlap of slices:

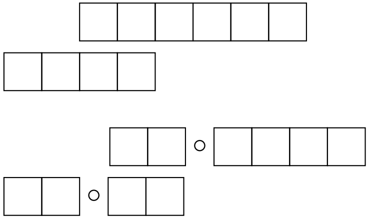
$x_i[5{:}0]$

$x_i[8{:}4]$

becomes

$x_i[5{:}4] \circ x_i[3{:}0]$

$x_i[8{:}6] \circ x_i[5{:}4]$



The more constraints are in $\mathcal{A}(\vec{x}, y)$, the thinner the slices.

In the worst case, slices $=$ bits.

An example of specialised conflict explanation mechanism

$\mathcal{A}(\vec{x}, y)$ is thus transformed into

- a set E_s of equalities, and
- a set D_s of disjunctions of disequalities

between (non-overlapping) slices of $\vec{x}$ and y, and constants.

An example of specialised conflict explanation mechanism

$\mathcal{A}(\vec{x}, y)$ is thus transformed into

- a set E_s of equalities, and
- a set D_s of disjunctions of disequalities

between (non-overlapping) slices of $\vec{x}$ and y, and constants.

$\mathcal{M}$ assigns values to $\vec{x}$-slices and constants, but cannot be extended with values for y-slices in a way that satisfies $E_s \wedge D_s$.

An example of specialised conflict explanation mechanism

$\mathcal{A}(\vec{x}, y)$ is thus transformed into

- a set E_s of equalities, and
- a set D_s of disjunctions of disequalities

between (non-overlapping) slices of $\vec{x}$ and y, and constants.

$\mathcal{M}$ assigns values to $\vec{x}$-slices and constants, but cannot be extended with values for y-slices in a way that satisfies $E_s \wedge D_s$.

In our preliminary report,
we generate the interpolant for the transformed problem:
Were it not for the cardinality constraints of bit-vectors,
it is almost a pure equality problem,
so we base our algorithm on an E-graph between slices.

Computing UNSAT cores before conflict explanation

Our specialised conflict explanation mechanism is optimised under the assumption that $\mathcal{A}(\vec{x}, y)$ is an UNSAT **core** relative to $\mathcal{M}$: Removing any constraint from $\mathcal{A}(\vec{x}, y)$, there exists a value v for y such that $\mathcal{M}, y \mapsto v$ satisfies the constraints.

Computing UNSAT cores before conflict explanation

Our specialised conflict explanation mechanism is optimised under the assumption that $\mathcal{A}(\vec{x}, y)$ is an UNSAT **core** relative to $\mathcal{M}$: Removing any constraint from $\mathcal{A}(\vec{x}, y)$, there exists a value v for y such that $\mathcal{M}, y \mapsto v$ satisfies the constraints.

When conflict is detected (the BDD for y becomes empty), the constraints that are unit in y do not necessarily form such a core: We propose to use BDDs to isolate a core $\mathcal{A}_c \subseteq \mathcal{A}$, e.g. by relying on the quick-explain mechanism [Jun01],

Computing UNSAT cores before conflict explanation

Our specialised conflict explanation mechanism is optimised under the assumption that $\mathcal{A}(\vec{x}, y)$ is an UNSAT **core** relative to $\mathcal{M}$: Removing any constraint from $\mathcal{A}(\vec{x}, y)$, there exists a value v for y such that $\mathcal{M}, y \mapsto v$ satisfies the constraints.

When conflict is detected (the BDD for y becomes empty), the constraints that are unit in y do not necessarily form such a core: We propose to use BDDs to isolate a core $\mathcal{A}_c \subseteq \mathcal{A}$, e.g. by relying on the quick-explain mechanism [Jun01],

...then decide which explanation procedure to apply depending on the fragment of BV where $\mathcal{A}_c$ lives.

Computing UNSAT cores before conflict explanation

Our specialised conflict explanation mechanism is optimised under the assumption that $\mathcal{A}(\vec{x}, y)$ is an UNSAT **core** relative to $\mathcal{M}$: Removing any constraint from $\mathcal{A}(\vec{x}, y)$, there exists a value v for y such that $\mathcal{M}, y \mapsto v$ satisfies the constraints.

When conflict is detected (the BDD for y becomes empty), the constraints that are unit in y do not necessarily form such a core: We propose to use BDDs to isolate a core $\mathcal{A}_c \subseteq \mathcal{A}$,
e.g. by relying on the quick-explain mechanism [Jun01],

. . . then decide which explanation procedure to apply depending on the fragment of BV where $\mathcal{A}_c$ lives.

The smaller $\mathcal{A}_c$ is, the higher the chances are that it lives in an isolated fragment of BV.

Computing UNSAT cores before conflict explanation

Our specialised conflict explanation mechanism is optimised under the assumption that $\mathcal{A}(\vec{x}, y)$ is an UNSAT **core** relative to $\mathcal{M}$: Removing any constraint from $\mathcal{A}(\vec{x}, y)$, there exists a value v for y such that $\mathcal{M}, y \mapsto v$ satisfies the constraints.

When conflict is detected (the BDD for y becomes empty), the constraints that are unit in y do not necessarily form such a core: We propose to use BDDs to isolate a core $\mathcal{A}_c \subseteq \mathcal{A}$, e.g. by relying on the quick-explain mechanism [Jun01],

…then decide which explanation procedure to apply depending on the fragment of BV where $\mathcal{A}_c$ lives.

The smaller $\mathcal{A}_c$ is, the higher the chances are that it lives in an isolated fragment of BV.

Even for a given conflict explanation mechanism (as with slicing), the smaller $\mathcal{A}_c$ is, the higher the chances are that our interpolant is close to the word level.

Related work

An MCSAT treatment of bit-vectors was proposed in [ZWR16]
A lot of the work there goes into propagation mechanisms,
e.g. if $(y <_u x)$ and $y \mapsto 1110$ then $x \mapsto 1111$ is propagated,
and the justifications for such propagations are recorded
(for conflict analysis).
Whereas our BDD approach relies on learning.

Related work

An MCSAT treatment of bit-vectors was proposed in [ZWR16]
A lot of the work there goes into propagation mechanisms,
e.g. if $(y <_u x)$ and $y \mapsto 1110$ then $x \mapsto 1111$ is propagated,
and the justifications for such propagations are recorded
(for conflict analysis).
Whereas our BDD approach relies on learning.

Very recently, [CBB17] suggested techniques for bit-vectors similar to MCSAT. Shares with [ZWR16] the use of patterns (e.g. ?0?1) to record constraints on bv-variables, and recording justifications for why some of the bits have been assigned.

Conclusion and Further work

- To do:
  identify conflict explanation mechanisms for other fragments
  e.g. [JW16] should provide an conflict explanation mechanism specialised to bit-vector arithmetic. Details to be checked.

Conclusion and Further work

- To do:
  identify conflict explanation mechanisms for other fragments
  e.g. [JW16] should provide an conflict explanation mechanism specialised to bit-vector arithmetic. Details to be checked.
- Implementation is ongoing $\Longrightarrow$ no experimental results yet

Conclusion and Further work

- To do:
  identify conflict explanation mechanisms for other fragments
  e.g. [JW16] should provide an conflict explanation mechanism specialised to bit-vector arithmetic. Details to be checked.
- Implementation is ongoing $\Longrightarrow$ no experimental results yet
- Using BDD to solve arbitrary problems in Boolean logic can be slow, especially as in the case of bit-vectors, the number of variables can be huge.
  Here, we limit their use to the bit variables of a single bv-variable, so their size is controlled.

Conclusion and Further work

- To do:
  identify conflict explanation mechanisms for other fragments
  e.g. [JW16] should provide an conflict explanation mechanism specialised to bit-vector arithmetic. Details to be checked.
- Implementation is ongoing $\Longrightarrow$ no experimental results yet
- Using BDD to solve arbitrary problems in Boolean logic can be slow, especially as in the case of bit-vectors, the number of variables can be huge.
  Here, we limit their use to the bit variables of a single bv-variable, so their size is controlled.
- BDD have also been proposed as an approach to quantified bit-vector formulae, with Q3B implementation [JS16].
  To do: look at quantified problems, as one key ingredient of MCSAT, namely producing an interpolant $I(\vec{x})$ for $\exists y \mathcal{A}(\vec{x}, y)$ with respect to a model $\mathcal{M}$ for $\vec{x}$, relates to quantifier elimination.
  Investigating the connection with [BJ15] is on our agenda.

Thank you!

More on related works 1/2

In [ZWR16], a lot of the work goes into propagation mechanisms, e.g. if $(y <_u x)$ and $y \mapsto 1110$ then $x \mapsto 1111$ is propagated, and the justifications for such propagations are recorded (for conflict analysis).

More on related works 1/2

In [ZWR16], a lot of the work goes into propagation mechanisms, e.g. if $(y <_u x)$ and $y \mapsto 1110$ then $x \mapsto 1111$ is propagated, and the justifications for such propagations are recorded (for conflict analysis).

In our setting, such propagations correspond to situations where the BDD for a variable becomes a singleton.
The assignment, here $x \mapsto 1111$, can then also be propagated, but the justification for it is not readily available.
It will come up later and on demand,
when looking for an explanation of a conflict involving $x \mapsto 1111$.

More on related works 1/2

In [ZWR16], a lot of the work goes into propagation mechanisms, e.g. if $(y <_u x)$ and $y \mapsto 1110$ then $x \mapsto 1111$ is propagated, and the justifications for such propagations are recorded (for conflict analysis).

In our setting, such propagations correspond to situations where the BDD for a variable becomes a singleton.
The assignment, here $x \mapsto 1111$, can then also be propagated, but the justification for it is not readily available.
It will come up later and on demand,
when looking for an explanation of a conflict involving $x \mapsto 1111$.

The two approaches are not incompatible:
If a specific propagation rule can apply with a readily available justification, record the justification. Otherwise propagate the value when the BDD becomes a singleton, without justification.

More on related works 2/2

In [ZWR16], another part of the work goes into generalising conflicts, so that they rule out as many models as possible: When $x \mapsto v$ led to a conflict, see if the conflict still holds

- by widening v into an interval containing v;
- by unassigning some of the bits in v.

More on related works 2/2

In [ZWR16], another part of the work goes into generalising conflicts, so that they rule out as many models as possible: When $x \mapsto v$ led to a conflict, see if the conflict still holds

- by widening v into an interval containing v;
- by unassigning some of the bits in v.

We hope that this will no longer be necessary with specialised conflict explanation mechanisms whose role is to describe what was wrong with $x \mapsto v$.

M. P. Bonacina, S. Graham-Lengrand, and N. Shankar.
Satisfiability modulo theories and assignments.
In L. de Moura, editor, *Proc. of the 26th Int. Conf. on Automated Deduction (CADE'17)*, volume 10395 of *LNAI*. Springer-Verlag, 2017

N. Bjorner and M. Janota.
Playing with quantified satisfaction.
In M. Davis, A. Fehnker, A. McIver, and A. Voronkov, editors, *Proc. of the the 20th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'15)*, volume 9450 of *LNCS*. Springer-Verlag, 2015.

R. Bruttomesso and N. Sharygina.
A scalable decision procedure for fixed-width bit-vectors.
In *Proceedings of the 2009 International Conference on Computer-Aided Design*, ICCAD'09, pages 13–20. ACM, 2009.

Z. Chihani, F. Bobot, and S. Bardin.
CDCL-inspired Word-level Learning for Bit-vector Constraint Solving.
2017.
Preprint.
Available at
https://hal.archives-ouvertes.fr/hal-01531336

D. Cyrluk, O. Möller, and H. Rueß.
An efficient decision procedure for the theory of fixed-sized bit-vectors.
In O. Grumberg, editor, *Computer Aided Verification: 9th International Conference, CAV'97 Haifa, Israel, June 22–25, 1997 Proceedings*, pages 60–71. Springer Berlin Heidelberg, 1997.

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.

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

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.

M. Jonáš and J. Strejček.
Solving quantified bit-vector formulas using binary decision diagrams.
In N. Creignou and D. Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016,*

Proceedings, pages 267–283. Springer International Publishing, 2016.

U. Junker.
Quickxplain: Conflict detection for arbitrary constraint propagation algorithms.
In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, 2001.

M. Janota and C. M. Wintersteiger.
On intervals and bounds in bit-vector arithmetic.
In T. King and R. Piskac, editors, *Proc. of the 14th Int. Work. on Satisfiability Modulo Theories (SMT'16)*, volume 1617 of *CEUR Workshop Proceedings*, pages 81–84. CEUR-WS.org, 2016

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.