Interpolating bit-vector arithmetic constraints in MCSAT

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

Computer Science Laboratory, SRI International, USA

Abstract

We present an interpolation mechanism for a fragment of bit-vector arithmetic. Given a conjunction of constraints under one existential quantifier, and given an interpretation of its free variables that falsifies it, we produce a quantifier-free interpolant of the constraints and of the interpretation: the interpolant is implied by the constraints and it is still falsified by the interpretation. The interpolant explains why the interpretation does not satisfy the constraints. This can be used in an implementation of model-constructing satisfiability for bit-vectors, reasoning about bit-vector arithmetic at the word level rather than at the bit level.

1 Introduction

The interpolation mechanism we present here targets formulae of the form

$$\mathcal{A}(\vec{x}) = \exists y(C_1 \wedge \cdots \wedge C_m) \ ,$$

with free variables $\vec{x} = x_1, \ldots, x_n$, and with $C_1, \ldots, C_m$ being literals of a fragment of bit-vector arithmetic, which we shall define below. The interpolation mechanism applies in presence of a model Γ made of assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$, and such that the evaluation of $\mathcal{A}$ in model Γ, denoted $[\![\mathcal{A}]\!]_\Gamma$, is false.

This situation typically arises when trying to solve an SMT-problem in the theory of bit-vectors via the MCSAT approach [dMJ13]: Given some constraints to satisfy, assignments of bit-vector values to bit-vector variables are successively guessed with the invariant that none of the constraints evaluates to false given the assignments that have been made so far. If the invariant can be maintained until all variables are assigned, then these assignments constitute a model of the constraints, which in that case are satisfiable. On the contrary, if at one point the invariant cannot be maintained, it means that, for a certain bit-vector variable y that seeks a bit-vector value, every possible value makes one of the constraints evaluate to false. Technically, this means that, for a subset $\{C_1, \ldots, C_m\}$ of the constraints, the assignments $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$ made so far result in the existential formula $\exists y(C_1 \wedge \cdots \wedge C_m)$, a.k.a. $\mathcal{A}$, evaluating to false. This is the situation described above, and henceforth called a *conflict*, with *conflict variable* y and *conflict literals* $C_1 \wedge \cdots \wedge C_m$.

When such a conflict happens, backtracking is needed so that other values can be picked for $x_1, \ldots, x_n$, with the hope that y can then be assigned a value without breaking the invariant. Picking the same values again must of course be avoided (lest the algorithm loops forever), but even different values could result in a conflict for the same "reason" the original values resulted in a conflict. They too could be avoided, thus speeding up the search for a model satisfying the constraints. Such a "reason" *explains* why the choice of values for $x_1, \ldots, x_n$ necessarily makes $\mathcal{A}$ evaluate to false. Technically, the explanation produces new constraints on $x_1, \ldots, x_n$ (without mentioning y) that future assignments will have to satisfy. We express the explanation as a *conflict clause* of the form $\mathcal{C} \Rightarrow I$, where $\mathcal{C} \subseteq \{C_1, \ldots, C_m\}$ is the core of the conflict. We call the clause I above an *interpolating clause* as defined below.

Definition 1 (Interpolating clause)**.** *Let $\mathcal{C}(\vec{x}, y)$ be a set of bit-vector constraints, and let Γ be a model for $\vec{x}$. We call a clause I an* interpolating clause *for $\mathcal{C}$ at Γ, if (i) $\mathcal{C} \Rightarrow I$ is valid in the bit-vector theory, (ii) I only contains variables $\vec{x}$, and (iii) I evaluates to false in Γ.*

Conditions (i) and (ii) are equivalent to the fact that $(\exists y \mathcal{C}) \Rightarrow I$ is valid, so we also say that I is an interpolant for $\exists y \mathcal{C}$ at Γ. If $\mathcal{C}$ is the whole set $\{C_1, \ldots, C_m\}$, I is an interpolant for $\mathcal{A}$ at Γ. In MCSAT, the interpolating clause interpolates between the core constraints $\mathcal{C}$, and the current assignment in the trail. The application of MCSAT to the bit-vector theory has been studied in [ZWR16], as well as in [GLJ17]. The latter puts forward the idea of basing conflict analysis on interpolation mechanisms that are dedicated to specific fragments of the bit-vector theory. The present work is an instance of this approach. In [JC16], quantifier elimination for linear bit-vector arithmetic is investigated. Quantifier elimination, when given a formula $\mathcal{A}$ as above, produces a quantifier-free formula F that is *equivalent* to $\mathcal{A}$, whereas an interpolating clause I is only *implied* by $\mathcal{A}$, and dependent on the model Γ that drives its construction, which we describe below. Connections with the procedures from [JC16] is left for future work.

Section 2 presents the preliminaries used for the interpolation mechanism, as well as a bird's eye view of it. Sections 3 and 4 present the two main stages of the interpolation algorithm. Section 5 sketches a generalisation of the interpolation mechanism that can treat bit-vector arithmetic constraints augmented with lower bits extraction operators.

2 Preliminaries and bird's eye view of interpolation

In this paper it is useful to keep in mind that bit-vector arithmetic for bit-width w coincides with arithmetic modulo 2^w. There is an isomorphism between the domain of 2^w bit-vector values and $\mathbb{Z}/2^w\mathbb{Z}$ with addition and multiplication being associative and commutative operators.

The fragment of bit-vector arithmetic that the interpolation mechanism applies to is *linear*, in that multiplication is only allowed with constants, and uses the following grammar:

$$\begin{array}{lllll} \text{Constraints (literals)} & C & ::= & a \mid \neg a \\ \text{Atoms} & a & ::= & t_1^w \leq^s t_2^w \mid t_1^w \leq^u t_2^w \mid t_1^w \simeq t_2^w \\ \text{Terms}^w & t^w & ::= & c_0^w + \Sigma_{i=1}^n c_i^w \cdot x_i^w \end{array}$$

where $\leq^s$ and $\leq^u$ respectively denote signed and unsigned comparison, w is a bit-width, $(x_i^w)_{i=1}^n$ are bit-vector variables of bit-width w, and $(c_i^w)_{i=0}^n$ are bit-vector constants of bit-width w, called *coefficients*. In what follows our notations will drop the bit-width superscript whenever there is no ambiguity. The strict comparison literals $t_1 <^s t_2$ and $t_1 <^u t_2$ and the disequality literal $t_1 \not\simeq t_2$ can be seen as abbreviating $\neg(t_2 \leq^s t_1)$, $\neg(t_2 \leq^u t_1)$ and $\neg(t_1 \simeq t_2)$, respectively. Without loss of generality we will assume that in a term $c_0 + \Sigma_{i=1}^n c_i \cdot x_i$, the variables are pairwise distinct, and the only constant that may be equal to 0 is c_0. Terms of bit-width w are closed under bit-vector addition, subtraction, and multiplication by a constant. Given a bit-vector variable x and a term t of bit-width w, we write $c_x(t)$ for the coefficient of x in t. With the above convention, $c_x(t)$ is 0 if and only if x does not appear in t.

The interpolation mechansim we present applies to formulae of the form $\exists y(C_1 \wedge \cdots \wedge C_m)$ where y is a bit-vector variable of bit-width w. The interpolant for that formula at model Γ that our mechanism produces has the form $(\overline{E_1} \vee \cdots \vee \overline{E_p})$, or equivalently $E_1 \wedge \cdots \wedge E_p \Rightarrow \bot$, where E_1,..., E_p are constraints from the grammar not mentioning y.

We can assume without loss of generality that all terms in the constraints $(C_i)_{i=1}^n$ have bit-width w; indeed if one of the constraints, say C_1, had a different bit-width, it could not mention y, and therefore the formula is equivalent to $C_1 \wedge \exists y(C_2 \wedge \cdots \wedge C_m)$, for which $C_1 \Rightarrow I$ is an interpolant whenever I is an interpolant for $\exists y(C_2 \wedge \cdots \wedge C_m)$.

However our interpolation mechanism only applies when constraints $C_1 \wedge \cdots \wedge C_m$ satisfy a particular property with respect to y, described below.

Definition 2. *We say that a term t is* y-friendly *if $c_y(t) \in \{-1, 0, 1\}$, and that an atom $t_1 \leq^s t_2$, $t_1 \leq^u t_2$ or $t_1 \simeq t_2$ is* y-friendly *if t_1 and t_2 are y-friendly and $c_y(t_1) \cdot c_y(t_2) \in \{0, 1\}$.*[1]

Our interpolation mechanism applies to a formula $\mathcal{A}$ of the form $\exists y(C_1 \wedge \cdots \wedge C_m)$ where $C_1, \ldots, C_m$ are literals over y-friendly atoms, and $[\![\mathcal{A}]\!]_\Gamma$ is false. Below is its general structure:

- For each C_i, $1 \leq i \leq m$, we determine a *forbidden interval* $I_i = [l_i; u_i[$, where l_i and u_i are terms not mentioning y, such that $C_i \Leftrightarrow (y \notin I_i)$. Hence, formula $\mathcal{A}$ is intuitively equivalent to $\exists y(y \notin I_1 \wedge \cdots \wedge y \notin I_m)$, or equivalently $\exists y(y \notin \bigcup_{i=1}^m I_i)$, expressing the fact that $\bigcup_{i=1}^n I_i$ does not cover the full domain of all 2^w bit-vector values of bit-width w. The fact that $[\![\mathcal{A}]\!]_\Gamma$ is false means that $\bigcup_{i=1}^m [\![I_i]\!]_\Gamma$ does indeed cover the full domain.
- We produce a series of constraints $E_1, \ldots, E_p$ which, collectively, express the fact that $\bigcup_{i=1}^m I_i$ is a coverage of the full domain. The interpolant will be $E_1 \wedge \cdots \wedge E_p \Rightarrow \bot$: it is implied by $\mathcal{A}$, and evaluates to false in Γ.

3 Turning constraints into forbidden intervals

We now assume that we have a constraint C that we want to express as a forbidden interval for y, of bit-width w. The first thing we do is focus on only one kind of atom, namely $t_1 \leq^u t_2$. The two other kinds of atoms are reduced to equivalent atoms of the above kind, using the following rewrite rules:

$$\begin{array}{lcl} t_1 \leq^s t_2 & \rightsquigarrow & t_1 + 2^{w-1} \leq^u t_2 + 2^{w-1} \\ t_1 \simeq t_2 & \rightsquigarrow & t_1 - t_2 \leq^u 0 \end{array}$$

Note that for any variable x (which may or may not be y), the original atom is x-friendly if and only if the rewritten atom is.

Now we also rewrite the atom $t_1 \leq^u t_2$ so that the coefficients of y on each side are either 0 or 1. If it is not, then we apply the following rewrite rule:

$$t_1 \leq^u t_2 \rightsquigarrow -(t_2 + 1) \leq^u -(t_1 + 1)$$

We now produce an interval I_C for a constraint C with (normalised) atom $t_1 \leq^u t_2$. An interval takes the form $[l; u[$, where the lower bound l and upper bound u are terms, with l included and u excluded. The notion of interval used here is considered modulo 2^w, i.e. is "circular" over the domain of 2^w values, seen as $\mathbb{Z}/2^w\mathbb{Z}$. For instance an interval $[0b1111; 0b0001[$, with two constant bounds, is a perfectly valid interval containing two bit-vector values of width 4: $0b1111$ and $0b0000$. The semantics of an interval $[l; u[$ is ambiguous when the semantics of l coincides with that of u. By convention, we will fix it to be the full domain of 2^n values, rather than the empty domain, which we will denote $\emptyset$.

The rules for producing I_a or $I_{\neg a}$ for a normalised atom a are given in Figure 1, where c_1 and c_2 are terms that do not mention y. Depending on whether the coefficients of y in t_1 and t_2 are 0 or 1, there are four cases. The first three cases are actually split by a condition that identifies when the natural lower bound and upper bound would coincide, in which case the produced interval is either empty or full. The last two cases concern the situation where y does not appear in the constraint. As explained above for constraints with a different bit-width than w, constraints without y could also be taken out of formula $\mathcal{A}$.

The interpolation mechanism looks at which line of the table applies, so that the condition evaluates to true in the model Γ, and outputs the corresponding interval I_C. The intuition is that, given the value assignments in Γ and the constraint C to satisfy, y cannot be in I_C.

[1] which is equivalent to $c_y(t_1) \cdot c_y(t_2)$ not being -1, *except* when the bit-width is 1, as then $1 = -1$.

Normalised atom a	**Forbidden interval that** a **(resp.** $\neg a$**) specifies for** y			
	I_a	$I_{\neg a}$	**Condition**	
$c_1 + y \leq^u c_2 + y$	$[-c_2; -c_1[$	$[-c_1; -c_2[$	$c_1 \not\simeq c_2$	1
	$\emptyset$	$[0; 0[$	$c_1 \simeq c_2$	2
$c_1 \leq^u c_2 + y$	$[-c_2; c_1 - c_2[$	$[c_1 - c_2; -c_2[$	$c_1 \not\simeq 0$	3
	$\emptyset$	$[0; 0[$	$c_1 \simeq 0$	4
$c_1 + y \leq^u c_2$	$[c_2 - c_1 + 1; -c_1[$	$[-c_1; c_2 - c_1 + 1[$	$c_2 \not\simeq -1$	5
	$\emptyset$	$[0; 0[$	$c_2 \simeq -1$	6
$c_1 \leq^u c_2$	$[0; 0[$	$\emptyset$	$c_2 <^u c_1$	7
	$\emptyset$	$[0; 0[$	$c_1 \leq^u c_2$	8

Figure 1: Creating the forbidden intervals

Example 1.

1.1 *Assume* C_1 *is literal* $\neg(x_1 \leq^u y)$ *and* $\Gamma = \{x_1 \mapsto 0b0000\}$. *Then line 4 of Figure 1 applies (because of* Γ*), and* I_{C_1} *is the full interval* $[0; 0[$ *with condition* $x_1 \simeq 0$.

1.2 *Assume* C_1 *is* $\neg(y \simeq x_1)$, C_2 *is* $(x_1 \leq^u x_3 + y)$, C_3 *is* $\neg(y - x_2 \leq^u x_3 + y)$, *and* $\Gamma = \{x_1 \mapsto 0b1100, x_2 \mapsto 0b1101, x_3 \mapsto 0b0000\}$. *Then by line 5,* $I_{C_1} = [x_1; x_1 + 1[$ *with trivial condition* $(0 \not\simeq -1)$, *by line 3,* $I_{C_2} = [-x_3; x_1 - x_3[$ *with condition* $(x_1 \not\simeq 0)$, *and by line 1,* $I_{C_3} = [x_2; -x_3[$ *with condition* $(-x_2 \not\simeq x_3)$.

The table is inspired by [JW16], Table 1. We leverage the approach for the purpose of building interpolants, so in our case the expressions c_1, c_2, etc are not constants, but can have variables (with values in model Γ). A rather cosmetic difference we make consists in working with intervals that exclude their upper bound, as this makes the theoretic and implemented treatment of those intervals simpler and more robust to the degenerate case of bit-width 1, where $1 = -1$. Another difference is that we take circular intervals, so that every constraint corresponds to exactly one interval; as a result, we do not need the case analyses expressed by the conditions of Table 1 in [JW16]. We do, however, make some new case analyses to detect when a constraint leads to an empty or full forbidden interval, since such intervals will be subject to a specific treatment when generating interpolants, as described in the next section.

4 Generating the interpolant

From constraints $C_1, \ldots, C_m$ with free variables $\vec{x}, y$ we now have forbidden intervals $I_{C_1}, \ldots, I_{C_m}$, more simply denoted $I_1, \ldots, I_m$, and conditions $c_1, \ldots, c_m$ that are satisfied by Γ. As said before, the fact that $[\![\mathcal{A}]\!]_\Gamma$ is false means that $\bigcup_{i=1}^m [\![I_i]\!]_\Gamma$ is the full domain. The property "$\bigcup_{i=1}^m I_i$ is the full domain" now needs to be expressed symbolically as a conjunction $E_1 \wedge \cdots \wedge E_p$ of constraints from the grammar, expressing some property about model Γ that left us no possible value to pick for y. In the context of MCSAT [dMJ13], we then backtrack over the construction of Γ to try and build a different model; while attempting the new model construction, we will avoid any model satisfying property $E_1 \wedge \cdots \wedge E_p$, in order not to fall into a conflict of the same nature as the one we just analysed. With different values for variables $\vec{x}$ than those specified by Γ, the intervals $\bigcup_{i=1}^m I_i$ could cover the full domain in many different ways. Here, we are trying to symbolically capture the way that Γ allows itmakes those intervals a full coverage.

First, imagine that one of the intervals, say I_{i_0}, is the full interval $[0; 0[$. Then we define $p = 1$ and E_1 is simply the condition c_{i_0}. The interpolant is therefore $c_{i_0} \Rightarrow \bot$.

Algorithm 1 Extracting a covering sequence of intervals

```
1:  function SEQ_EXTRACT({I_1, ..., I_m}, Γ)
2:      output ← ()                          ▷ output initialised with the empty sequence of intervals
3:      longest ← LONGEST({I_1, ..., I_m}, Γ)                    ▷ longest interval identified
4:      baseline ← longest.upper                  ▷ where to extend the coverage from
5:      while [[baseline]]_Γ ∉ [[longest]]_Γ do
6:          I ← FURTHEST_EXTEND(baseline, {I_1, ..., I_m}, Γ)
7:          output ← output, I                    ▷ adding I to the output sequence
8:          baseline ← I.upper                    ▷ updating the baseline for the next interval pick
9:      if [[baseline]]_Γ ∈ [[output.first]]_Γ then
10:         return output                         ▷ the circle is closed without the help of longest
11:     return output, longest                    ▷ longest is used to close the circle
```

Example 2.

2.1 In Example 1.1 where C_1 is literal $\neg(x_1 \leq^u y)$ and $\Gamma = \{x_1 \mapsto 0b0000\}$, the interpolant for $\exists y\ \neg(x_1 \leq^u y)$ at Γ is $(x_1 \simeq 0) \Rightarrow \bot$.

2.2 In contrast, Example 1.2 does not contain a full interval. The model Γ satisfies the three conditions $(0 \not\simeq -1)$, $(x_1 \not\simeq 0)$ and $(-x_2 \not\simeq x_3)$, and respectively evaluates the three intervals $I_1 = [x_1; x_1 + 1[$, $I_2 = [-x_3; x_1 - x_3[$, and $I_3 = [x_2; -x_3[$, as $[\![I_1]\!]_\Gamma = [0b1100; 0b1101[$, $[\![I_2]\!]_\Gamma = [0b0000; 0b1100[$, and $[\![I_3]\!]_\Gamma = [0b1101; 0b0000[$. Note how $\bigcup_{i=1}^{3} [\![I_i]\!]_\Gamma$ is the full domain.

In case none of the intervals are full (as in Example 2.2), a sequence $I_{\pi(1)}, \ldots, I_{\pi(q)}$ of intervals can be extracted from the set $\{I_1, \ldots, I_m\}$, such that the sequence $[\![I_{\pi(1)}]\!]_\Gamma, \ldots, [\![I_{\pi(q)}]\!]_\Gamma$ of *concrete* intervals leaves no "hole" between an interval of the sequence and the next, and goes round the full circle of domain $\mathbb{Z}/2^w\mathbb{Z}$. The sequence forms a circular chain of linking intervals, and the produced constraints $E_1, \ldots, E_p$ will express the fact that the intervals do link up with each other, e.g. the upper bound of a link belongs to the next link.

4.1 Generating the sequence of intervals

In practice we can identify, given Γ, such a covering sequence $I_{\pi(1)}, \ldots, I_{\pi(q)}$ by applying a coverage algorithm like Algorithm 1, where:

- LONGEST$(\{I_1, \ldots, I_m\}, \Gamma)$ returns an interval among $\{I_1, \ldots, I_m\}$ whose concrete version $[\![I]\!]_\Gamma$ has maximal length;
- I.upper denotes the upper bound of an interval I (remember it is *excluded* from I);
- FURTHEST_EXTEND$(a, \{I_1, \ldots, I_m\}, \Gamma)$ returns an interval I among $\{I_1, \ldots, I_m\}$ that furthest extends a according to Γ (technically, an interval I that $\leq^u$-maximises $[\![I.\mathsf{upper} - a]\!]_\Gamma$ among those intervals I such that $[\![a]\!]_\Gamma \in [\![I]\!]_\Gamma$).
- output.first denotes the first element of a sequence output;

We stop with the first interval I that closes the circle, in that its concrete upper bound $[\![I.\mathsf{upper}]\!]_\Gamma$ belongs to $[\![\mathsf{longest}]\!]_\Gamma$ (it may or may not close the circle without the help of $[\![\mathsf{longest}]\!]_\Gamma$, hence the final if...then...else). Note that $\bigcup_{i=1}^{m} [\![I_i]\!]_\Gamma$ is not the full domain if and only if one of the calls FURTHEST_EXTEND$(a, \{I_1, \ldots, I_m\}, \Gamma)$ fails.

Remark 1. *The reason why we identify an interval of maximal length is to obtain a* minimal *coverage of the full domain: otherwise the last interval added to the sequence could include some*

of the first ones; removing those from the sequence would still produce a covering sequence.[2]

4.2 Generating the constraints for the interpolant

From now on we assume without loss of generality that the covering sequence of intervals is $I_1, \ldots, I_q$. We now produce the symbolic constraints that express the property that the covering has no hole and is circular. We take $p = q$ and for each i between 1 and q, E_i expresses "the upper bound of I_i belongs to I_{i+1}".[3] A naive way to express it would be to take E_i to be $(l_{i+1} \leq^u u_i <^u u_{i+1})$. That would fail to capture the possibility that the intervals overflow.[4] Instead, we take E_i to be $(u_i - l_{i+1} <^u u_{i+1} - l_{i+1})$.[5]

Then the interpolant for $\mathcal{A}$ at Γ is $E_1 \wedge \cdots \wedge E_q \Rightarrow \bot$.

Example 3. *For Example 2.2, Algorithm 1 identifies* $[\![I_2]\!]_\Gamma$ *as the longest concrete interval, and produces the coverage sequence* I_1, I_3, I_2*, i.e.* $[x_1; x_1{+}1[$, $[x_2; -x_3[$, $[-x_3; x_1 - x_3[$*. We express the properties* $(x_1{+}1) \in I_3$, $(-x_3) \in I_2$, *and* $(x_1 - x_3) \in I_1$ *as the three constraints* $E_1 = ((x_1{+}1 - x_2 <^u -x_3 - x_2))$, $E_3 = (0 <^u x_1)$, *and* $E_2 = (-x_3 <^u 1)$*. The interpolant is*

$$((x_1{+}1 - x_2 <^u -x_3 - x_2)) \wedge (0 <^u x_1) \wedge (-x_3 <^u 1) \Rightarrow \bot.$$

5 Generalisation with lower bits extraction

In this section we generalise the approach to a bigger fragment of the bit-vector theory.

5.1 Preliminaries

The original goal was to be able to treat bit-vector comparison predicates $\leq^u, \leq^s, \simeq$ when the arguments t_1 and t_2 have different bit-widths w_1 and w_2: if for instance $w_1 < w_2$ then t_1 has to be cast into a bit-vector of width w_2 by using a *0-extension* or a *sign-extension*, so that it can be compared with t_2 on w_2 bits. With bit-vectors, the 0-extension of t_1 with m bits, denoted $0\text{-extend}_m(t_1)$, adds m zeros to the most significant bits of t_1, while the sign-extension of t_1 with m bits, denoted $\pm\text{-extend}_m(t_1)$, adds m replicas of t_1's sign bit in front of it. Mathematically, they correspond to two injections of $\mathbb{Z}/2^{w_1}\mathbb{Z}$ into $\mathbb{Z}/2^{w_1+m}\mathbb{Z}$: 0-extension corresponds to the identity function from $[0; 2^{w_1}[$ to $[0; 2^{w_1+m}[$, while sign-extension corresponds to the identity function from $[-2^{w_1-1}; 2^{w_1-1}[$ to $[-2^{w_1+m-1}; 2^{w_1+m-1}[$.

[2] The issue does not occur in MCSAT as currently implemented, where we have an extra piece of information, namely that the original constraints $C_1, \ldots, C_m$ form a *core* of the conflict: if one of them, say C_1, is removed, then $\exists y(C_2 \wedge \cdots \wedge C_m)$ evaluates to true in Γ. Given that each interval I_m exactly captures its constraint C_i, it means that none of the concrete intervals $[\![I_1]\!]_\Gamma, \ldots, [\![I_m]\!]_\Gamma$ is empty and all of them are needed, so that the sequence must be of length m and is just an ordering of the set of intervals. Moreover if one of the intervals is full, then it must be the only interval. Still, the algorithm above allows us to produce the ordering.

[3] $(i{+}1)$ is to be understood modulo q, i.e. $q + 1 = 1$, capturing the circularity property.

[4] A particular case could be made for the interval(s) that overflow(s), expressing the linking property differently, but that would actually give a particular role to the constant 0 in the circular domain $\mathbb{Z}/2^w\mathbb{Z}$. This would weaken the interpolant, in the sense that it would rule out fewer models that fail to satisfy $\mathcal{A}$ "for the same reason" Γ does. Indeed, imagine another model Γ' falsifying $\mathcal{A}$ and leading to concrete intervals $[\![I_1]\!]_{\Gamma'}, \ldots, [\![I_m]\!]_{\Gamma'}$ that only differ from $[\![I_1]\!]_\Gamma, \ldots, [\![I_m]\!]_\Gamma$ in that all bounds are shifted by a common constant. The interpolant that gives a special role to 0 may not rule out Γ', whereas the interpolant we produce does.

[5] It may look strange that, in that interpolant, none of the interval conditions c_i play a role. Indeed, the fact that Γ satisfies c_i is a condition under which $y \notin I_i$ is equivalent to C_i. However, the constraint $(u_{i-1} - l_i <^u u_i - l_i)$ that ends up in the interpolant supersedes condition c_i: it implies that $0 <^u u_i - l_i$ and therefore that $u_i \not\simeq l_i$; this is exactly what condition c_i expresses: the fact that the two bounds are not allowed to coincide. Except for the degenerate case in which one of $I_1, \ldots, I_m$ is the full interval, which we have treated separately, the intervals $\{I_1, \ldots, I_q\}$ can never be empty or full, should $E_1, \ldots, E_q$ be satisfied.

0-extensions and sign-extensions do not have very good properties with respect to addition and multiplication in arithmetic modulo, as for instance $0\text{-extend}_m(a+b)$ is in general different from $0\text{-extend}_m(a)+0\text{-extend}_m(b)$ (and similarly for $\pm\text{-extend}_m(a+b)$ and for multiplication).

What does have very nice properties with respect to arithmetic modulo is *lower bits extraction*: in this section we denote the extraction of the m lower bits of t as $t\langle m\rangle$,[6] where t has bit-width at least m. Indeed, $(a+b)\langle m\rangle \simeq a\langle m\rangle + b\langle m\rangle$ and $(a\cdot b)\langle m\rangle \simeq a\langle m\rangle \cdot b\langle m\rangle$.

In a satisfiability problem, 0-extensions and sign-extensions can be expressed in terms of lower bits extraction: For a term t of bit-width w, a term $0\text{-extend}_m(t)$ in the input of a satisfiability problem can be replaced by a variable x during pre-processing, with constraints $t \simeq x\langle m\rangle$ and $x <^u 2^w$ added to the problem; a term $\pm\text{-extend}_m(t)$ can be replaced by a variable x during pre-processing, with constraints $t \simeq x\langle m\rangle$ and $x + 2^{w-1} <^u 2^w$ added to the problem.

5.2 Adapting the grammar and the production of forbidden intervals

In order to treat lower bits extraction, we can use the distributivity laws that it satisfies over $+$ and $\cdot$, and extend the grammar from Section 2 by changing the grammar for terms to:

$$\text{Terms}^w \quad t^w \quad ::= \quad c_0^w + \Sigma_{i=1}^n c_i^w \cdot (x_i^{w_i}\langle w\rangle)$$

where $x_i^{w_i}$ ranges over bit-vector variables of width w_i and $w \leq w_i$.

Note that the grammar is still closed under addition and multiplication by a constant, and is now also closed under lower bits extraction. But it is now no longer true that, in the formula $\exists y(C_1 \wedge \cdots \wedge C_m)$ for which we seek an interpolant, all atoms have the same bit-width. Let w be the bit-width of y. Given any constraint C among $C_1, \ldots, C_m$, let w_C be the bit-width of C, with $w_C \leq w$. The generation of an interval I_C from constraint C occurs exactly as in Section 3, using Figure 1 but with $y\langle w_C\rangle$ now playing the role of y. Provided the condition in the relevant line of Figure 1 is met, $C \Leftrightarrow (y\langle w_C\rangle) \notin I_C$.

5.3 Adapting the generation of the interpolant

The generation of the interpolant, however, needs to be refined, as now the intervals $\{I_1, \ldots, I_m\}$ live in domains of different bit-widths and concern different lower bits extracts of y. Again, in case one of the intervals I_{i_0} is the full interval for its domain $\mathbb{Z}/2^{w_{i_0}}\mathbb{Z}$, then none of the other intervals are needed and we produce the interpolant as in Section 4. Otherwise, the intervals need to complement each other, despite the fact that they concern different bit-widths.

Otherwise, we adapt the procedure described in Sections 4.1 and 4.2 in order to produce the constraints $E_1, \ldots, E_q$ that the interpolant $E_1 \wedge \cdots \wedge E_q \Rightarrow \bot$ is made of.

First, we group the intervals into different *layers* characterised by their bit-widths: an interval that originates from a conflict literal C of bit-width w_C, has its two bounds of bit-width w_C and forbids values for $y\langle w_C\rangle$; it will henceforth be called a *w_C-interval*. We order the groups of intervals by decreasing bit-widths $w_1 > w_2 > \cdots > w_j$, as shown in Figure 2.

Then we adapt Algorithm 1 from Section 4.1 into an algorithm handling different bit-widths, namely Algorithm 2. The first difference (a minor one) concerns the algorithm's output: instead of producing a covering sequence of intervals, and then in a second pass turning that sequence into the interpolant's constraints (as described in Section 4.2), Algorithm 2 directly outputs the interpolant's constraints $E_1, \ldots, E_q$. Hence, `output` is a set of constraints. The second difference concerns the alorithm's input (besides the model Γ): while Algorithm 1 takes as input a set $\mathcal{S}$ of intervals $\{I_1, \ldots, I_m\}$, Algorithm 2 takes the sequence of sets $(\mathcal{S}_1, \ldots, \mathcal{S}_j)$ of intervals grouped

[6] The more traditional (but longer) notation for it is usually $t[(m-1):0]$ or $t[0:(m-1)]$.

Bit-width	w_1	$>$	w_2	$> \cdots >$	w_j
Interval layer	w_1-intervals $\mathcal{S}_1 = \{I_{1.1}, I_{1.2}, \ldots\}$		w_2-intervals $\mathcal{S}_2 = \{I_{2.1}, I_{2.2}, \ldots\}$	$\ldots$ $\ldots$	w_j-intervals $\mathcal{S}_j = \{I_{j.1}, I_{j.2}, \ldots\}$
Forbidding values for	$y\langle w_1\rangle$		$y\langle w_2\rangle$	$\ldots$	$y\langle w_j\rangle$

Figure 2: Intervals collected from $C_1 \wedge \cdots \wedge C_m$

by (decreasing) bit-widths as described in Figure 2. In case only one bit-width is involved, the sequence is the singleton $(\mathcal{S}_1)$ and Algorithm 2 degenerates into Algorithm 1.

We now describe the internal mechanics of the algorithm. We know that, given all intervals forbidding values for different lower bits extracts of y, every value for y is ruled out. But for one particular bit-width w_i, the union of the w_i-intervals in model Γ does not necessarily cover the full domain $\mathbb{Z}/2^{w_i}\mathbb{Z}$ (i.e. $\bigcup_{I\in\mathcal{S}_i}[\![I]\!]_\Gamma$ may be different from $\mathbb{Z}/2^{w_i}\mathbb{Z}$). The coverage can leave "holes", and values in that hole are ruled out by constraints of other bit-widths. Algorithm 2 proceeds with bit-widths in decreasing order, starting with w_1. For every hole that $\bigcup_{I\in\mathcal{S}_1}[\![I]\!]_\Gamma$ leaves uncovered, it must determine how intervals of smaller bit-widths can cover it. Algorithm 2 is thus a *recursive* algorithm that calls itself on smaller bit-widths to cover the holes that the current layer leaves uncovered (termination of that recursion is thus trivial).

At every call, Algorithm 2 has the same structure as Algorithm 1, using the same variables longest and baseline and the same sub-routines LONGEST(_, _) and FURTHEST_EXTEND(_, _, _). In contrast to Algorithm 1, the call to FURTHEST_EXTEND(baseline, $\mathcal{S}_1, \Gamma$) is not guaranteed to succeed (as $[\![\mathsf{baseline}]\!]_\Gamma$ could be in an uncovered hole), were it not guarded by the condition that there is at least one interval in $\mathcal{S}_1$ that covers baseline in Γ (line 6). In that case the algorithm proceeds as Algorithm 1, and when output is extended with (baseline $\in I$) (line 8), the added constraint is expressed as described in Section 4.2, namely as $((\mathsf{baseline} - l) <^u (u - l))$, where $I = [l; u[$. In the opposite case, a hole has been discovered, and the extent of the hole is determined by calling a new sub-routine NEXT_COVERED_POINT(baseline, $\mathcal{S}_1, \Gamma$) (line 11); this call outputs the lower bound next of an interval in $\mathcal{S}_1$ that $\leq^u$-minimises $[\![\mathsf{next} - \mathsf{baseline}]\!]_\Gamma$.

If the hole is bigger than 2^{w_2} (i.e. $2^{w_2} \leq^u [\![\mathsf{next}-\mathsf{baseline}]\!]_\Gamma$), then the intervals of the next layers (starting with w_2-intervals) need to rule out every possible value for $y\langle w_2\rangle$, and the w_1-intervals were not needed for the coverage of layer w_1. So the algorithm returns the output of a recursive call on bit-width w_2 (line 17). If on the contrary the hole is smaller (i.e. $[\![\mathsf{next}-\mathsf{baseline}]\!]_\Gamma <^u 2^{w_2}$), then interval [baseline; next[in domain $\mathbb{Z}/2^{w_1}\mathbb{Z}$ is projected as an interval $[\mathsf{baseline}\langle w_2\rangle; \mathsf{next}\langle w_2\rangle[$ in domain $\mathbb{Z}/2^{w_2}\mathbb{Z}$ and needs to be covered by the intervals of bit-width w_2 and smaller. This is performed by a recursive call of the procedure on bit-width w_2 (line 14); the fact that only the hole $[\mathsf{baseline}\langle w_2\rangle; \mathsf{next}\langle w_2\rangle[$ needs to be covered by the recursive call, rather than the full domain $\mathbb{Z}/2^{w_2}\mathbb{Z}$, is implemented by adding to $\mathcal{S}_2$ in the recursive call the complement $[\mathsf{next}\langle w_2\rangle; \mathsf{baseline}\langle w_2\rangle[$ of $[\mathsf{baseline}\langle w_2\rangle; \mathsf{next}\langle w_2\rangle[$. The result of the recursive call is added to the output variable, as well as the fact that the hole must be small.

The final interpolant is $(\bigwedge_{E\in\mathsf{output}} E) \Rightarrow \bot$.[7]

Example 4. *Consider a variant of Example 1.2 with the constraints C_1, C_2, C_3, C_4 presented on the first line of Figure 3, and model $\Gamma = \{x_1 \mapsto 0b1100, x_2 \mapsto 0b1101, x_3 \mapsto 0b0000\}$. The second line is obtained from Figure 1, with the conditions on the third line being satisfied in Γ.*

[7] Note that Algorithm 2 here does not apply the same optimisation as Algorithm 1 (line 9-11), which detects when the use of interval longest is unnecessary, but it could also be done (at the cost of a more complex figure).

Algorithm 2 Producing the interpolating constraints with multiple bit-widths

function COVER($(\mathcal{S}_1, \ldots, \mathcal{S}_j), \Gamma$)
  output $\leftarrow \emptyset$ ▷ output initialised with the empty set of constraints
  longest $\leftarrow$ LONGEST($\mathcal{S}_1, \Gamma$) ▷ longest interval identified
  baseline $\leftarrow$ longest.upper ▷ where to extend the coverage from
  while $[\![\mathsf{baseline}]\!]_\Gamma \notin [\![\mathsf{longest}]\!]_\Gamma$ **do**
    if $\exists I \in \mathcal{S}_1, [\![\mathsf{baseline}]\!]_\Gamma \in [\![I]\!]_\Gamma$ **then**
      $I \leftarrow$ FURTHEST_EXTEND(baseline, $\mathcal{S}_1, \Gamma$)
      output $\leftarrow$ output $\cup \{\mathsf{baseline} \in I\}$ ▷ adding linking constraint
      baseline $\leftarrow I$.upper ▷ updating the baseline for the next interval pick
    else ▷ there is a hole in the coverage of $\mathbb{Z}/2^{w_1}\mathbb{Z}$ by intervals in $\mathcal{S}_1$
      next $\leftarrow$ NEXT_COVERED_POINT(baseline, $\mathcal{S}_1, \Gamma$) ▷ the hole is [baseline; next[
      if $[\![\mathsf{next}]\!]_\Gamma - [\![\mathsf{baseline}]\!]_\Gamma <^u 2^{w_2}$ **then**
        $I \leftarrow [\mathsf{next}\langle w_2\rangle; \mathsf{baseline}\langle w_2\rangle[$ ▷ it is projected on w_2 bits and complemented
        output $\leftarrow$ output $\cup$ (next$-$baseline $<^u 2^{w_2}$) $\cup$ COVER($((\mathcal{S}_2 \cup I), \mathcal{S}_3, \ldots, \mathcal{S}_j), \Gamma$)
        baseline $\leftarrow$ next ▷ updating the baseline for the next interval pick
      else ▷ intervals of bit-widths $\leq w_2$ must forbid all values for $y\langle w_2\rangle$
        return COVER($(\mathcal{S}_2, \ldots, \mathcal{S}_j), \Gamma$) ▷ $\mathcal{S}_1$ was not needed
  return output $\cup \{\mathsf{baseline} \in \mathsf{longest}\}$ ▷ adding final linking constraint

Constraint C	C_1 $\neg(y \simeq x_1)$	C_2 $(x_1 \leq^u x_3 + y)$	C_3 $(y\langle 2\rangle \leq^u x_2\langle 2\rangle)$	C_4 $(y\langle 1\rangle \simeq 0)$
Forbidden interval I_C	$[x_1; x_1 + 1[$	$[-x_3; x_1 - x_3[$	$[x_2\langle 2\rangle + 1; 0[$	$[1; 0[$
Condition c	$(0 \not\simeq -1)$	$(x_1 \not\simeq 0)$	$(x_2\langle 2\rangle \not\simeq -1)$	$(0 \not\simeq -1)$
Concrete interval $[\![I_C]\!]_\Gamma$	$[0b1100; 0b1101[$	$[0b0000; 0b1100[$	$[0b10; 0b00[$	$[0b1; 0b0[$
Bit-width w_i	$w_1 = 4$		$w_2 = 2$	$w_3 = 1$
Interval layer $\mathcal{S}_i$	$\mathcal{S}_1 = \{I_{C_1}, I_{C_2}\}$		$\mathcal{S}_2 = \{I_{C_3}\}$	$\mathcal{S}_3 = \{I_{C_4}\}$
Forbidding values for	y		$y\langle 2\rangle$	$y\langle 1\rangle$

Figure 3: Example with multiple bit-widths

Algorithm 2 identifies I_{C_2} as the longest among $\mathcal{S}_1$ in model Γ. The next interval among $\mathcal{S}_1$ covering $(x_1{-}x_3)$ in Γ is I_{C_1}, so $(x_1{-}x_3) \in I_{C_1}$ is added as an interpolant constraint E_1. Then $x_1{+}1$ is not covered in Γ by any interval in $\mathcal{S}_1$: it starts a hole that spans up to $-x_3$. The hole $[x_1{+}1; -x_3[$ has length $0b0011 <^u 2^2$ in Γ, so $(-x_3{-}x_1{-}1 <^u 2^2)$ is added as an interpolant constraint E_2 and a recursive call is made on $\mathcal{S}'_2 = \{I_{C_3}, I\}$ and $\mathcal{S}_3 = \{I_{C_4}\}$, where $I = [-x_3\langle 2\rangle; x_1\langle 2\rangle{+}1[$. The longest interval among $\mathcal{S}'_2$ in Γ is I_{C_3}, and it upper bound $0b00$ is covered in Γ by I, so $0b00 \in I$ is added as an interpolant constraint E_3. Then $x_1\langle 2\rangle{+}1$ is not covered in Γ by any interval in $\mathcal{S}'_2$: it starts a hole that spans up to $x_2\langle 2\rangle + 1$. The hole $[x_1\langle 2\rangle{+}1; x_2\langle 2\rangle{+}1[$ has length $0b01 <^u 2^1$ in Γ, so $(x_2\langle 2\rangle{-}x_1\langle 2\rangle <^u 2^1)$ is added as an interpolant constraint E_4 and a recursive call is made on $\mathcal{S}'_3 = \{I_{C_4}, I'\}$ where $I' = [x_2\langle 1\rangle{+}1; x_1\langle 1\rangle{+}1[$. Intervals I_{C_4} and I' finally cover $\mathbb{Z}/2\mathbb{Z}$, with $(x_1\langle 1\rangle{+}1) \in I_{C_4}$ and $0b0 \in I'$ added as interpolant constraints E_5 and E_6. Coming back from the recursive calls, $(x_2\langle 2\rangle{+}1) \in I_{C_3}$ and then $-x_3 \in I_{C_2}$ are added as interpolant constraints E_7 and E_8. The interpolant is $\bigwedge_{i=1}^{8} E_i \Rightarrow \bot$.

6 Conclusion

The interpolation mechanism presented in this paper (including for constraints with lower bits extraction) has been implemented in the MCSAT part of the Yices SMT-solver [Dut14]. The handling of 0-extensions and sign-extensions, which are present in numerous benchmarks of the SMTLib library [BST10], is not yet implemented; however, preliminary experimental results already show that our technique can perform very well on some benchmarks that can be difficult to address by other techniques. For instance the `QF_BV/pspace/ndist.*.smt2` benchmarks, are all solved in less than a second in our experimentation, as are the `QF_BV/pspace/shift1add.*.smt2` benchmarks. Note that on benchmarks that are not entirely in the bit-vector fragment described in this paper, our interpolation mechanism needs to collaborate with other interpolation mechanisms. Improving the way in which different interpolation mechanisms can collaborate for conflict analysis in MCSAT is one of our ongoing research directions. The treatment of the `QF_BV/pspace/shift1add.*.smt2` benchmarks is in fact an example of collaboration between the interpolation mechanism described in this paper, and that which we implemented for the fragment of bit-vector arithmetic consisting of extraction, concatenation, and equalities, and which we described in [GLJ17]. Further extensions of the scope of application of our interpolation method constitute future research directions, possibly by incorporating some of the quantifier elimination techniques described in [JC16].

Acknowledgments The research presented in this paper has been supported in part by NSF grant 1816936, and by DARPA project N66001-18-C-4011. 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, DARPA, or the U.S. Government. We wish to thank the anonymous reviewers for their constructive comments, and for pointing out [JC16].

References

[BST10] C. Barrett, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB), 2010. www.SMT-LIB.org.

[dMJ13] 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.

[Dut14] B. Dutertre. Yices 2.2. In A. Biere and R. Bloem, editors, *Proc. of the 26th Int. Conf. on Computer Aided Verification (CAV'14)*, volume 8559 of *LNCS*, pages 737–744. Springer-Verlag, 2014.

[GLJ17] 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.

[JC16] A. K. John and S. Chakraborty. A layered algorithm for quantifier elimination from linear modular constraints. *Formal Methods in System Design*, 49(3):272–323, 2016.

[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

[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 (RTA'06)*, volume 9710 of *LNCS*, pages 249–266. Springer-Verlag, 2016.