An MCSAT treatment of Bit-Vectors (preliminary report)

Stéphane Graham-Lengrand, Dejan Jovanović

An MCSAT treatment of Bit-Vectors (preliminary report)

Stéphane Graham-Lengrand[1,2] and Dejan Jovanović[1]

[1] SRI International
[2] CNRS - INRIA - École Polytechnique

Abstract

We propose a general scheme for treating the theory of bit-vectors ($\mathcal{BV}$) in the MCSAT framework, complementing the approach by Zeljić, Wintersteiger, and Rümmer. MCSAT assigns values to first-order variables. In order to keep track of the set of feasible values for a given bit-vector variable, we propose the use of Binary Decision Diagram. This allows an assignment mechanism that is generic for $\mathcal{BV}$. When a conflict arises, involving some of the constraints and some of the assignments made so far, MCSAT must produce an explanation for the conflict. This mechanism can be specialized according to the constraints involved in the conflict. We propose an explanation mechanism that applies when these constraints are in the core fragment of $\mathcal{BV}$, based on slicing and equality reasoning. We plan to add support for more $\mathcal{BV}$ fragments in the future.

1 Introduction

In this preliminary report of work in progress, we broach the theory of bit-vectors ($\mathcal{BV}$) in the context of *model-constructing satisfiability* (MCSAT), an approach to SMT-solving proposed in [dMJ13, JBdM13, Jov17]. MCSAT builds on the mechanisms of CDCL [MLM09], adapting them to first-order terms and first-order theories. A treatment of $\mathcal{BV}$ in the MCSAT framework was proposed by Zeljić, Wintersteiger, and Rümmer in [ZWR16]. The present work in progress explores some ideas that can complement their approach.

In MCSAT, the range of values that can be assigned to a first-order variable is restricted by the constraints to be satisfied. This restriction mechanism is theory-specific, and finding a suitable representation to express such a range is a key step in integrating this theory into the MCSAT framework. The approach proposed in [ZWR16] for $\mathcal{BV}$ uses *intervals* and *patterns* to express restrictions on the range of feasible values for a given bit-vector variable.

The first idea that we are exploring is the use of a more general representation to record the range of feasible values for a given variable, namely Binary Decision Diagrams (BDD) [Bry86] over the bits of the variable. BDDs can express the exact range of feasible values, regardless of the nature of the bit-vector constraints to be satisfied. For instance, BDDs can detect when the range becomes a singleton, in which case MCSAT can propagate the assignment of the last feasible value to the variable. BDDs offer a generic mechanism for proposing and propagating values; the question of specializing the treatment of bit-vectors to specific views of what these bit-vectors represent (e.g. integers), can be limited to the other main mechanism of MCSAT: *conflict explanation*.

Indeed, MCSAT also features a notion of *conflict* that generalizes that of CDCL. Typically, a conflict arises when the range of values for a variable becomes empty (a situation easily detected by BDDs): the assignments made so far do not lead to a model. MCSAT must produce a symbolic *explanation* for a conflict, so that some of the assignments made so far can be undone and replaced by new ones, taking the explanation into account.

The generation of the explanation is theory-specific. For $\mathcal{BV}$, two mechanisms were proposed in [ZWR16] to make a conflict explanation rule out more potential assignments than those that were actually used and that led to the conflict. The first one consists in generalizing a value

that was assigned into an interval containing the value. The second one consists in generalizing a value that was assigned by unassigning some of its bits.

These are examples of specialized mechanisms for conflict explanation, which can work better than what we consider as the default explanation mechanism: bit-blasting the constraints that are involved in the conflict. If the said constraints live in a particular fragment of $\mathcal{BV}$, alternative mechanisms can produce high-level explanations that generalize better. We describe such a mechanism for the core fragment of $\mathcal{BV}$ made of concatenation, extraction and equality. An MCSAT explanation is produced out of pure equality reasoning: the involved constraints are first transformed according to *coarsest-base slicing*, as described in e.g. [CMR97, BS09]. We adapt the approach to the generation of conflict explanations for MCSAT.

In Section 2 we review MCSAT, present the BDD approach, and the general considerations for conflict explanation. In Section 3 we present the dedicated conflict explanation method for the core fragment of $\mathcal{BV}$.

2 General scheme for bit-vectors

2.1 Review of MCSAT

MCSAT takes as input a quantifier-free formula and determines whether the formula is satisfiable in a given theory or a combination thereof. In this paper we consider the case where these formulas are built from the symbols of the bit-vectors theory and from bit-vector variables. The problem is satisfiable if we can assign a bit-vector value to every free variable appearing in the problem, in a way that makes the input formula evaluate to true (using the standard model and standard interpretation of symbols). We call such an assignment a *model*.

To determine satisfiability of the input, MCSAT builds on the ideas of CDCL [MLM09], whose rules are used to handle the Boolean structure of the input. CDCL attempts to construct a Boolean model by building a partial candidate model as sequence of literals, called *the trail*, elements of which are either decisions on the value of a variable, or an assignment that is a results of a Boolean inference (called *propagation*).

MCSAT generalizes the notion of trail to allow first-order decisions and propagations (see, e.g., [Jov17]). For example, in the context of bit-vectors, a decision $x \mapsto 0010$ for a bit-vector variable x, is allowed on the trail. As in the case of CDCL, during a run of MCSAT, the trail gets extended with decisions and propagations, and gets pruned when MCSAT backtracks. An example of an MCSAT trail containing bit-vector elements is

$$[\![\ (x <_u 0011) \mapsto \top,\ \ x \mapsto 0010,\ \ (y \circ y = x) \overset{C}{\rightsquigarrow} \top\ ]\!] \quad .$$

Above, the element $(x <_u 0011) \mapsto \top$ is a Boolean decision assigning the literal to true, $x \mapsto 0010$ is a bit-vector decision assigning the variable x to a bit-vector value, and the element $(y \circ y = x) \overset{C}{\rightsquigarrow} \top$ represents a Boolean propagation from clause C, assuming that the input problem contains a clause such as $C \equiv (x <_u 0011) \Rightarrow (y \circ y = x)$. As in the case of CDCL, we keep the clause C as a decoration of literal $y \circ y = x$ in the trail.

The assignments in the trail define a partial model that can be used to evaluate literals. For instance, the above trail, as a partial model, evaluates $x <_u 0011$ to true, but does not evaluate $y \circ y = x$, because the free variable y is unassigned.

In the CDCL calculus, the search process aims to maintain the invariant that no literal in the problem is implied (either by decision, or unit propagation) to have two different values. Otherwise, a *conflict* is detected, which triggers conflict analysis, backtracking, and a revision of

the search. This also holds in the MCSAT calculus, but in a more general setting that also takes into account the evaluation of theory-specific literals. For instance, in the example above, since the literal $(x <_u 0011)$ is assigned to true, the calculus is only allowed to pick values for x that are consistent with this constraint. More generally, in order to ensure consistency, a variable x can only be assigned to a value that is consistent with *all* literals in the trail where x is the only unassigned variable (*unit*). In our example, the decision that follows $(x \mapsto 0010)$, entails that the literal $(x <_u 0011)$ evaluates to true, which is consistent with the previous assignment for $(x <_u 0011)$. On the other hand, the assignment $x \mapsto 0011$ would not be allowed since this would imply two different interpretations for the literal $(x <_i 0011)$ (both true and false). Now, consider the literal $(y \circ y = x)$ that is propagated to be true. In order to complete the partial model with an assignment for y, we need to make sure that the literal $(y \circ y = x)$ can not also evaluate to false. Unfortunately, due to the value of x in the trail, any value that we could pick for y would make the literal $(y \circ y = x)$ evaluate to false. So no assignment for y is possible, and MCSAT is in a *conflict* state, from where we must backtrack on the decisions that we made.

As in CDCL, backtracking removes at least one of the decisions that were made in the trail, and performs a conflict analysis. We learn from this conflict analysis a *conflict clause*, symbolically describing the conflict in terms of the previous decisions that were made. Taking this new clause into account avoids making the same faulty decisions in the future model construction attempts. After finitely many attempts, MCSAT will either succeed in proposing a full model without raising a conflict, or find a conflict that involves no decision but only the input problem.

2.2 Using BDDs to represent sets of feasible values

Detecting the situation where no value for a given variable is feasible with respect to the trail is a key feature of MCSAT. For this, for each unassigned variable y, we collect all the constraints that can be evaluated as soon as y gets a value, i.e., all constraints whose only unassigned free variable is y, a.k.a. the *unit constraints in* y. Each of those constraints restricts the range of possible values whose assignment to y would be feasible. The conjunction of those unit constraints may restrict the range of feasible values to a singleton set, which leaves us no choice regarding the value to assign to y and we can propagate it; it may also restrict it to an empty set, which would prompt us to backtrack, as in the example above.

A key objective of an MCSAT implementation for a particular theory is to devise a data-structure that can represent the feasible set of a variable y, with efficient 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 data-structure depends on the theory being considered.

For instance, if instead of the theory of bit-vectors we were working in the theory of linear arithmetic, a typical constraint would be an inequality between a linear combination of arithmetic variables and a constant, i.e. $a_1x_1 + \cdots a_nx_n + ay \lessdot c$ with $\lessdot \in \{<, \leq\}$. When this constraint becomes unit in y, it means that values have been assigned to $x_1, \ldots, x_n$, and the inequality simply restricts the values for y by imposing an upper or lower bound (depending on the sign of a). Hence, the conjunction of inequality constraints that are unit in y define the set of feasible values as an interval, with at most two of those constraints being useful, namely that imposing the lowest upper bound (if there is one) and that imposing the greatest lower bound (if there is one). Then it is easy to check whether this interval is a singleton or empty.

Coming back to the theory of bit-vectors, Zeljić, Wintersteiger and Rümmer [ZWR16] rep-

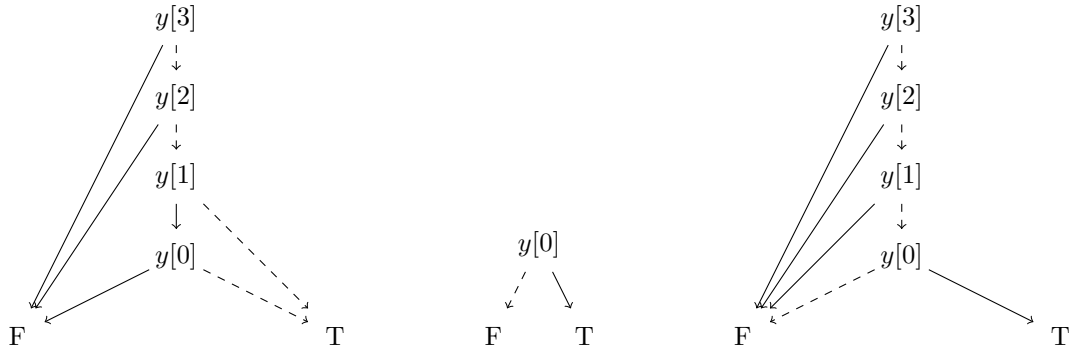

Figure 1: BDDs representing the feasible space of two constraints $C_1 \equiv (y <_u x)$ and $C_2 \equiv (y[0{:}0] \simeq x[0{:}0])$ that are unit modulo the assignment $x \mapsto 0011$ (left and center), and the conjunction BDD (right).

resent restrictions on feasible values with the combination of an interval (e.g. $[0000, 0010]$), and of a pattern (e.g., ???1), imposing the value of some of the bits. We propose here to use a Binary Decision Diagram (BDD) [Bry86] over the bits of y. BDDs have the advantage that they can encode the exact set of feasible values: these are exactly those values that give rise to a path to True in the BDD. This precision can also be a disadvantage if the complexity of constructing a BDDs is prohibitive. But, since in the MCSAT context, we are only considering unit constraints, with a single free bit-vector variable, we believe that BDDs are quite appropriate.

The BDD for each variable is initialized with the BDD constant True. Assume that we must satisfy the constraint $(y <_u x)$, on two bit-vector variables x and y of length 4. If we assign to x the value 0011, for example, the constraint becomes unit and gives rise to the first BDD of Fig. 1. Assume then another unit constraint $(y[0] = x[0])$ is processed that forces y to follow the pattern ???1, i.e., forcing y to be odd. The corresponding BDD, the second one of Fig. 1, is combined with the first one in a conjunction, producing the third one. Since this BDD encodes the singleton $\{0001\}$, the assignment $y \mapsto 0001$ can then be propagated on the trail.

Note that in case a BDD for y becomes a singleton, the unique assignment $y \mapsto v$ that can be "propagated" on the trail does not have an explicit justification and technically has to be placed on the trail as a decision. In contrast, the propagation mechanisms of [ZWR16] have explicit justifications, which can directly be used in conflict analysis: If the propagated assignment contributes to a conflict, having its explicit justification allows an immediate resolution step in the conflict analysis. Having this assignment as a decision (suggested by a singleton BDD $\{v\}$ for y) would stop the conflict analysis there; but what is learned from the conflict immediately entails $y \not\simeq v$, which would empty the BDD for y and trigger a new conflict.

Indeed, if at some point the BDD representing the feasible set for y becomes empty, then MCSAT will detect a conflict, explain it, and backjump. We describe this in the next section.

2.3 Explaining conflicts

When the set of feasible values for a bit-vector variable y becomes empty, then the trail contains a set of bit-vector constraints

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

with free variables $\vec{x} = x_1, \ldots, x_n$, and y, as well as a the assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$ forming a partial model $\mathcal{M}$. In addition, the conjunction of BDDs over the bits of y formed by the constraints in $\mathcal{A}$ under the partial model $\mathcal{M}$, is the empty BDD.

The trivial and the simplest way to explain this conflict is to state it directly, i.e.

$$A_1 \wedge \cdots \wedge A_m \Rightarrow (x_1 \not\simeq v_1) \vee \cdots \vee (x_n \not\simeq v_n) \ .$$

This conflict allows MCSAT to undo the latest decision and resume the search with some other choice.

For many theories other than bit-vectors, ruling out one model, as above, would not lead to a terminating calculus if the domain of values is infinite: infinitely many attempts would be needed to exhaust the possible values. For the theory of bit-vectors, since the underlying domain is finite, ruling out models one at a time does lead to a sound, complete and terminating calculus. But this is of course impractical. So, as with theories with infinite value domains (e.g, arithmetic [JBdM13]), we should try to learn a general explanation of the conflict that allows us to rule out many more values than one. To do so, we need to identify what it is, about the constraints $A_1, \ldots, A_m$, and the partial model $\mathcal{M}$, that makes the feasible space for y empty. We will express the explanation as a conflict clause of the form $\mathcal{A}_c \Rightarrow I$, where $\mathcal{A}_c \subseteq \mathcal{A}$ is the core of the conflict. We call the clause I above an *interpolating clause* as defined below.

Definition 1 (Interpolating clause)**.** *Let $\mathcal{A}(\vec{x}, y)$ be a set of bit-vector constraints, and let $\mathcal{M}$ be a model for $\vec{x}$. We call a clause I an* interpolating clause *for $\mathcal{A}$ at $\mathcal{M}$, if*

1. *$\mathcal{A} \Rightarrow I$ is a valid in bit-vectors,*
2. *I only contains variables $\vec{x}$, and*
3. *I evaluates to false in $\mathcal{M}$.*

In other words, the interpolating clause interpolates between the constraints $\mathcal{A}$, and the current assignment in the trail.

Apart for the trivial explanation, for bit-vectors, there is a procedure that can provide both the core $\mathcal{A}_c$ and the interpolating clause I: bit-blasting with a SAT solver. We can bit-blast the constraints in $A_1, \ldots, A_m$ into a SAT problem and solve the resulting SAT problem *under assumptions* that all constraints A_i are true, and that each bit of $x_1, \ldots, x_n$ is true or false as indicated by the values $v_1, \ldots, v_n$. Since the original problem is not satisfiable, the SAT solver will return a *core*, indicating which bits of $v_1, \ldots, v_n$ (and which constraints $A_1, \ldots, A_m$) contributed to the unsatisfiability. From this core we can then trivially construct an explanation. The smaller the core, the more values we can rule out with the explanation.

Bit-blasting always applies, and can be used as the default procedure to produce a conflict clause. Note that we only need to bit-blast the set of constraints that were unit in y. This can be significantly smaller than the whole set of bit-vectors constraints on the trail.

Although always applicable, by expanding the constraints to individual bits, the bit-blasting approach is not too appealing as a way of producing explanation. In some cases, when the constraints in $\mathcal{A}$ live in a suitable fragment of $\mathcal{BV}$, there are better ways to produce explanations. By better we mean that the cost of generating the explanation may be cheaper, and/or the explanation itself may rule out more values.

Consider, for example, the following constraints

$$\mathcal{A} = \{x_1 \not\simeq x_2, \ \ x_1 \simeq y, \ \ x_2 \simeq y\} \ .$$

From a model with assignments $x_1 \mapsto 1001, x_2 \mapsto 0101$, the bit-blasting approach might produce the explanation

$$(x_1 \simeq y \wedge x_2 \simeq y) \Rightarrow (x_1[3] \Rightarrow x_2[3]) \ .$$

After revising the model, on the second attempt we might similarly learn that $(x_2[3] \Rightarrow x_1[3])$. Continuing in this way, we might need 8 iterations in total to learn enough information to represent the high-level explanation for the conflict:

$$(x_1 \simeq y \wedge x_2 \simeq y) \Rightarrow x_1 \simeq x_2 \ .$$

A procedure that can produce $(x_1 \simeq x_2)$ directly would be much more desirable.

Following the MCSAT approach, we propose the following path to producing explanations. First, use BDDs to isolate the core $\mathcal{A}_c \subseteq \mathcal{A}$ of the constraints in conflict (e.g., by relying on the quick-explain mechanism [Jun01]), and then decide which explanation procedure to apply depending on the fragment of $\mathcal{BV}$ where the core $\mathcal{A}_c$ lives. The smaller the $\mathcal{A}_c$, the higher the chances are that it lives in an isolated fragment of $\mathcal{BV}$.

In the next section, we present such a procedure for the very simple fragment of $\mathcal{BV}$ made of equality, extraction and concatenation.

3 Conflict explanation for the core fragment of $\mathcal{BV}$

In this section we consider the core fragment of $\mathcal{BV}$, consisting of equalities and disequalities between bit-vector terms made of variables, constants, extraction and concatenation:

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

where x ranges over the bit-vector variables, each of which has an implicit length, c ranges over the bit-vector constants, and h and l range over integers. Terms are assumed to be well-formed with respect to the bit-vector lengths, i.e. in $t \simeq u$, t and u are assumed to be of the same length, and in $t[h{:}l]$, h and l are assumed to be valid bit-vector indices for t, with $l \leq h$. We abbreviate $t[i{:}i]$ as $t[i]$.

In the rest of the section we propose a procedure that produces a conflict clause, also called explanation, when given a conflict, as defined in the previous section. We therefore have

- a set of variables $X = \{x_1, \ldots, x_n, y\}$;
- a partial assignment $\mathcal{M}$ mapping the variables x_i to bit vectors values v_i;
- a set of equalities $E = \{a_i \simeq b_i\}_{i \in \mathfrak{E}}$ with variables in X, and
- a set of disequalities $D = \{a_i \not\simeq b_i\}_{i \in \mathfrak{D}}$ with variables in X.

In addition, we know that there is no model of $\mathcal{A} = \{E, D\}$ extending $\mathcal{M}$ with a value for y (it is a conflict). Furthermore, we can assume that this conflict is a core, i.e. that no constraint can be removed from $\mathcal{A}$ while remaining a conflict.

3.1 Slicing

In order to proceed with explaining the conflict, we first reduce the conflicting core E, D into a normalized form of equalities and disequalities between *normal* terms. Normal terms are either of the form $z[h{:}l]$, called *slices* (of a variable z), or constants. Moreover, we make sure that the slices of z that appear in the normalized problem are non-overlapping. The transformed problem has the same free variables as the original problem, and is logically equivalent to it. This transformation is done by computing the *coarsest-base slicing*, following the well known procedures such as [CMR97, BS09].

Consider for instance the following problem on variables y, of length 6, and x_1, of length 8.

$$E = \{\ x_1[3{:}0] \simeq x_1[7{:}4]\ ,\ y[5{:}2] \simeq y[3{:}0]\ \},\ D = \{\ y[3{:}0] \not\simeq x_1[7{:}4]\ \}\ .$$

Note that the above slices of y are overlapping. By careful isolating the appropriate indexing intervals, this problem can be transformed into

$$E_s = \{\ x_1[3{:}2] \simeq x_1[7{:}6]\ ,\ x_1[1{:}0] \simeq x_1[5{:}4])\ ,\ y[5{:}4] \simeq y[3{:}2]\ ,\ y[3{:}2] \simeq y[1{:}0]\ \}\ ,$$
$$D_s = \{\ (y[3{:}2] \not\simeq x_1[7{:}6]) \vee (y[1{:}0] \not\simeq x_1[5{:}4])\ \}.$$

After slicing, the set of equalities E remains a set of equalities (over normal terms), while the set of disequalities D becomes a set of clauses D_s containing disequalities over normal terms.

3.2 Generation of explanations

After slicing, we have a set of equalities E_s, and a set of clauses D_s, all of whose literals are disequalities. All terms involved have their variables among $x_1, \ldots, x_n, y$. The model $\mathcal{M}$, made of the trail assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$, allows the evaluation of any normal term that is a constant or the slice of a variable x_i. Moreover, since the slices are non-overlapping, the transformed problem lies within the theory of equality (seeing slices as independent terms), with a twist being the cardinality constraints on the bit-vector domain – having to satisfy more disequalities than cardinality allows may be the reason of the conflict.

We first proceed to analyze the conflict with equality reasoning. We construct the E-graph $\mathcal{G}$ from E_s [DNS05], also taking into account the values assigned by model $\mathcal{M}$, as presented in Algorithm 1. The goal is to make sure that in each component of the E-graph, the terms that $\mathcal{M}$ can evaluate all evaluate to the same value, otherwise a conflict is found and the explanation can be produced. The predicate denoted $\mathsf{isdef}_{\mathcal{M}}(t)$ means that term t can be evaluated in $\mathcal{M}$ (i.e. it is not a slice of y), in which case $\mathsf{eval}_{\mathcal{M}}(t)$ denotes its value. We maintain the invariant that if a component contains a term that evaluates to value v in $\mathcal{M}$, then the representative of that component evaluates to value v in $\mathcal{M}$. To maintain this invariant, we ensure that the following property holds whenever we merge two components: if both representatives have values, these values are identical, and if only one of the two representatives has a value, then this representative becomes the representative of the merged component.

Algorithm 1 E-graph with value management

```
1:  function E_GRAPH(E_s, M)
2:      INITIALIZE(G)
3:      for t_1 ≃ t_2 ∈ E_s do
4:          t'_1 ← REP(t_1, G)                    ▷ get representative for t_1's component
5:          t'_2 ← REP(t_2, G)                    ▷ get representative for t_2's component
6:          if isdef_M(t'_1) and isdef_M(t'_2) and eval_M(t'_1) ≠ eval_M(t'_2) then
7:              raise_conflict(E ⇒ t'_1 ≃ t'_2)                    ▷ D must be empty
8:          t_3 ← SELECT(t'_1, t'_2)              ▷ select representative for merged component
9:          G ← MERGE(t_1, t_2, t_3, G)           ▷ merge the components with new representative t_3
10:     return G
```

The E-graph construction can detect and explain conflicts between the equalities in E and the current assignment, as in the following example.

Example 1. *Let r_1, r_2, r_3 be bit ranges of the same length. Let E be such that $E_s = \{x_1[r_1] \simeq y[r_3],\ x_2[r_2] \simeq y[r_3]\}$, and let D be empty. Consider the model $\mathcal{M}$ that maps $x_1 \mapsto 0\ldots0, x_2 \mapsto 1\ldots1$. Then, E_GRAPH$(E_s, \mathcal{M})$ raises the conflict $E \Rightarrow x_1[r_1] \simeq x_2[r_2]$.*

If the E-graph construction does not raise a conflict, then $\mathcal{M}$ is compatible with the equalities entailed by E_s and provides a value to each E-graph component that has at least one constant

term or a slice of some x_i. Moreover, giving an arbitrary value to every other component (those entirely made of slices of y) yields a value v for y such that $\mathcal{M}, y \mapsto v$ satisfies E_s, and therefore E. But since we assumed that $\mathcal{M}$ cannot be extended into a model of $E \wedge D$, any assignment of values to components made of slices of y will be inconsistent with the clauses in D_s. In other words, the conflict involves the disequalities.

In order to analyze the (disequality) conflict further, we note that any clause $C \in D_s$ can be decomposed into the following components

$$C_{E_s} \vee C_{\mathcal{M}} \vee C_{\mathsf{interface}} \vee C_{\mathsf{free}} \ ,$$

where

- C_{E_s} contains disequalities $t_1 \not\simeq t_2$ such that t_1 and t_2 have the same E-graph representative: in other words, this disequality is necessarily false because of the equalities in E_s;
- $C_{\mathcal{M}}$ is made of disequalities $t_1 \not\simeq t_2$ such that t_1 and t_2 have distinct E-graph representatives, but these evaluate in $\mathcal{M}$ to the same value: in other words, this disequality is necessarily false because of the values assigned by $\mathcal{M}$;
- $C_{\mathsf{interface}}$ is made of disequalities $t_1 \not\simeq t_2$ such that t_1 and t_2 have distinct E-graph representatives, namely t_1' and t_2', such that t_1' evaluates in $\mathcal{M}$ to a value, but t_2' does not: it is necessarily a slice of y, and we can still satisfy $t_1 \not\simeq t_2$ by picking a good value for y; we say t_1' is *interfaced with a slice of* y, and call it *interface term*;
- C_{free} is made of disequalities $t_1 \not\simeq t_2$ such that t_1 and t_2 have distinct E-graph representatives, neither of which evaluates in $\mathcal{M}$: they are both slices of y, and we can still satisfy $t_1 \not\simeq t_2$ by picking a good value for y.

As already mentioned, E_s is satisfied for any value for y, and it is the conjunction of the clauses in D_s that empties the space of possible values. This can happen when (i) one of the clauses in D_s necessarily evaluates to false, in other words when $C_{\mathsf{interface}}$ and C_{free} are both empty. This can also happen if (ii) there is in each clause in D_s a disequality in $C_{\mathsf{interface}}$ or C_{free} that we can still satisfy, but whichever disequality we choose to satisfy in each clause, there are not enough values to satisfy the conjunction of the chosen disequalities (given the values that $\mathcal{M}$ is already using). In either case, we produce the conflict clause with Algorithm 2, scanning through D_s.

Algorithm 2 Disequality conflict

function DIS_CONFLICT($D_s, \mathcal{M}, \mathcal{G}$)
  $S \leftarrow \emptyset$ ▷ where we collect interface terms
  $C_0 \leftarrow \emptyset$ ▷ where we collect the disequalities that evaluate to false
  for $C \in D_s$ **do**
    $C_{\mathcal{M}}^{\mathsf{rep}} \leftarrow \bigvee\{\text{REP}(t_1, \mathcal{G}) \not\simeq \text{REP}(t_2, \mathcal{G}) \mid (t_1 \not\simeq t_2) \in C_{\mathcal{M}}\}$
    if IS_EMPTY($C_{\mathsf{interface}}$) and IS_EMPTY(C_{free}) **then**
      $\mathsf{raise\_conflict}(E \wedge D \Rightarrow C_{\mathcal{M}}^{\mathsf{rep}})$
    else
      $C_0 \leftarrow C_0 \vee C_{\mathcal{M}}^{\mathsf{rep}}$ ▷ we collect the disequalities made false in the model
      for $t_1 \not\simeq t_2 \in C_{\mathsf{interface}}$ with $\mathsf{isdef}_{\mathcal{M}}(\text{REP}(t_1, \mathcal{G}))$ **do**
        $S \leftarrow S \cup \{\text{REP}(t_1, \mathcal{G})\}$ ▷ we collect the interface term
  $C_{\neq} \leftarrow \bigvee\{t_1 \simeq t_2 \mid \mathsf{eval}_{\mathcal{M}}(t_1) \neq \mathsf{eval}_{\mathcal{M}}(t_2),\ t_1, t_2 \in S\}$
  $C_{=} \leftarrow \bigvee\{t_1 \not\simeq t_2 \mid \mathsf{eval}_{\mathcal{M}}(t_1) = \mathsf{eval}_{\mathcal{M}}(t_2),\ t_1 \neq t_2,\ t_1, t_2 \in S\}$
  return $E \wedge D \Rightarrow C_0 \vee C_{\neq} \vee C_{=}$

Let us describe how the algorithm behaves on a specific example of conflict of type (i):

Example 2. *Let r_1 and r_2 be bit ranges of the same length, let r_3, r_4, r_5 be bit ranges of the same length. Let E be such that E_s contains*

$$\{\ x_1[r_1] \simeq y[r_1]\ ,\ x_2[r_2] \simeq y[r_2]\ ,\ y[r_3] \simeq y[r_5]\ ,\ y[r_4] \simeq y[r_5]\ \}.$$

Let D be such that D_s is the singleton

$$\{\ (y[r_1] \not\simeq y[r_2] \vee y[r_3] \not\simeq y[r_4])\ \}.$$

And let $\mathcal{M}$ map x_1 and x_2 to $0 \dots 0$.

Assume the E-graph has selected $y[r_5]$ as the representative for the component

$$\{\ y[r_3], y[r_4], y[r_5]\ \}.$$

Then DIS_CONFLICT*$(D_s, \mathcal{M}, \mathcal{G})$ treats the unique clause C of D_s and:*

- *For the first disequality, the representatives of $y[r_1]$ and $y[r_2]$, namely $x_1[r_1]$ and $x_2[r_2]$, both evaluate to $0 \dots 0$, so the disequality belongs to $C_{\mathcal{M}}$;*
- *And for the second disequality, the representatives of $y[r_3]$ and $y[r_4]$ are the same, namely $y[r_5]$, so this second disequality (which does not evaluate in $\mathcal{M}$) belongs to C_{E_s}.*

As $C_{\mathsf{interface}}$ and C_{free} are empty, Algorithm 2 raises the conflict $E \wedge D \Rightarrow x_1[r_1] \not\simeq x_2[r_2]$.

More generally in a conflict of type (i), it will always be the case that D_s is a singleton: the clause of D_s that evaluates to false, together with E_s and $\mathcal{M}$, empties the range of possible values for y, so having assumed that E, D is a core, it means that D_s is a singleton that only contains that clause.

When analyzing a conflict of type (ii), the equalities and disequalities that hold in $\mathcal{M}$ between the interface terms make the slices of y require more values than we have. So our conflict clause includes (the negation of) all such equalities and disequalities. An example can be given as follows:

Example 3. *Let E be empty, and let D be such that D_s is*

$$\{\ (\ x_2[0] \not\simeq x_2[1] \vee y[0] \not\simeq y[1]\ )\ ,\ x_1[0] \not\simeq y[0]\ ,\ x_1[1] \not\simeq y[1]\ \}.$$

And let $\mathcal{M}$ map x_1 and x_2 to 00.

As E is empty, the E-graph components are simply the same as the terms of the problem.

We detail below the behavior of DIS_CONFLICT*$(D_s, \mathcal{M}, \mathcal{G})$:*

- *When treating the first clause, call it C, the first disequality is in $C_{\mathcal{M}}$, as the two sides belong to different components but evaluate to the same value; therefore C_0 becomes $\{\ x_2[0] \not\simeq x_2[1]\ \}$; the second disequality features two slices of y and therefore is in C_{free}, so the clause is potentially satisfiable and we move on to the next clause.*
- *The second clause is a unit one, and its only disequality cannot be evaluated, as its left-hand side can but its right-hand side cannot; so S becomes $\{\ x_1[0]\ \}$; the clause is potentially satisfiable so we move on to the next clause.*
- *Again, the third clause is a unit clause whose only disequality cannot be evaluated, as its left-hand side can but its right-hand side cannot; so S becomes $\{\ x_1[0]\ ,\ x_1[1]\ \}$; the clause is potentially satisfiable and, since this was the last clause of D_s, we conclude that the unsatisfiability of the original problem is a cardinality issue.*

Indeed, $y[0]$ should be different from 0 because of the second clause, $y[1]$ should also be different from 0 because of the third clause, but $y[0]$ and $y[1]$ should still be different from each other because of the first clause, and we hit the fact that we only have two values for bits.

Algorithm 2 produces the conflict clause

$$D \Rightarrow (\ x_2[0] \not\simeq x_2[1] \vee x_1[0] \not\simeq x_1[1]\ ).$$

Indeed, $x_2[0] \not\simeq x_2[1]$ is necessary because, if it were true in $\mathcal{M}$, we would not have to satisfy $y[0] \not\simeq y[1]$ and therefore $y \leftarrow 11$ would work. And $x_1[0] \not\simeq x_1[1]$ is necessary because, if it were true in $\mathcal{M}$, say with $x_1 \leftarrow 01$ (resp. $x_1 \leftarrow 10$), then $y \leftarrow 11$ (resp. $y \leftarrow 00$) would work.

Correctness of the method relies on the following lemma:

Lemma 1 (The produced clauses are interpolating clauses)**.**

- *If Algorithm 1 reaches line 7,* $t'_1 \simeq t'_2$ *is an interpolating clause for* $E \wedge D$ *at* $\mathcal{M}$*.*
- *If Algorithm 2 reaches line 7,* $C^{\mathsf{rep}}_{\mathcal{M}}$ *is an interpolating clause for* $E \wedge D$ *at* $\mathcal{M}$*.*
- *If Algorithm 2 reaches line 14,* $C_0 \vee C_{\neq} \vee C_{=}$ *is an interpolating clause for* $E \wedge D$ *at* $\mathcal{M}$*.*

4 Conclusion

In this report on our work in progress, we have presented two main ideas for the treatment of $\mathcal{BV}$ in MCSAT, complementing the approach proposed in [ZWR16].

By using BDDs, the search mechanism of MCSAT can be generic, while specific mechanisms for conflict explanation can be chosen depending on the constraints involved in the conflict. Identifying the core of the constraints that contribute to the conflict can be done with BDDs, and increases the chances that a dedicated mechanism can be used for explanations (the default mechanism being bit-blasting). BDDs also offer a propagation mechanism that differs from those in [ZWR16], in that the justification of a propagated assignment is not computed before the assignment is ruled out by a conflict. Computing the core of the conflict at that point can be seen as recovering the justification of the propagation. The propagation mechanisms of [ZWR16] could actually cohabit with our BDD-based mechanisms, whenever the benefits that they give for conflict resolution outweigh the cost of detecting their applicability. Experimentation is needed to compare this cost to the cost of identifying conflict cores through BDDs.

In case the conflicting constraints live in the core fragment of $\mathcal{BV}$, we proposed an explanation mechanism based on coarsest-base slicing and equality reasoning, which we tuned for the production of explanations. Compared to previous work on coarsest-base slicing, such as [BS09], we only apply the transformation on the core constraints of a particular conflict, rather than the whole problem. This should in general make the slices coarser, which we expect to positively impact efficiency.

We are currently working on dedicated explanation mechanisms for bigger or different fragments of $\mathcal{BV}$; for instance the mechanisms studied in [JW16] (outside of the context of MCSAT) should lead to an explanation mechanism for bit-vector arithmetic without multiplication. In general we intend to rely, if possible, on equality reasoning, as equality is one of the aspects of bit-vectors that is lost by bit-blasting but that can be useful in learned lemmas.

Our work in progress of course involves the implementation of the ideas presented in this report. These are mostly based on the MCSAT principle that handling constraints that are unit in a bit-vector variable is in practice easier than handling multivariate constraints. This principle probably applies less to benchmarks whose constraints feature complex re-uses of a single variable, and we anticipate that our approach will apply to software verification problem better than to hardware verification problems.

Future work includes relating our approach to the very recent report by Chihani, Bobot, and Bardin [CBB17], which aims at lifting the CDCL mechanisms to the word level of bit-vector reasoning, and therefore seems very close to MCSAT.

Future work also includes exploring the integration of the proposed MCSAT treatment of bit-vectors with other components of SMT-solvers, whether in the context of MCSAT or in different architectures. An approach for this is the recent framework of *Conflict-Driven Satisfiability* (CDSAT) [BGLS17], which precisely aims at organizing, along the MCSAT mechanisms, the collaboration between generic theory modules.

Acknowledgments The authors thank Aleksandar Zeljić for fruitful discussions. The research presented in this paper has been supported in part by NSF grant 1528153. The views

and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF or the U.S. Government.

References

[BGLS17] 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 *LNCS*. Springer-Verlag, 2017.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[BS09] 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.

[CBB17] 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.

[CMR97] 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.

[dMJ13] L. M. de Moura and D. Jovanović. 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.

[DNS05] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM (JACM)*, 52(3):365–473, 2005.

[JBdM13] 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.

[Jov17] 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.

[Jun01] U. Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, 2001.

[JW16] 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.

[MLM09] J. Marques Silva, I. Lynce, and S. Malik. Conflict-driven clause learning SAT solvers. In A. Biere, M. Heule, H. V. Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press, 2009.

[ZWR16] 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 (SAT'16)*, volume 9710 of *LNCS*, pages 249–266. Springer-Verlag, 2016.