Solving Quantified Bit-Vectors
using Invertibility Conditions

Aina Niemetz[1], Mathias Preiner[1], Andrew Reynolds[2],
Clark Barrett[1], and Cesare Tinelli[2]

[1] Stanford University
[2] The University of Iowa

Abstract. We present a novel approach for solving quantified bit-vector formulas in Satisfiability Modulo Theories (SMT) based on computing symbolic inverses of bit-vector operators. We derive conditions that precisely characterize when bit-vector constraints are invertible for a representative set of bit-vector operators commonly supported by SMT solvers. We utilize syntax-guided synthesis techniques to aid in establishing these conditions and verify them independently by using several SMT solvers. We show that invertibility conditions can be embedded into quantifier instantiations using Hilbert choice expressions, and give experimental evidence that a counterexample-guided approach for quantifier instantiation utilizing these techniques leads to performance improvements with respect to state-of-the-art solvers for quantified bit-vector constraints.

1 Introduction

Many applications in hardware and software verification rely on Satisfiability Modulo Theories (SMT) solvers for bit-precise reasoning. In recent years, the quantifier-free fragment of the theory of fixed-size bit-vectors has received a lot of interest, as witnessed by the number of applications that generate problems in that fragment and by the high, and increasing, number of solvers that participate in the corresponding division of the annual SMT competition. Modeling properties of programs and circuits, e.g., universal safety properties and program invariants, however, often requires the use of *quantified* bit-vector formulas. Despite a multitude of applications, reasoning efficiently about such formulas is still a challenge in the automated reasoning community.

The majority of solvers that support quantified bit-vector logics employ instantiation-based techniques [24,21,8,20], which aim to find conflicting ground instances of quantified formulas. For that, it is crucial to select good instantiations for the universal variables, or else the solver may be overwhelmed by the number of ground instances generated. For example, consider a quantified formula $\psi = \forall x. (x + s \not\approx t)$ where x, s and t denote bit-vectors of size 32. To prove that ψ is unsatisfiable we can instantiate x with all 2^{32} possible bit-vector values. However, ideally, we would like to find a proof that requires much fewer instantiations. In this example, if we instantiate x with the symbolic term $t - s$ (the inverse of $x + s \approx t$ when solved for x), we can immediately conclude that ψ is unsatisfiable since $(t - s) + s \not\approx t$ simplifies to false.

Operators in the theory of bit-vectors are not always invertible. However, we observe it is possible to identify quantifier-free conditions that precisely *characterize* when

they are. We do that for a representative set of operators in the standard theory of bit-vectors supported by SMT solvers. For example, we have proven that the constraint $x \cdot s \approx t$ is solvable for x if and only if $(-s \mid s)\ \&\ t \approx t$ is satisfiable. Using this observation, we develop a novel approach for solving quantified bit-vector formulas that utilizes invertibility conditions to generate symbolic instantiations. We show that invertibility conditions can be embedded into quantifier instantiations using Hilbert choice functions in a sound manner. This approach has compelling advantages with respect to previous approaches, which we demonstrate in our experiments.

More specifically, this paper makes the following *contributions*.

– We derive and present invertibility conditions for a representative set of bit-vector operators that allow us to model all bit-vector constraints in SMT-LIB [3].
– We provide details on how invertibility conditions can be automatically synthesized using syntax-guided synthesis (SyGuS) [1] techniques, and make public 162 available challenge problems for SyGuS solvers that are encodings of this task.
– We prove that our approach can efficiently reduce a class of quantified formulas, which we call *unit linear invertible*, to quantifier-free constraints.
– Leveraging invertibility conditions, we implement a novel quantifier instantiation scheme as an extension of the SMT solver CVC4 [2], which shows improvements with respect to state-of-the-art solvers for quantified bit-vector constraints.

Related work Quantified bit-vector logics are currently supported by the SMT solvers Boolector [16], CVC4 [2], Yices [7], and Z3 [6] and a Binary Decision Diagram (BDD)-based tool called Q3B [14]. Out of these, only CVC4 and Z3 provide support for combining quantified bit-vectors with other theories, e.g., the theories of arrays or real arithmetic. Arbitrarily nested quantifiers are handled by all but Yices, which only supports bit-vector formulas of the form $\exists \boldsymbol{x} \forall \boldsymbol{y}.\, Q[\boldsymbol{x}, \boldsymbol{y}]$ [8]. For quantified bit-vectors, CVC4 employs counterexample-guided quantifier instantiation (CEGQI) [21], where concrete models of a set of ground instances and the negation of the input formula (the counterexamples) serve as instantiations for the universal variables. In Z3, model-based quantifier instantiation (MBQI) [10] is combined with a template-based model finding procedure [24]. In contrast to CVC4, Z3 not only relies on concrete counterexamples as candidates for quantifier instantiation but generalizes these counterexamples to generate symbolic instantiations by selecting ground terms with the same model value. Boolector employs a syntax-guided synthesis approach to synthesize interpretations for Skolem functions based on a set of ground instances of the formula, and uses a counterexample refinement loop similar to MBQI [20]. Other counterexample-guided approaches for quantified formulas in SMT solvers have been considered by Bjørner and Janota [4] and by Reynolds et al. [22], but they have mostly targeted quantified linear arithmetic and do not specifically address bit-vectors. Quantifier elimination for a fragment of bit-vectors that covers modular linear arithmetic has been recently addressed by John and Chakraborty [13], although we do not explore that direction in this paper.

2 Preliminaries

We assume the usual notions and terminology of many-sorted first-order logic with equality (denoted by $\approx$). Let S be a set of *sort symbols*, and for every sort $\sigma \in S$ let X_σ

be an infinite set of *variables of sort σ*. We assume that sets X_σ are pairwise disjoint and define X as the union of sets X_σ. Let Σ be a *signature* consisting of a set $\Sigma^s \subseteq S$ of sort symbols and a set Σ^f of interpreted (and sorted) function symbols $f^{\sigma_1 \cdots \sigma_n \sigma}$ with arity $n \geq 0$ and $\sigma_1, ..., \sigma_n, \sigma \in \Sigma^s$. We assume that a signature Σ includes a Boolean sort Bool and the Boolean constants $\top$ (true) and $\bot$ (false). Let $\mathcal{I}$ be a Σ-*interpretation* that maps: each $\sigma \in \Sigma^s$ to a non-empty set $\sigma^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$), with $\text{Bool}^{\mathcal{I}} = \{\top, \bot\}$; each $x \in X_\sigma$ to an element $x^{\mathcal{I}} \in \sigma^{\mathcal{I}}$; and each $f^{\sigma_1 \cdots \sigma_n \sigma} \in \Sigma^f$ to a total function $f^{\mathcal{I}}: \sigma_1^{\mathcal{I}} \times ... \times \sigma_n^{\mathcal{I}} \to \sigma^{\mathcal{I}}$ if $n > 0$, and to an element in $\sigma^{\mathcal{I}}$ if $n = 0$. If $x \in X_\sigma$ and $v \in \sigma^{\mathcal{I}}$, we denote by $\mathcal{I}[x \mapsto v]$ the interpretation that maps x to v and is otherwise identical to $\mathcal{I}$. We use the usual inductive definition of a satisfiability relation $\models$ between Σ-interpretations and Σ-formulas.

We assume the usual definition of well-sorted terms, literals, and formulas as Bool terms with variables in X and symbols in Σ, and refer to them as Σ-terms, Σ-atoms, and so on. A *ground* term/formula is a Σ-term/formula without variables. We define $\boldsymbol{x} = (x_1, ..., x_n)$ as a tuple of variables and write $Q\boldsymbol{x}\varphi$ with $Q \in \{\forall, \exists\}$ for a *quantified* formula $Qx_1 \cdots Qx_n\varphi$. We use $\text{Lit}(\varphi)$ to denote the set of Σ-literals of Σ-formula φ. For a Σ-term or Σ-formula e, we denote the *free variables* of e (defined as usual) as $FV(e)$ and use $e[\boldsymbol{x}]$ to denote that the variables in $\boldsymbol{x}$ occur free in e. For a tuple of Σ-terms $\boldsymbol{t} = (t_1, ..., t_n)$, we write $e[\boldsymbol{t}]$ for the term or formula obtained from e by simultaneously replacing each occurrence of x_i in e by t_i. Given a Σ-formula $\varphi[x]$ with $x \in X_\sigma$, we use Hilbert's *choice* operator ε [12] to describe *properties* of x. We define a *choice function* $\varepsilon x.\, \varphi[x]$ as a term where x is bound by ε. In every interpretation $\mathcal{I}$, $\varepsilon x.\, \varphi[x]$ denotes some value $v \in \sigma^{\mathcal{I}}$ such that $\mathcal{I}[x \mapsto v]$ satisfies $\varphi[x]$ if such values exist, and denotes an arbitrary element of $\sigma^{\mathcal{I}}$ otherwise. This means that the formula $\exists x.\, \varphi[x] \Leftrightarrow \varphi[\varepsilon x.\, \varphi[x]]$ is satisfied by every interpretation.

A *theory* T is a pair (Σ, I), where Σ is a signature and I is a non-empty class of Σ-interpretations (the *models* of T) that is closed under variable reassignment, i.e., every Σ-interpretation that only differs from an $\mathcal{I} \in I$ in how it interprets variables is also in I. A Σ-formula φ is *T-satisfiable* (resp. *T-unsatisfiable*) if it is satisfied by some (resp. no) interpretation in I; it is *T-valid* if it is satisfied by all interpretations in I. We say T is a *complete* theory if for all closed Σ-formulas φ, φ is either T-valid or T-unsatisfiable. A choice function $\varepsilon x.\, \varphi[x]$ is *(T-)valid* if $\exists x.\, \varphi[x]$ is *(T-)valid*. We refer to a term t as *ε-(T-)valid* if all occurrences of choice functions in t are *(T-)valid*. We will sometimes omit T when the theory is understood from context.

We will focus on the theory $T_{BV} = (\Sigma_{BV}, I_{BV})$ of fixed-size bit-vectors as defined by the SMT-LIB 2 standard [3]. The signature Σ_{BV} includes a unique sort for each positive bit-vector width n, denoted here as $\sigma_{[n]}$. Similarly, $X_{[n]}$ is the set of *bit-vector variables* of sort $\sigma_{[n]}$, and X_{BV} is the union of all sets $X_{[n]}$. We assume that Σ_{BV} includes all *bit-vector constants* of sort $\sigma_{[n]}$ for each n, represented as bit-strings. However, to simplify the notation we will sometimes denote them by the corresponding natural number in $\{0, \dots, 2^{n-1}\}$. All interpretations $\mathcal{I} \in I_{BV}$ are identical except for the value they assign to variables. They interpret sort and function symbols as specified in SMT-LIB 2. All function symbols in Σ_{BV}^f are overloaded for every $\sigma_{[n]} \in \Sigma_{BV}^s$. We denote a Σ_{BV}-term (or *bit-vector term*) t of width n as $t_{[n]}$ when we want to specify its bit-width explicitly. We use $\max_{s[n]}$ or $\min_{s[n]}$ for the *maximum* or *minimum signed*

Symbol	SMT-LIB Syntax	Sort
$\approx, <_u, >_u, <_s, >_s$	=, bvult, bvugt, bvslt, bvsgt	$\sigma_{[n]} \times \sigma_{[n]} \to \mathsf{Bool}$
$\sim, -$	bvnot, bvneg	$\sigma_{[n]} \to \sigma_{[n]}$
$\&, |, \ll, \gg, \gg_a$	bvand, bvor, bvshl, bvlshr, bvashr	$\sigma_{[n]} \times \sigma_{[n]} \to \sigma_{[n]}$
$+, \cdot, \mathrm{mod}, \div$	bvadd, bvmul, bvurem, bvudiv	$\sigma_{[n]} \times \sigma_{[n]} \to \sigma_{[n]}$
$\circ$	concat	$\sigma_{[n]} \times \sigma_{[m]} \to \sigma_{[n+m]}$
$[u:l]$	extract	$\sigma_{[n]} \to \sigma_{[u-l+1]}, 0 \le l \le u < n$

Table 1. Set of considered bit-vector operators with corresponding SMT-LIB 2 syntax.

value of width n, e.g., $\max_{s[4]} = 0111$ and $\min_{s[4]} = 1000$. The width of a bit-vector sort or term is given by the function κ, e.g., $\kappa(\sigma_{[n]}) = n$ and $\kappa(t_{[n]}) = n$.

Without loss of generality, we consider a restricted set of bit-vector function symbols (or *bit-vector operators*) Σ_{BV}^f as listed in Table 1. The selection of operators in this set is arbitrary but complete in the sense that it suffices to express all bit-vector operators defined in SMT-LIB 2.

3 Invertibility Conditions for Bit-Vector Constraints

This section formally introduces the concept of an invertibility condition and shows that such conditions can be used to construct symbolic solutions for a class of quantifier-free bit-vector constraints that have a linear shape.

Consider a bit-vector literal $x + s \approx t$ and assume that we want to solve for x. If the literal is *linear* in x, that is, has only one occurrence of x, a general solution for x is given by the inverse of bit-vector addition over equality: $x = t - s$. Computing the inverse of a bit-vector operation, however, is not always possible. For example, for $x \cdot s \approx t$, an inverse always exists only if s always evaluates to an odd bit-vector. Otherwise, there are values for s and t where no such inverse exists, e.g., $x \cdot 2 \approx 3$. However, even if there is no unconditional inverse for the general case, we can identify the condition under which a bit-vector operation is invertible. For the bit-vector multiplication constraint $x \cdot s \approx t$ with $x \notin FV(s) \cup FV(t)$, the *invertibility condition* for x can be expressed by the formula $(-s \mid s) \mathbin{\&} t \approx t$.

Definition 1. *(Invertibility Condition) Let $\ell[x]$ be a Σ_{BV}-literal. A quantifier-free Σ_{BV}-formula ϕ_c is an* invertibility condition *for x in $\ell[x]$ if $x \notin FV(\phi_c)$ and $\phi_c \Leftrightarrow \exists x.\, \ell[x]$ is T_{BV}-valid.*

An invertibility condition for a literal $\ell[x]$ provides the *exact conditions* under which $\ell[x]$ is solvable for x. We call it an "invertibility" condition because we can use Hilbert choice functions to express *all* such conditional solutions with a *single* symbolic term, that is, a term whose possible values are exactly the solutions for x in $\ell[x]$. Recall that a choice function $\varepsilon y.\, \varphi[y]$ represents a solution for a formula $\varphi[x]$ if there exists one, and represents an arbitrary value otherwise. We may use a choice function to describe inverse solutions for a literal $\ell[x]$ with invertibility condition ϕ_c as $\varepsilon y.\, (\phi_c \Rightarrow \ell[y])$. For example, for the general case of bit-vector multiplication over equality the choice function is defined as $\varepsilon y.\, ((-s \mid s) \mathbin{\&} t \approx t \;\Rightarrow\; y \cdot s \approx t)$.

```
solve(x, e[x] ⋈ t):
    If e = x
        If ⋈ ∈ {≈} then return t
        else return εy. (getIC(x, x ⋈ t) ⇒ y ⋈ t).
    else e = ◇(e₁, …, eᵢ[x], …, eₙ) with n > 0 and x ∉ FV(eⱼ) for all j ≠ i.
        Let d[x′] = ◇(e₁, …, eᵢ₋₁, x′, eᵢ₊₁, …, eₙ) where x′ is a fresh variable.
        If ⋈ ∈ {≈, ≉} and ◇ ∈ {∼, −, +}
        then let t′ = getInverse(x′, d[x′] ≈ t) and return solve(x, eᵢ ⋈ t′)
        else let φc = getIC(x′, d[x′] ⋈ t) and return solve(x, eᵢ ≈ εy. (φc ⇒ d[y] ⋈ t)).
```

Fig. 1. Function solve for constructing a symbolic solution for x given a linear literal $e[x] \bowtie t$.

Lemma 2. *If ϕ_c is an invertibility condition for an ε-valid Σ_{BV}-literal $\ell[x]$ and r is the term $\varepsilon y. (\phi_c \Rightarrow \ell[y])$, then r is ε-valid and $\ell[r] \Leftrightarrow \exists x. \ell[x]$ is T_{BV}-valid.*

Proof. First we show that $r = \varepsilon y. (\phi_c \Rightarrow \ell[y])$ is ε-valid, where ϕ_c is an invertibility condition for y in $\ell[y]$. To do so, since $\ell[y]$ is ε-valid, we must show $\exists y. \phi_c \Rightarrow \ell[y]$ holds in all models of T_{BV}. Since ϕ_c is an invertibility condition for $\ell[y]$, we have that $y \notin FV(\phi_c)$ and hence this formula is equivalent to $\phi_c \Rightarrow \exists y. \ell[y]$. Let $\mathcal{I}$ be any model of T_{BV} that satisfies ϕ_c. Since ϕ_c is an invertibility condition for $\ell[y]$, by Definition 1, $\mathcal{I}$ satisfies $\exists y. \ell[y]$ as well. Thus, $\exists y. \phi_c \Rightarrow \ell[y]$ holds in all models of T_{BV} and hence r is ε-valid.

To show $\ell[r] \Leftrightarrow \exists x. \ell[x]$ where r is $\varepsilon y. (\phi_c \Rightarrow \ell[y])$, first consider direction $\exists x. \ell[x] \Rightarrow \ell[r]$. Let $\mathcal{I}$ be any model of T_{BV} that satisfies $\exists x. \ell[x]$. By definition of ε, $\mathcal{I}$ also satisfies $\ell[\varepsilon y. \ell[y]]$. Since ϕ_c is an invertibility condition for $\ell[x]$, from Def. 1 we have that $\phi_c \Leftrightarrow \exists y. \ell[y]$ holds in all models of T_{BV}, and thus $\mathcal{I}$ also satisfies ϕ_c. Hence, since $\mathcal{I}$ satisfies $\ell[\varepsilon y. \ell[y]]$, it also satifies $\ell[\varepsilon y. (\phi_c \Rightarrow \ell[y])]$, which is $\ell[r]$. Thus, $\exists x. \ell[x] \Rightarrow \ell[r]$ is T_{BV}-valid. The other direction $\ell[r] \Rightarrow \exists x. \ell[x]$ trivially holds in all models of T_{BV}. □

Intuitively, the lemma states that when $\ell[x]$ is satisfiable (under condition ϕ_c), any value returned by the choice function $\varepsilon y. (\phi_c \Rightarrow \ell[y])$ is a solution of $\ell[x]$ (and thus $\exists x. \ell[x]$ holds). Conversely, if there exists a value v for x that makes $\ell[x]$ true, then there is a model of T_{BV} that interprets $\varepsilon y. (\phi_c \Rightarrow \ell[y])$ as v.

Now, suppose that Σ_{BV}-literal ℓ is again linear in x but that x occurs arbitrarily deep in ℓ. Consider, for example, a literal $s_1 \cdot (s_2 + x) \approx t$ where x does not occur in s_1, s_2 or t. We can solve this literal for x by recursively computing the (possibly conditional) inverses of all bit-vector operations that involve x. That is, first we solve $s_1 \cdot x' \approx t$ for x', where x' is a fresh variable abstracting $s_2 + x$, which yields the choice function $x' = \varepsilon y. ((-s_1 \mid s_1) \mathbin{\&} t \approx t \Rightarrow s_1 \cdot y \approx t)$. Then, we solve $s_2 + x \approx x'$ for x, which yields the solution $x = x' - s_2 = \varepsilon y. ((-s_1 \mid s_1) \mathbin{\&} t \approx t \Rightarrow s_1 \cdot y \approx t) - s_2$.

Figure 1 describes in pseudo code the procedure to solve for x in an arbitrary literal $\ell[x] = e[x] \bowtie t$ that is linear in x. We assume that $e[x]$ is built over the set of bit-vector operators listed in Table 1. Function solve recursively constructs a symbolic solution by computing (conditional) inverses as follows. Let function getInverse$(x, \ell[x])$ return a term t' that is the inverse of x in $\ell[x]$, i.e., such that $\ell[x] \Leftrightarrow x \approx t'$. Furthermore,

let function getIC$(x, \ell[x])$ return the invertibility condition ϕ_c for x in $\ell[x]$. If $e[x]$ has the form $\diamond(e_1, \ldots, e_n)$ with $n > 0$, x must occur in exactly one of the subterms $e_1, \ldots, e_n$ given that e is linear in x. Let d be the term obtained from e by replacing e_i (the subterm containing x) with a fresh variable x'. We solve for subterm $e_i[x]$ (treating it as a variable x') and compute an inverse getInverse$(x', d[x'] \approx t)$, if it exists. Note that for a disequality $e[x] \not\approx t$, it suffices to compute the inverse over equality and propagate the disequality down. (For example, for $e_i[x] + s \not\approx t$, we compute the inverse $t' = $ getInverse$(x', x' + s \approx t) = t - s$ and recurse on $e_i[x] \not\approx t'$.) If no inverse for $e[x] \bowtie t$ exists, we first determine the invertibility condition ϕ_c for $d[x']$ via getIC$(x', d[x'] \bowtie t)$, construct the choice function $\varepsilon y. (\phi_c \Rightarrow d[y] \bowtie t)$, and set it equal to $e_i[x]$, before recursively solving for x. If $e_i[x] = x$ and the given literal is an equality, we have reached the base case and return t as the solution for x. Note that in Figure 1, for simplicity we omitted one case for which an inverse can be determined, namely $x \cdot c \approx t$ where c is an odd constant.

Theorem 3. *Let $\ell[x]$ be an ε-valid Σ_{BV}-literal linear in x, and let $r = $ solve$(x, \ell[x])$. Then r is ε-valid, $FV(r) \subseteq FV(\ell) \setminus \{x\}$ and $\ell[r] \Leftrightarrow \exists x. \ell[x]$ is T_{BV}-valid.*

Proof. We assume without loss of generality that $\ell[x]$ is of the form $e[x] \bowtie t$. Since $\ell[x]$ is linear with respect to x, we have that $x \notin FV(t)$. We show all statements of the Theorem by structural induction on the term $e[x]$.

Consider the case when $e[x]$ is x. If $\bowtie$ is $\approx$, then r is t. We have that r is ε-valid, $x \notin FV(r)$ since $x \notin FV(t)$ and $t \approx t \Leftrightarrow \exists x. x \approx t$ holds in all models of T_{BV}. Otherwise, when $\bowtie$ is not $\approx$, we have that r is $\varepsilon y. (\psi \Rightarrow y \bowtie t)$ where ψ is getIC$(x', x' \bowtie t)$. By definition of getIC, we have that $FV(\psi) \subseteq FV(t)$, and hence $FV(r)$ is a subset of $FV(t)$, which is equal to $FV(\ell[x]) \setminus \{x\}$. Furthermore, since ψ is an invertibility condition for x' in $x' \bowtie t$, by Lemma 2, r is ε-valid and $r \bowtie t \Leftrightarrow \exists x. x \bowtie t$.

Otherwise, $e[x]$ must be of the form $\diamond(e_1, ..., e_i[x], ..., e_n)$ for $n > 0$, where $x \notin FV(e_j)$ for each $i \neq j$. Let $d[x']$ be $\diamond(e_1, ..., e_{i-1}, x', e_{i+1}, ...e_n)$ where notice that $x \notin FV(d[x'])$. We have that r is solve$(x, e_i[x] \bowtie_i t_i)$ for some relation $\bowtie_i$ and term t_i, where t_i is either getInverse$(x', d[x'] \approx t)$ or $\varepsilon y. (\text{getIC}(x', d[x'] \bowtie t) \Rightarrow d[y] \bowtie t)$. In both cases, by definition of getInverse and getIC, we have that t_i is ε-valid due to Lemma 2 and since $\ell[x]$ is ε-valid. Also in both cases, we have that $FV(t_i) \subseteq FV(t) \cup FV(d[x']) \setminus \{x'\} \subseteq FV(\ell[x]) \setminus \{x\}$. Since $e[x] \bowtie t$ is ε-valid and linear with respect to x and $x \notin FV(t_i)$, the literal $e_i[x] \bowtie_i t_i$ is ε-valid and linear with respect to x as well. Thus, by the induction hypothesis, we have that r is ε-valid, $FV(r) \subseteq FV(e_i[x] \bowtie_i t_i) \setminus \{x\}$ and the formula $e_i[r] \bowtie_i t_i \Leftrightarrow \exists x. e_i[x] \bowtie_i t_i$ holds in all models of T_{BV}.

Since $FV(r) \subseteq FV(e_i[x] \bowtie_i t_i) \setminus \{x\}$ and since $FV(e_i[x]) \subseteq FV(\ell[x])$ and $FV(t_i) \subseteq FV(\ell[x])$, we have that $FV(r) \subseteq FV(\ell[x]) \setminus \{x\}$.

It remains to show $e[r] \bowtie t \Leftrightarrow \exists x. e[x] \bowtie t$. In the case that $\bowtie \in \{\approx, \not\approx\}$ and $\diamond \in \{\sim, -, +\}$, we have that $\bowtie_i$ is $\bowtie$ and t_i is getInverse$(x', d[x'] \approx t)$. By definition of getInverse and since $\bowtie \in \{\approx, \not\approx\}$, we have that $e_i[x] \bowtie t_i$ and $d[e_i[x]] \bowtie t$ are equivalent. Since $d[e_i[x]] = e[x]$, the latter literal is $e[x] \bowtie t$. Thus, since $e_i[r] \bowtie t_i \Leftrightarrow \exists x. e_i[x] \bowtie t_i$, we have that $e[r] \bowtie t \Leftrightarrow \exists x. e[x] \bowtie t$. Otherwise, we have that $\bowtie_i$ is $\approx$ and t_i is $\varepsilon y. (\text{getIC}(x', d[x'] \bowtie t) \Rightarrow d[y] \bowtie t)$. Since ψ is an invertibility condition for x' in $d[x'] \bowtie t$, by Lemma 2, we have that $d[t_i] \bowtie t \Leftrightarrow \exists x_i. d[x_i] \bowtie t$ holds in all

models of T_{BV}. Clearly $e[r] \bowtie t \Rightarrow \exists x. e[x] \bowtie t$ holds in all models of T_{BV}. Now, consider any model $\mathcal{I}$ of T_{BV} that satisfies $\exists x. e[x] \bowtie t$. Since $e[x] = d[e_i[x]]$, we have that $\mathcal{I}$ satisfies $\exists x_i. d[x_i] \bowtie t$ as well. Thus, since $d[t_i] \bowtie t \Leftrightarrow \exists x_i. d[x_i] \bowtie t$ holds in all models of T_{BV}, we have that $\mathcal{I}$ satisfies $d[t_i] \bowtie t$. Notice that $\mathcal{I}$ satisfies $\exists x. d[e_i[x]] \bowtie t$ by assumption and $\mathcal{I}$ satisfies $d[t_i] \bowtie t$. Thus, since t_i is $\varepsilon y. (\psi \Rightarrow d[y] \bowtie t)$, we have that $\exists x_i. e_i[x_i] \approx t_i$. Since by the induction hypothesis $e_i[r] \approx t_i \Leftrightarrow \exists x. e_i[x] \approx t_i$ holds in all models of T_{BV}, we have that $\mathcal{I}$ satisfies $e_i[r] \approx t_i$. Thus, since $\mathcal{I}$ satisfies $d[t_i] \bowtie t$, we have that $\mathcal{I}$ also satisfies $d[e_i[r]] \bowtie t$, which is $e[r] \bowtie t$. Thus, $e[r] \bowtie t \Leftarrow \exists x. e[x] \bowtie t$ holds in all models of T_{BV}. $\qquad\square$

Tables 2-3 list the invertibility conditions for bit-vector operators $\{\cdot, \mod, \div, \&,$ $|, \gg, \gg_a, \ll, \circ\}$ over relations $\{\approx, \not\approx, <_u, >_u\}$. Due to space restrictions we omit the conditions for signed inequalities since they can be expressed in terms of unsigned inequality. We omit the invertibility conditions over $\{\leq_u, \geq_u\}$ since they can generally be constructed by combining the corresponding conditions for equality and inequality—although there might be more succinct equivalent conditions.

Table 4 shows the rules for inverse computation for bit-vector operators $\{-, \sim, +, \cdot\}$ over equality. Tables 5-8 list the remaining invertibility conditions for $x \bowtie t$ and bit-vector operators $\{\sim, -, +, \cdot, \mod, \div, \&, |, \gg, \gg_a, \ll, \circ\}$.

The idea of computing the inverse of bit-vector operators has been used successfully in a recent local search approach for solving quantifier-free bit-vector constraints by Niemetz et al. [17]. There, target values are propagated via inverse value computation. In contrast, our approach does not determine single inverse values based on concrete assignments but aims at finding symbolic solutions through the generation of conditional inverses. In an extended version of that work [18], the same authors present rules for inverse value computation over equality but they provide no proof of correctness for them. We define invertibility conditions not only over equality but also disequality and (un)signed inequality, and verify their correctness up to a certain bit-width.

3.1 Synthesizing Invertibility Conditions

We have defined invertibility conditions for all bit-vector operators in Σ_{BV} where no general inverse exists (162 in total). A noteworthy aspect of this work is that we were able to leverage syntax-guided synthesis (SyGuS) technology [1] to help identify these conditions. The problem of finding invertibility conditions for a literal of the form $x \diamond s \bowtie t$ (or, dually, $s \diamond x \bowtie t$) linear in x can be recast as a SyGuS problem by asking whether there exists a binary Boolean function C such that the (second-order) formula $\exists C \forall s \forall t. ((\exists x. x \diamond s \bowtie t) \Leftrightarrow C(s, t))$ is satisfiable. If a SyGuS solver is able to synthesize the function C, then C can be used as the invertibility condition for $x \diamond s \bowtie t$. To simplify the SyGuS problem we chose a bit-width of 4 for x, s, and t and eliminated the quantification over x in the formula above by by expanding it to

$$\exists C \forall s \forall t. \left(\bigvee_{i=0}^{15} i \diamond s \bowtie t \right) \Leftrightarrow C(s, t)$$

Since the search space for SyGuS solvers heavily depends on the input grammar (which defines the solution space for C), we decided to use two grammars with the same set of

$\ell[x]$	$\approx$	$\not\approx$
$x \cdot s \bowtie t$	$(-s \mid s)\ \&\ t \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \bmod s \bowtie t$	$\sim(-s) \geq_u t$	$s \not\approx 1 \vee t \not\approx 0$
$s \bmod x \bowtie t$	$(t + t - s)\ \&\ s \geq_u t$	$s \not\approx 0 \vee t \not\approx 0$
$x \div s \bowtie t$	$(s \cdot t) \div s \approx t$	$s \not\approx 0 \vee t \not\approx \sim 0$
$s \div x \bowtie t$	$s \div (s \div t) \approx t$	$\begin{cases} s\ \&\ t \approx 0 & \text{for } \kappa(s) = 1 \\ \top & \text{otherwise} \end{cases}$
$x\ \&\ s \bowtie t$	$t\ \&\ s \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \mid s \bowtie t$	$t \mid s \approx t$	$s \not\approx \sim 0 \vee t \not\approx \sim 0$
$x \gg s \bowtie t$	$(t \ll s) \gg s \approx t$	$t \not\approx 0 \vee s <_u \kappa(s)$
$s \gg x \bowtie t$	$\displaystyle\bigvee_{i=0}^{\kappa(s)} s \gg i \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \gg_a s \bowtie t$	$(s <_u \kappa(s) \Rightarrow (t \ll s) \gg_a s \approx t) \wedge$ $(s \geq_u \kappa(s) \Rightarrow (t \approx \sim 0 \vee t \approx 0))$	$\top$
$s \gg_a x \bowtie t$	$\displaystyle\bigvee_{i=0}^{\kappa(s)} s \gg_a i \approx t$	$(t \not\approx 0 \vee s \not\approx 0) \wedge$ $(t \not\approx \sim 0 \vee s \not\approx \sim 0)$
$x \ll s \bowtie t$	$(t \gg s) \ll s \approx t$	$t \not\approx 0 \vee s <_u \kappa(s)$
$s \ll x \bowtie t$	$\displaystyle\bigvee_{i=0}^{\kappa(s)} s \ll i \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \circ s \bowtie t$	$s \approx t[\kappa(s) - 1 : 0]$	$\top$
$s \circ x \bowtie t$	$s \approx t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	$\top$

Table 2. Conditions for the invertibility of bit-vector operators over (dis)equality. Those for $\cdot$, $\&$ and $\mid$ are given modulo commutativity of those operators.

Boolean connectives but different sets of bit-vector operators:

$$O_r = \{\neg, \wedge, \approx, <_u, <_s, 0, \min_s, \max_s, s, t, \sim, -, \&, \mid\}$$
$$O_g = \{\neg, \wedge, \vee, \approx, <_u, <_s, \geq_u, \geq_s, 0, \min_s, \max_s, s, t, \sim, +, -, \&, \mid, \gg, \ll\}$$

The selection of constants in the grammar turned out to be crucial for finding solutions, e.g., by adding $\min_s$ and $\max_s$ we were able to synthesize substantially more invertibility conditions for signed inequalities. For each of the two sets of operators, we generated 140 SyGuS problems[3], one for each combination of bit-vector operator $\diamond \in \{\cdot, \bmod, \div, \&, \mid, \gg, \gg_a, \ll\}$ over relation $\bowtie \in \{\approx, \not\approx, <_u, \leq_u, >_u, \geq_u, <_s, \leq_s, >_s, \geq_s\}$, and used the SyGuS extension of the CVC4 solver [21] to solve these problems.

Using operators O_r (O_g) we were able to synthesize 98 (116) out of 140 invertibility conditions, with 118 unique solutions overall. When we found more than one solution for a condition (either with operators O_r and O_g, or manually) we chose the one that

---

[3] Available at https://cvc4.cs.stanford.edu/papers/CAV2018-QBV/

$\ell[x]$	$<_u$	$>_u$
$x \cdot s \bowtie t$	$t \not\approx 0$	$t <_u -s \mid s$
$x \bmod s \bowtie t$	$t \not\approx 0$	$t <_u \sim(-s)$
$s \bmod x \bowtie t$	$t \not\approx 0$	$t <_u s$
$x \div s \bowtie t$	$0 <_u s \wedge 0 <_u t$	$\sim 0 \div s >_u t$
$s \div x \bowtie t$	$0 <_u \sim(-t \mathbin{\&} s) \wedge 0 <_u t$	$t <_u \sim 0$
$x \mathbin{\&} s \bowtie t$	$t \not\approx 0$	$t <_u s$
$x \mid s \bowtie t$	$s <_u t$	$t <_u \sim 0$
$x \gg s \bowtie t$	$t \not\approx 0$	$t <_u \sim s \gg s$
$s \gg x \bowtie t$	$t \not\approx 0$	$t <_u s$
$x \gg_a s \bowtie t$	$t \not\approx 0$	$t <_u \sim 0$
$s \gg_a x \bowtie t$	$(s <_u t \vee s \geq_s 0) \wedge t \not\approx 0$	$s <_s (s \gg \sim t) \vee t <_u s$
$x \ll s \bowtie t$	$t \not\approx 0$	$t <_u \sim 0 \ll s$
$s \ll x \bowtie t$	$t \not\approx 0$	$\displaystyle\bigvee_{i=0}^{\kappa(s)} (s \ll i) >_u t$
$x \circ s \bowtie t$	$t_x \approx 0 \Rightarrow s <_u t_s$	$t_x \approx 0 \Rightarrow s >_u t_s$
	where $t_x = t[\kappa(t) - 1 : \kappa(t) - \kappa(x)], t_s = t[\kappa(s) - 1 : 0]$	
$s \circ x \bowtie t$	$s \leq_u t_s \wedge (s \approx t_s \Rightarrow t_x \not\approx 0)$	$s \geq_u t_s \wedge s \approx t_s \Rightarrow t_x \not\approx \sim 0$
	where $t_x = t[\kappa(x) - 1 : 0], t_s = t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	

Table 3. Conditions for the invertibility of bit-vector operators over unsigned inequality. Those for $\cdot$, $\&$ and $\mid$ are given modulo commutativity of those operators.

involved the smallest number of bit-vector operators. Thus, we ended up using 79 out of 118 synthesized conditions and 83 manually crafted conditions.

In some cases, the SyGuS approach was able to synthesize invertibility conditions that were smaller than those we had manually crafted. For example, we manually defined the invertibility condition for $x \cdot s \approx t$ as $(t \approx 0) \vee ((t \mathbin{\&} -t) \geq_u (s \mathbin{\&} -s) \wedge (s \not\approx 0))$. With SyGuS we obtained $((-s \mid s) \mathbin{\&} t) \approx t$. For some other cases, however, the synthesized solution involved more bit-vector operators than needed. For example, for $x \bmod s \not\approx t$ we manually defined the invertibility condition $(s \not\approx 1) \vee (t \not\approx 0)$, whereas SyGuS produced the solution $\sim(-s) \mid t \not\approx 0$. For the majority of invertibility conditions, finding a solution did not require more than one hour of CPU time on an Intel Xeon E5-2637 with 3.5GHz. Interestingly, the most time-consuming synthesis task (over 107 hours of CPU time) was finding condition $((t + t) - s) \mathbin{\&} s \geq_u t$ for

$\ell[x]$	$-x \approx t$	$\sim x \approx t$	$x + s \approx t$	$x \cdot s \approx t$ with s const
$\mathsf{getInverse}(x, \ell[x])$	$-t$	$\sim t$	$t - s$	$t \cdot s^{-1}$ with $s \cdot s^{-1} \approx 1$

Table 4. Inverse computation for bit-vector operators $\{-, \sim, +, \cdot\}$ over $\approx$. Those for $+$ and $\cdot$ are given modulo commutativity of those operators.

$\ell[x]$	$<_u$	$>_u$	$<_s$	$>_s$	$\leq_u, \geq_u, \leq_s, \geq_s$
$x \bowtie t$					
$-x \bowtie t$	$t \not\approx 0$	$t \not\approx \sim 0$	$t \not\approx \min_s$	$t \not\approx \max_s$	$\top$
$\sim x \bowtie t$					
$x + s \bowtie t$					

Table 5. Conditions for the invertibility for $x \bowtie t$ and bit-vector operators $\{-, \sim, +\}$ over inequality. The one for $+$ given modulo commutativity of $+$.

$\ell[x]$	$\leq_u$	$\geq_u$
$x \cdot s \bowtie t$	$\top$	$-s \mid s \geq_u t$
$x \bmod s \bowtie t$	$\top$	$\sim(-s) \geq_u t$
$s \bmod x \bowtie t$	$\top$	$(t + t - s) \mathbin{\&} s \geq_u t \lor t <_u s$
$x \div s \bowtie t$	$s \mid t \geq_u \sim(-s)$	$(s \cdot t) \div t \mathbin{\&} s \approx s$
$s \div x \bowtie t$	$0 <_u \sim s \mid t$	$\top$
$x \mathbin{\&} s \bowtie t$	$\top$	$s \geq_u t$
$x \mid s \bowtie t$	$t \geq_u s$	$\top$
$x \gg s \bowtie t$	$\top$	$(t \ll s) \gg s \approx t$
$s \gg x \bowtie t$	$\top$	$s \geq_u t$
$x \gg_a s \bowtie t$	$\top$	$\top$
$s \gg_a x \bowtie t$	$s <_u \min_s \lor t \geq_u s$	$s \geq_u \sim s \lor s \geq_u t$
$x \ll s \bowtie t$	$\top$	$\sim 0 \ll s \geq_u t$
$s \ll x \bowtie t$	$\top$	$\bigvee\limits_{i=0}^{\kappa(s)} (s \ll i) \geq_u t$
$x \circ s \bowtie t$	$t_x \approx 0 \Rightarrow s \leq_u t_s$	$t_x \approx \sim 0 \Rightarrow s \geq_u t_s$
	where $t_x = t[\kappa(t) - 1 : \kappa(t) - \kappa(x)], t_s = t[\kappa(s) - 1 : 0]$	
$s \circ x \bowtie t$	$s \leq_u t_s$	$s \geq_u t_s$
	where $t_x = t[\kappa(x) - 1 : 0], t_s = t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	

Table 6. Conditions for the invertibility of bit-vector operators over $\leq_u$ and $\geq_u$. Those for $\cdot$, $\&$ and $\mid$ are given modulo commutativity of those operators.

$\ell[x]$	$<_s$	$>_s$
$x \cdot s \bowtie t$	$\sim(-t) \mathbin{\&} (-s \mid s) <_s t$	$t <_s t - ((s \mid t) \mid -s)$
$x \bmod s \bowtie t$	$\sim t <_s (-s \mid -t)$	$(s >_s 0 \Rightarrow t <_s \sim(-s)) \wedge$
		$(s \leq_s 0 \Rightarrow t \not\approx \max_s) \wedge$
		$(t \not\approx 0 \vee s \not\approx 1)$
$s \bmod x \bowtie t$	$s <_s t \vee 0 <_s t$	$(s \geq_s 0 \Rightarrow s >_s t) \wedge$
		$(s <_s 0 \Rightarrow ((s-1) \gg 1) >_s t)$
$x \div s \bowtie t$	$t \leq_s 0 \Rightarrow \min_s \div s <_s t$	$\sim 0 \div s >_s t \vee \max_s \div s >_s t$
$s \div x \bowtie t$	$s <_s t \vee t \geq_s 0$	$\begin{cases} s >_s t & \text{for } \kappa(s) = 1 \\ (s \geq_s 0 \Rightarrow s >_s t) \wedge & \text{otherwise} \\ (s <_s 0 \Rightarrow s \gg 1 >_s t) \end{cases}$
$x \mathbin{\&} s \bowtie t$	$\sim(-t) \mathbin{\&} s <_s t$	$t <_s s \mathbin{\&} \max_s$
$x \mid s \bowtie t$	$\sim(s - t) \mid s <_s t$	$t <_s (s \mid \max_s)$
$s \mid x \bowtie t$		
$x \gg s \bowtie t$	$\sim(-t) \gg s <_s t$	$t <_s (\max_s \ll s) \gg s$
$s \gg x \bowtie t$	$s <_s t \vee 0 <_s t$	$(s <_s 0 \Rightarrow s \gg 1 >_s t) \wedge$
		$(s \geq_s 0 \Rightarrow s >_s t)$
$x \gg_a s \bowtie t$	$\min_s \gg_a s <_s t$	$t <_s \max_s \gg s$
$s \gg_a x \bowtie t$	$s <_s t \vee 0 <_s t$	$t <_s s \mathbin{\&} \max_s \wedge t <_s s \mid \max_s$
$x \ll s \bowtie t$	$(\min_s \gg s) \ll s <_s t$	$t <_s (\max_s \ll s) \mathbin{\&} \max_s$
$s \ll x \bowtie t$	$\min_s \ll s <_u t + \min_s$	$\displaystyle\bigvee_{i=0}^{\kappa(s)} (s \ll i) >_s t$
$x \circ s \bowtie t$	$t_x \approx \min_s \Rightarrow s <_u t_s$	$t_x \approx \max_s \Rightarrow s >_u t_s$
	where $t_x = t[\kappa(t) - 1 : \kappa(t) - \kappa(x)], t_s = t[\kappa(s) - 1 : 0]$	
$s \circ x \bowtie t$	$(s \leq_s t_s) \wedge (s \approx t_s \Rightarrow t_x \not\approx 0)$	$(s \geq_s t_s) \wedge (s \approx t_s \Rightarrow t_x \not\approx \sim 0)$
	where $t_x = t[\kappa(x) - 1 : 0], t_s = t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	

Table 7. Conditions for the invertibility of bit-vector operators over $<_s$ and $>_s$. Those for $\cdot$, $\&$ and $\mid$ are given modulo commutativity of those operators.

$\ell[x]$	$\leq_s$	$\geq_s$
$x \cdot s \bowtie t$	$\sim(s \approx 0 \land t <_s s)$	$(-s \mid s) \mathbin{\&} \max_s \geq_s t$
$x \bmod s \bowtie t$	$\sim 0 <_s -s \mathbin{\&} t$	$t <_s s \lor 0 \geq_s s$
$s \bmod x \bowtie t$	$t <_u \min_s \lor t \geq_s s$	$(s \geq_s 0 \Rightarrow s \geq_s t) \land$
		$((s <_s 0 \land t \geq_s 0) \Rightarrow s - t >_u t)$
$x \div s \bowtie t$	$((s \cdot t) \div s \approx t) \lor$	$(\sim 0 \div s \geq_s t) \lor (\max_s \div s \geq_s t)$
	$(t \leq_s 0 \Rightarrow \min_s \div s <_s t)$	
$s \div x \bowtie t$	$t \geq_s \sim 0 \lor t \geq_s s$	$(s \geq_s 0 \Rightarrow s \geq_s t) \land$
		$(s <_s 0 \Rightarrow s \gg 1 \geq_s t)$
$x \mathbin{\&} s \bowtie t$	$s \geq_u t \mathbin{\&} \min_s$	$s \mathbin{\&} t \approx t \lor t <_s (t - s) \mathbin{\&} s$
$x \mid s \bowtie t$	$t \geq_s s \mid \min_s$	$s \mathbin{\&} t$
$x \gg s \bowtie t$	$t \geq_s t \gg s$	$s \not\approx 0 \Rightarrow \ \sim 0 \gg s \geq_s t$
$s \gg x \bowtie t$	$t <_u \min_s \lor t \geq_s s$	$(s <_s 0 \Rightarrow s \gg 1 \geq_s t) \land$
		$(s \geq_s 0 \Rightarrow s \geq_s t)$
$x \gg_a s \bowtie t$	$t \geq_s \ \sim(\max_s \gg s)$	$\max_s \gg s \geq_s t$
$s \gg_a x \bowtie t$	$t \geq_s 0 \lor t \geq_s s$	$t \geq_u \ \sim t \lor s \geq_s t$
$x \ll s \bowtie t$	$t \gg (t \gg s) <_u \min_s$	$(\max_s \ll s) \mathbin{\&} \max_s \geq_s t$
$s \ll x \bowtie t$	$t \gg s <_u \min_s$	$\displaystyle\bigvee_{i=0}^{\kappa(s)} (s \ll i) \geq_s t$
$x \circ s \bowtie t$	$t_x \approx \min_s \Rightarrow s \leq_u t_s$	$t_x \approx \max_s \Rightarrow s \geq_u t_s$
	where $t_x = t[\kappa(t) - 1 : \kappa(t) - \kappa(x)], t_s = t[\kappa(s) - 1 : 0]$	
$s \circ x \bowtie t$	$s \leq_s t_s$	$s \geq_s t_s$
	where $t_x = t[\kappa(x) - 1 : 0], t_s = t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	

Table 8. Conditions for the invertibility of bit-vector operators over $\leq_s$ and $\geq_s$. Those for $\cdot$, $\&$ and $\mid$ are given modulo commutativity of those operators.

$s \bmod x \approx t$. A small number of synthesized solutions were only correct for a bit-width of 4, e.g, solution $(\sim s \ll s) \ll s <_s t$ for $x \div s <_s t$. In total, we found 6 width-dependent synthesized solutions, all of them for bit-vector operators $\div$ and $\bmod$. For those, we used the manually crafted invertibility conditions instead.

3.2 Verifying Invertibility Conditions

We verified the correctness of all 162 invertibility conditions for bit-widths from 1 to 65 by checking for each bit-width the T_{BV}-unsatisfiability of the formula $\neg(\phi_c \Leftrightarrow \exists x. \ell[x])$ where ℓ ranges over the literals in Tables 2–3 with s and t replaced by fresh constants, and ϕ_c is the corresponding invertibility condition.

In total, we generated 12,980 verification problems and used all participating solvers of the quantified bit-vector division of SMT-competition 2017 to verify them. For each solver/benchmark pair we used a CPU time limit of one hour and a memory limit of 8GB on the same machines as those mentioned in the previous section. We consider an invertibility condition to be verified for a certain bit-width if at least one of the solvers was able to report unsatisfiable for the corresponding formula within the given time limit. Out of the 12,980 instances, we were able to verify 12,277 (94.6%).

Overall, all verification tasks (including timeouts) required a total of 275 days of CPU time. The success rate of each individual solver was 91.4% for Boolector, 85.0% for CVC4, 50.8% for Q3B, and 92% for Z3. We observed that on 30.6% of the problems, Q3B exited with a Python exception without returning any result. For bit-vector operators $\{\sim, -, +, \&, |, \gg, \gg_a, \ll, \circ\}$, over all relations, and for operators $\{\cdot, \div, \bmod\}$ over relations $\{\not\approx, \leq_u, \leq_s\}$, we were able to verify all invertibility conditions for all bit-widths in the range 1–65. Interestingly, no solver was able to verify the invertibility conditions for $x \bmod s <_s t$ with a bit-width of 54 and $s \bmod x <_u t$ with bit-widths 35-37 within the allotted time. We attribute this to the underlying heuristics used by the SAT solvers in these systems. All other conditions for $<_s$ and $<_u$ were verified for all bit-vector operators up to bit-width 65. The remaining conditions for operators $\{\cdot, \div, \bmod\}$ over relations $\{\approx, >_u, \geq_u, >_s, \geq_s\}$ were verified up to at least a bit-width of 14. We discovered 3 conditions for $s \div x \bowtie t$ with $\bowtie \in \{\not\approx, >_s, \geq_s\}$ that were not correct for a bit-width of 1. For each of these cases, we added an additional invertibility condition that correctly handles that case.

We leave to future work the task of formally proving that our invertibility conditions are correct for all bit-widths. Since this will most likely require the development of an interactive proof, we could leverage some recent work by Ekici et al. [9] that includes a formalization in the Coq proof assistant of the SMT-LIB theory of bit-vectors.

4 Counterexample-Guided Instantiation for Bit-Vectors

In this section, we leverage techniques from the previous section for constructing symbolic solutions to bit-vector constraints to define a novel instantiation-based technique for quantified bit-vector formulas. At a high level, we use a *counterexample-guided* approach for quantifier instantiation that adds new instances to a set of quantifier-free

```
CEGQI_S(∃y∀x. ψ[y, x])
    Γ := ∅
    Repeat:
        1. If Γ is T-unsatisfiable, then return "unsat".
        2. Otherwise, if Γ' = Γ ∪ {¬ψ[y, x]} is T-unsatisfiable, then return "sat".
        3. Otherwise, let I be a model of T and Γ' and t = S(x, ψ, I, Γ). Γ := Γ∪{ψ[y, t]}.
```

Fig. 2. A counterexample-guided quantifier instantiation procedure CEGQI_S, parameterized by a selection function S, for determining the T-satisfiability of $\exists y \forall x.\, \psi[y, x]$ with ψ quantifier-free and $FV(\psi) = y \cup x$.

clauses based on models for the negated input formula. The procedure terminates if it finds a set of instances that is unsatisfiable or entails the negation of the input formula.

We use a counterexample-guided approach for quantifier instantiation, as given by procedure CEGQI_S in Figure 2. To simplify the exposition here, we focus on input problems expressed as a single formula in prenex normal form and with up to one quantifier alternation. We stress, though, that the approach applies in general to arbitrary sets of quantified formulas in some Σ-theory T with a decidable quantifier-free fragment. The procedure checks via instantiation the T-satisfiability of a quantified input formula φ of the form $\exists y \forall x.\, \psi[x, y]$ where ψ is quantifier-free and x and y are possibly empty sequences of variables. It maintains an evolving set Γ, initially empty, of quantifier-free instances of the input formula. During each iteration of the procedure's loop, there are three possible cases: 1) if Γ is T-unsatisfiable, the input formula φ is also T-unsatisfiable and "unsat" is returned; 2) if Γ is T-satisfiable but not together with with $\neg\psi[y, x]$, the negated body of φ, then Γ entails φ in T, hence φ is T-satisfiable and "sat" is returned. 3) If neither of previous cases holds, the procedure adds to Γ an instance of ψ obtained by replacing the variables x with some terms t, and continues. The procedure CEGQI is parametrized by a *selection function S* that generates the terms t.

Definition 4. *(Selection Function) A* selection function *takes as input a tuple of variables x, a model I of T, a quantifier-free Σ-formula $\psi[x]$, and a set Γ of Σ-formulas such that $x \notin FV(\Gamma)$ and $I \models \Gamma \cup \{\neg\psi\}$. It returns a tuple of ε-valid terms t of the same type as x such that $FV(t) \subseteq FV(\psi) \setminus x$.*

Definition 5. *Let $\psi[x]$ be a quantifier-free Σ-formula. A selection function is:*

1. *Finite for x and ψ if there is a finite set S^* such that $S(x, \psi, I, \Gamma) \in S^*$ for all legal inputs I and Γ.*
2. *Monotonic for x and ψ if for all legal inputs I and Γ, $S(x, \psi, I, \Gamma) = t$ only if $\psi[t] \notin \Gamma$.*

Procedure CEGQI_S is refutation-sound and model-sound for any selection function S, and terminating for selection functions that are finite and monotonic.

Theorem 6 (Correctness of CEGQI_S). *Let S be a selection function and let $\varphi = \exists y \forall x.\, \psi[y, x]$ be a legal input for CEGQI_S. Then the following hold.*

1. If $\mathsf{CEGQI}_\mathcal{S}(\varphi)$ *returns "unsat", then* φ *is* T*-unsatisfiable.*
2. If $\mathsf{CEGQI}_\mathcal{S}(\varphi)$ *returns "sat" for some final* Γ*, then* φ *is* T*-equivalent to* $\exists \boldsymbol{y}. \bigwedge_{\gamma \in \Gamma} \gamma.$
3. *If* $\mathcal{S}$ *is finite and monotonic for* $\boldsymbol{x}$ *and* ψ*, then* $\mathsf{CEGQI}_\mathcal{S}(\varphi)$ *terminates.*

Proof. We show each part of the theorem below. Note that by the definition of $\mathsf{CEGQI}_\mathcal{S}$ and since $\mathcal{S}$ is a selection function, all inputs $\mathcal{I}$ and Γ given to $\mathcal{S}$ in the loop of this function are legal inputs. Also note that for the first two parts, we have that $\mathsf{CEGQI}_\mathcal{S}(\varphi)$ terminates in a state where Γ is a set of formulas of the form $\psi[\boldsymbol{y}, \boldsymbol{t}]$ where, since $\mathcal{S}$ is a selection function, $\boldsymbol{t}$ is a tuple of ε-valid terms and $FV(\boldsymbol{t}) \subseteq \boldsymbol{y}$.

Part 1) By definition of $\mathsf{CEGQI}_\mathcal{S}$, if $\mathsf{CEGQI}_\mathcal{S}(\varphi)$ returns "unsat", then Γ is T-unsatisfiable. By the definition of $\mathsf{CEGQI}_\mathcal{S}$, we have that Γ is a set of formulas of the form $\psi[\boldsymbol{y}, \boldsymbol{t}]$. For each $\psi[\boldsymbol{y}, \boldsymbol{t}] \in \Gamma$, we have that $\boldsymbol{t}$ is ε-valid since $\mathcal{S}$ is a selection function, and hence $\forall \boldsymbol{x} \psi[\boldsymbol{y}, \boldsymbol{x}] \Rightarrow \psi[\boldsymbol{y}, \boldsymbol{t}]$ holds in all models of T. Since Γ is T-unsatisfiable, $\forall \boldsymbol{x} \psi[\boldsymbol{y}, \boldsymbol{x}]$ is T-unsatisfiable, and hence φ is as well.

Part 2) By definition of $\mathsf{CEGQI}_\mathcal{S}$, if $\mathsf{CEGQI}_\mathcal{S}(\varphi)$ returns "sat", the Γ is T-satisfiable and $\Gamma' = \Gamma \cup \{\neg \psi[\boldsymbol{y}, \boldsymbol{x}]\}$ is T-unsatisfiable. For each $\psi[\boldsymbol{y}, \boldsymbol{t}] \in \Gamma$, we have that $FV(\boldsymbol{t}) \subseteq \boldsymbol{y}$ and hence $FV(\Gamma) \subseteq \boldsymbol{y}$. Since $\boldsymbol{x} \notin FV(\Gamma)$, we have that $\Gamma \cup \{\exists \boldsymbol{x}. \neg \psi[\boldsymbol{y}, \boldsymbol{x}]\}$ is also T-unsatisfiable. Let $\mathcal{I}$ be an arbitrary model of Γ. Since Γ' is unsatisfiable, it must be that $\mathcal{I} \not\models \exists \boldsymbol{x}. \neg \psi[\boldsymbol{y}, \boldsymbol{x}]$ and hence $\mathcal{I} \models \forall \boldsymbol{x}. \psi[\boldsymbol{y}, \boldsymbol{x}]$. Thus $\Gamma \Rightarrow \exists \boldsymbol{x}. \neg \psi[\boldsymbol{y}, \boldsymbol{x}]$ and hence $\exists \boldsymbol{y}. \Gamma \Rightarrow \varphi$ holds in all models of T. By the same reasoning as Part 1, we have that $\varphi \Rightarrow \exists \boldsymbol{y}. \Gamma$ holds in all models as well. Thus, φ is equivalent to $\exists \boldsymbol{y}. \Gamma$.

Part 3) Assume $\mathcal{S}$ is monotonic and finite for $\psi[\boldsymbol{y}, \boldsymbol{x}]$. Since it is finite, let $\mathcal{S}^*$ be a finite set such that $\mathcal{S}(\boldsymbol{x}, \psi, \mathcal{I}, \Gamma) \in \mathcal{S}^*$ for all valid inputs $\mathcal{I}, \Gamma$. Since it is monotonic, each iteration of the loop adds a new formula from $\mathcal{S}^*$ to Γ. Since $\mathcal{S}^*$ is finite, the number of iterations of this loop is bounded by the size of $\mathcal{S}^*$. Hence, $\mathsf{CEGQI}_\mathcal{S}(\varphi)$ terminates. $\qquad\square$
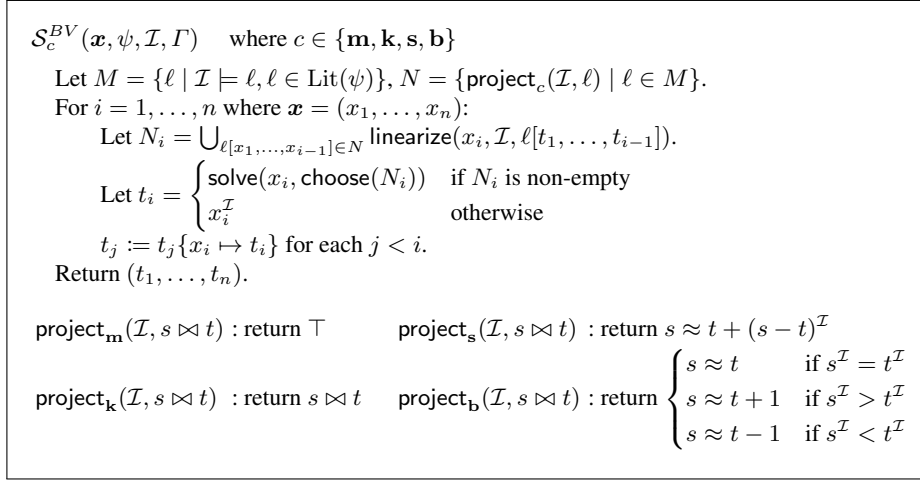
Thanks to this theorem, to define a T-satisfiability procedure for quantified Σ-formulas, it suffices to define a selection function satisfying the criteria of Definition 4. We do that in the following section for T_{BV}.

4.1 Selection functions for bit-vectors

In Figure 3, we define a (class of) selection functions $\mathcal{S}_c^{BV}$ for quantifier-free bit-vector formulas, which is parameterized by a *configuration* c, a value of the enumeration type $\{\mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}\}$. The selection function collects in the set M all the literals occurring in Γ' that are satisfied by $\mathcal{I}$. Then, it collects in the set N a *projected form* of each literal in M. This form is computed by the function $\mathsf{project}_c$ parameterized by configuration c. That function transforms its input literal into a form suitable for function solve from Figure 1. We discuss the intuition for projection operations in more detail below.

Example 7. Consider the Σ_{BV}-literal $a \geq_u b$ and the interpretation $\mathcal{I}$ where $a^\mathcal{I} = 5$ and $b^\mathcal{I} = 3$. We have that $\mathsf{project}_c$ returns $\top$ for $c = \mathbf{m}$, $a \geq_u b$ for $c = \mathbf{k}$, $a \approx b + (5 - 3)$ for $c = \mathbf{s}$, and $a \approx b + 1$ for $c = \mathbf{b}$. $\qquad\triangle$

After constructing set N, the selection function computes a term t_i for each variable x_i in tuple $\boldsymbol{x}$, which we call the *solved form* of x_i. To do that, it first constructs a set

15

$$\mathcal{S}_c^{BV}(\boldsymbol{x}, \psi, \mathcal{I}, \Gamma) \quad \text{where } c \in \{\mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}\}$$

Let $M = \{\ell \mid \mathcal{I} \models \ell, \ell \in \mathrm{Lit}(\psi)\}$, $N = \{\mathsf{project}_c(\mathcal{I}, \ell) \mid \ell \in M\}$.
For $i = 1, \ldots, n$ where $\boldsymbol{x} = (x_1, \ldots, x_n)$:

    Let $N_i = \bigcup_{\ell[x_1, \ldots, x_{i-1}] \in N} \mathsf{linearize}(x_i, \mathcal{I}, \ell[t_1, \ldots, t_{i-1}])$.

    Let $t_i = \begin{cases} \mathsf{solve}(x_i, \mathsf{choose}(N_i)) & \text{if } N_i \text{ is non-empty} \\ x_i^{\mathcal{I}} & \text{otherwise} \end{cases}$

    $t_j \coloneqq t_j\{x_i \mapsto t_i\}$ for each $j < i$.
Return $(t_1, \ldots, t_n)$.

$\mathsf{project}_{\mathbf{m}}(\mathcal{I}, s \bowtie t) : \text{return } \top$ $\mathsf{project}_{\mathbf{s}}(\mathcal{I}, s \bowtie t) : \text{return } s \approx t + (s - t)^{\mathcal{I}}$

$\mathsf{project}_{\mathbf{k}}(\mathcal{I}, s \bowtie t) : \text{return } s \bowtie t$ $\mathsf{project}_{\mathbf{b}}(\mathcal{I}, s \bowtie t) : \text{return } \begin{cases} s \approx t & \text{if } s^{\mathcal{I}} = t^{\mathcal{I}} \\ s \approx t + 1 & \text{if } s^{\mathcal{I}} > t^{\mathcal{I}} \\ s \approx t - 1 & \text{if } s^{\mathcal{I}} < t^{\mathcal{I}} \end{cases}$

Fig. 3. Selection functions $\mathcal{S}_c^{BV}$ for quantifier-free bit-vector formulas. The procedure is parameterized by a configuration c, one of either $\mathbf{m}$ (model value), $\mathbf{k}$ (keep), $\mathbf{s}$ (slack), or $\mathbf{b}$ (boundary).

of literals N_i all linear in x_i. It considers literals ℓ from N and replaces all previously solved variables $x_1, \ldots, x_{i-1}$ by their respective solved forms to obtain the literal $\ell' = \ell[t_1, \ldots, t_{i-1}]$. It then calls function linearize on literal ℓ' which returns a *set* of literals, each obtained by replacing all but one occurrence of x_i in ℓ with the value of x_i in $\mathcal{I}$.[4]

Example 8. Consider an interpretation $\mathcal{I}$ where $x^{\mathcal{I}} = 1$, and Σ_{BV}-terms a and b with $x \notin FV(a) \cup FV(b)$. We have that $\mathsf{linearize}(x, \mathcal{I}, x \cdot (x + a) \approx b)$ returns the set $\{1 \cdot (x + a) \approx b, x \cdot (1 + a) \approx b\}$; $\mathsf{linearize}(x, \mathcal{I}, x \geq_u a)$ returns the singleton set $\{x \geq_u a\}$; $\mathsf{linearize}(x, \mathcal{I}, a \not\approx b)$ returns the empty set. $\triangle$

If the set N_i is non-empty, the selection function heuristically chooses a literal from N_i (indicated in Figure 3 with $\mathsf{choose}(N_i)$). It then computes a solved form t_i for x_i by solving the chosen literal for x_i with the function solve described in the previous section. If N_i is empty, we let t_i is simply the value of x_i in the given model $\mathcal{I}$. After that, x_i is eliminated from all the previous terms $t_1, \ldots, t_{i-1}$ by replacing it with t_i. After processing all n variables of $\boldsymbol{x}$, the tuple $(t_1, \ldots, t_n)$ is returned.

The configurations of selection function $\mathcal{S}_c^{BV}$ determine how literals in M are modified by the $\mathsf{project}_c$ function prior to computing solved forms, based on the current model $\mathcal{I}$. With the *model value* configuration $\mathbf{m}$, the selection function effective ignores the structure of all literals in M and (because the set N_i is empty) ends up choosing the value $x_i^{\mathcal{I}}$ as the solved form variable x_i, for each i. On the other end of the spectrum, the configuration $\mathbf{k}$ *keeps* all literals in M unchanged. The remaining two configurations have an effect on how disequalities and inequalities are handled by $\mathsf{project}_c$. With configuration $\mathbf{s}$ $\mathsf{project}_c$ normalizes any kind of literal (equality, inequality or disequality) $s \bowtie t$ to an equality by adding the *slack* value $(s - t)^{\mathcal{I}}$ to t. With configuration

---

[4] This is a simple heuristic to generate literals that can be solved for x_i. More elaborate heuristics could be used in practice.

b it maps equalities to themselves and inequalities and disequalities to an equality corresponding to a *boundary point* of the relation between s and t based on the current model. Specifically, it adds one to t if s is greater than t in $\mathcal{I}$, it subtracts one if s is smaller than t, and returns $s \approx t$ if their value is the same. These two configurations are inspired by quantifier elimination techniques for linear arithmetic [5,15]. In the following, we provide an end-to-end example of our technique for quantifier instantiation that makes use of selection function $\mathcal{S}_c^{BV}$.

Example 9. Consider formula $\varphi = \exists \boldsymbol{y}. \forall x_1. (x_1 \cdot a \leq_u b)$ where a and b are terms with no free occurrences of x_1. To determine the satisfiability of φ, we invoke $\mathsf{CEGQI}_{\mathcal{S}_c^{BV}}$ on φ for some configuration c. Say that in the first iteration of the loop, we find that $\Gamma' = \Gamma \cup \{x_1 \cdot a >_u b\}$ is satisfied by some model $\mathcal{I}$ of T_{BV} such that $x_1^{\mathcal{I}} = 1$, $a^{\mathcal{I}} = 1$, and $b^{\mathcal{I}} = 0$. We invoke $\mathcal{S}_c^{BV}((x_1), \mathcal{I}, \Gamma')$ and first compute $M = \{x_1 \cdot a >_u b\}$, the set of literals of Γ' that are satisfied by $\mathcal{I}$. The table below summarizes the values of the internal variables of $\mathcal{S}_c^{BV}$ for the various configurations:

config	N_1	t_1
m	$\emptyset$	1
k	$\{x_1 \cdot a >_u b\}$	$\varepsilon z. (a <_u -b \mid b) \Rightarrow z \cdot a >_u b$
s, b	$\{x_1 \cdot a \approx b + 1\}$	$\varepsilon z. ((-a \mid a) \,\&\, b + 1 \approx b + 1) \Rightarrow z \cdot a \approx b + 1$

In each case, $\mathcal{S}_c^{BV}$ returns the tuple (t_1), and we add the instance $t_1 \cdot a \leq_u b$ to Γ. Consider configuration **k** where t_1 is the choice expression $\varepsilon z. ((a <_u -b \mid b) \Rightarrow z \cdot a >_u b)$. Since t_1 is ε-valid, due to the semantics of ε, this instance is equivalent to:

$$((a <_u -b \mid b) \Rightarrow k \cdot a >_u b) \wedge k \cdot a \leq_u b \tag{1}$$

for fresh variable k. This formula is T_{BV}-satisfiable if and only if $\neg(a <_u -b \mid b)$ is T_{BV}-satisfiable. In the second iteration of the loop in $\mathsf{CEGQI}_{\mathcal{S}_c^{BV}}$, set Γ contains formula (1) above. We have two possible outcomes:

   $i)$ $\neg(a <_u -b \mid b)$ is T_{BV}-unsatisfiable. Then (1) and hence Γ are T_{BV}-unsatisfiable, and the procedure terminates with "unsat".

   $ii)$ $\neg(a <_u -b \mid b)$ is satisfied by some model $\mathcal{J}$ of T_{BV}. Then $\exists z. z \cdot a \leq_u b$ is false in $\mathcal{J}$ since the invertibility condition of $z \cdot a \leq_u b$ is false in $\mathcal{J}$. Hence, $\Gamma' = \Gamma \cup \{x_1 \cdot a >_u b\}$ is unsatisfiable, and the algorithm terminates with "sat".

   In fact, we argue later that quantified bit-vector formulas like φ above, which contain only one occurrence of a universal variable, require at most one instantiation before $\mathsf{CEGQI}_{\mathcal{S}_k^{BV}}$ terminates. The same guarantee does not hold with the other configurations. In particular, configuration **m** generates the instantiation where t_1 is 1, which simplifies to $a \leq_u b$. This may not be sufficient to show that Γ or Γ' is unsatisfiable in the second iteration of the loop and the algorithm may resort to *enumerating* a repeating pattern of instantiations, such as $x_1 \mapsto 1, 2, 3, \ldots$ and so on. This obviously does not scale for problems with large bit-widths. $\triangle$

Example 10. As the last example demonstrates, $\mathsf{CEGQI}_{\mathcal{S}_k^{BV}}$ may terminate with at most one instance for input formulas whose body has just one literal and a single occurrence of each universal variable. However, consider extending the quantified formula from

17

the previous example to a disjunction of two literals: $\exists \boldsymbol{y} \forall x_1. (x_1 \cdot a \leq_u b \vee \ell[x_1])$. Assume that our selection function chooses the same t_1 as in the previous example. The corresponding instance is equivalent to:

$$((a <_u -b \mid b) \Rightarrow k \cdot a >_u b) \wedge (k \cdot a \leq_u b \vee \ell[k]) \tag{2}$$

In contrast to the previous example, the second iteration of the loop from Figure 2 is not guaranteed to terminate for this example. The above formula may be satisfied by a model $\mathcal{J}$ where $k \cdot a >_u b$ and $\ell[k]$ hold. Notice that $\mathcal{J}$ may also satisfy $a <_u -b \mid b$, meaning it may still be the case that $x_1 \cdot a \leq_u b$ together with the above instance is satisfied by $\mathcal{J}$. In such a case, we may invoke $\mathsf{CEGQI}_{\mathcal{S}_k^{BV}}$ again, which may produce the same solved form for x_1 if it constructs a solved form for x_1 again based on the literal $x_1 \cdot a \leq_u b$. Hence, by the terminology from Definition 5, the selection function $\mathcal{S}_{\mathbf{k}}^{BV}$ is not monotonic for quantified formulas with more than one occurrence of a universal variable. $\triangle$

Similarly, if the literals of the input formula have multiple occurrences of x_1, then multiple instances may be returned by the selection function since the literals returned by linearize in Figure 3 depend on the model value of x_1, and hence more than one possible instance may be considered in loop in Figure 2.

The following theorem summarizes the properties of our selection functions. In the following, we say a quantified formula is *unit linear invertible* if it is of the form $\forall x.\ell[x]$ where ℓ is linear in x and has an invertibility condition for x. We say a selection function is *n-finite* for a quantified formula ψ if the number of possible instantiations it returns is at most n for some positive integer n.

Theorem 11. *Let $\psi[\boldsymbol{x}]$ be a quantifier-free formula in the signature of T_{BV}.*

1. $\mathcal{S}_c^{BV}$ *is a finite selection function for $\boldsymbol{x}$ and ψ for all $c \in \{\mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}\}$.*
2. $\mathcal{S}_{\mathbf{m}}^{BV}$ *is monotonic.*
3. $\mathcal{S}_{\mathbf{k}}^{BV}$ *is 1-finite if ψ is unit linear invertible.*
4. $\mathcal{S}_{\mathbf{k}}^{BV}$ *is monotonic if ψ is unit linear invertible.*

Proof. Let $\boldsymbol{x} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})$ be a tuple of variables and let φ be a quantifier-free T_{BV}-formula. We show each part for the case where $n = 1$; the arguments below can be lifted to $n > 1$ in a straightforward way. Let Γ be a set of formulas such that $x_1 \notin FV(\Gamma)$, let $\mathcal{I}$ be a model of T_{BV} such that $\mathcal{I} \models \Gamma \cup \{\neg\varphi\}$, and let t_1 be $\mathcal{S}_c^{BV}((x_1), \mathcal{I}, \Gamma)$.

Part 1) To show that $\mathcal{S}_c^{BV}$ is a selection function, we must show that t_1 is ε-valid and $FV(t_1) \subseteq FV(\varphi) \setminus \{x_1\}$. Notice that for all configurations, the value (t_1) returned by $\mathcal{S}_c^{BV}$ is either of the form $x_1^{\mathcal{I}}$ or $\mathsf{solve}(x_1, \ell)$ for some $\ell \in \mathrm{Lit}(\Gamma)$. In the former case, we have that t_1 is clearly ε-valid and $FV(t_1) = \emptyset$. In the latter case, as a consequence of Theorem 3, we have that t_1 is ε-valid and $FV(t_1) \subseteq FV(\ell)$. We have that $\ell \in \mathrm{Lit}(\varphi)$ and thus $FV(\ell) \subseteq FV(\varphi)$. Thus, in either case, we have $FV(t_1) \subseteq FV(\varphi) \setminus \{x_1\}$. Hence, $\mathcal{S}_c^{BV}$ is a selection function for $c = \mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}$. To show these selection functions are finite for φ, note that the number of terms of form $x_1^{\mathcal{I}}$ is finite. Also note that the number of literals in $\mathrm{Lit}(\varphi)$ is finite. For each c and $\ell \in \mathrm{Lit}(\varphi)$, set of literals of the

form $\text{project}_c(\mathcal{I}, \ell)$, call this set N, is finite. For each literal $\ell' \in N$, the set of literals returned by $\text{linearize}(x_i, \mathcal{I}, \ell')$, call this set N', is also finite. Since it is either the case that $t_1 = \text{solve}(x_1, \ell'')$ for some $\ell'' \in N'$ or $t_1 = x_1^{\mathcal{I}}$, the number of possible return values of $\mathcal{S}_c^{BV}$ is finite for (x_1) and φ for $c = \mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}$.

Part 2) Since $x_1 \notin FV(\ell)$ for any literal returned by $\text{project}_{\mathbf{m}}(\mathcal{I}, \ell)$, it must be the case that $t_1 = x_1^{\mathcal{I}}$. Assume that $\text{project}_{\mathbf{m}}$ was not monotonic for (x_1) and $\varphi = \neg\psi[x_1]$. Since $\text{project}_{\mathbf{m}}$ is a selection function by Part 1 and $\mathcal{I}, \Gamma$ is a legal input to $\text{project}_{\mathbf{m}}$, then without loss of generality we may assume that $\psi[t_1] \in \Gamma$. However, $\neg\psi[x_1] \in \Gamma$, $\mathcal{I} \models \Gamma$ and since $t_1^{\mathcal{I}} = x_1^{\mathcal{I}}$, it must instead be the case that $\varphi[t_1] \notin \Gamma$. Hence, $\text{project}_{\mathbf{m}}$ is monotonic for φ.

Part 3) Assume φ is the unit linear invertible formula ℓ. By definition of $\text{project}_{\mathbf{k}}$, linearize, and since by definition ℓ is linear with respect to x_1 and $\mathcal{I} \models \neg\ell$, we have that t_1 must be the term returned by $\text{solve}(x_1, \neg\ell)$. Hence, $\mathcal{S}_{\mathbf{k}}^{BV}$ has only one possible return value and hence is 1-finite.

Part 4) Assume φ is a unit linear invertible formula $\ell[x_1]$. The return value of $\mathcal{S}_{\mathbf{k}}^{BV}$ is the tuple (t_1), where by the reasoning in Part 3, we have that t_1 is the term returned by $\text{solve}(x_1, \neg\ell[x_1])$. Thus by Theorem 3, we have that $\neg\ell[t_1] \Leftrightarrow \exists z. \neg\ell[z]$ holds in all models of T_{BV}. Now, assume that $\text{project}_{\mathbf{k}}$ was not monotonic for (x_1) and $\ell[x_1]$. Since $\text{project}_{\mathbf{k}}$ is a selection function by Part 1 and $\mathcal{I}, \Gamma$ is a legal input to $\text{project}_{\mathbf{k}}$, then without loss of generality we may assume that $\ell[t_1] \in \Gamma$. Since $\mathcal{I} \models \Gamma$, and since $\neg\ell[t_1] \Leftarrow \exists z. \neg\ell[z]$ holds in all models of T_{BV}, we have that $\mathcal{I}$ must satisfy $\neg\exists z. \neg\ell[z]$, which is $\forall z. \ell[z]$. However, we also have that by assumption $\neg\ell[x_1] \in \Gamma$. Hence, it must instead be the case that $\varphi[t_1] \notin \Gamma$ and thus $\text{project}_{\mathbf{k}}$ is monotonic for φ. $\qquad\square$

This theorem implies that counterexample-guided instantiation using configuration $\mathcal{S}_{\mathbf{m}}^{BV}$ is a decision procedure for quantified bit-vectors. However, in practice the worst-case number of instances considered by this configuration for a variable $x_{[n]}$ is proportional to the number of its possible values (2^n), which is practically infeasible for sufficiently large n. More interestingly, counterexample-guided instantiation using $\mathcal{S}_{\mathbf{k}}^{BV}$ is a decision procedure for quantified formulas that are unit linear invertible, and moreover has the guarantee that at most one instantiation is returned by this selection function. Hence, formulas in this fragment can be effectively reduced to quantifier-free bit-vector constraints in at most two iterations of the loop of procedure $\text{CEGQI}_{\mathcal{S}}$ in Figure 2.

4.2 Implementation

We implemented the new instantiation techniques described in this section as an extension of CVC4, which is a $\text{DPLL}(T)$-based SMT solver [19] that supports quantifier-free bit-vector constraints, (arbitrarily nested) quantified formulas, and support for choice expressions. For the latter, all choice expressions $\varepsilon x. \varphi[x]$ are eliminated from assertions by replacing them with a fresh variable k of the same type and adding $\varphi[k]$ as a new assertion, which notice is sound since all choice expressions we consider are ε-valid. In the remainder of the paper, we will refer to our extension of the solver as **cegqi**. In the following, we discuss important implementation details of the extension.

Handling Duplicate Instantiations The selection functions $\mathcal{S}_{\mathbf{s}}^{BV}$ and $\mathcal{S}_{\mathbf{b}}^{BV}$ are not guaranteed to be monotonic, neither is $\mathcal{S}_{\mathbf{k}}^{BV}$ for quantified formulas that contain more than

one occurrence of universal variables. Hence, when applying these strategies to arbitrary quantified formulas, we use a two-tiered strategy that invokes $\mathcal{S}_{\mathbf{m}}^{BV}$ as a second resort if the instance for the terms returned by a selection function already exists in Γ.

Linearizing Rewrites Our selection function in Figure 3 uses the function linearize to compute literals that are linear in the variable x_i to solve for. The way we presently implement linearize makes those literals dependent on the value of x_i in the current model $\mathcal{I}$, with the risk of overfitting to that model. To address this limitation, we use a set of equivalence-preserving rewrite rules whose goal is to reduce the number of occurrences of x_i to one when possible, by applying basic algebraic manipulations. As a trivial example, a literal like $x_i + x_i \approx a$ is rewritten first to $2 \cdot x_i \approx a$ which is linear in x_i if a does not contain x_i. In that case, this literal, and so the original one, has an invertibility condition as discussed in Section 3.

Variable Elimination We use procedure solve from Section 3 not only for selecting quantifier instantiations, but also for eliminating variables from quantified formulas. In particular, for a quantified formula of the form $\forall x\boldsymbol{y}. \ell \Rightarrow \varphi[x, \boldsymbol{y}]$, if ℓ is linear in x and solve(x, ℓ) returns a term s containing no ε-expressions, we can replace this formula by $\forall \boldsymbol{y}. \varphi[s, \boldsymbol{y}]$. When ℓ is an equality, this is sometimes called destructive equality resolution (DER) and is an important implementation-level optimization in state-of-the-art bit-vector solvers [24]. In our approach, to increase the likelihood that solve returns a term that contains no ε-expressions, we include several optimizations that determine when it can be determined that ℓ has a unique solution for x. A common example is an equality that involves multiplication by an odd constant, i.e. $x \cdot c \approx t$ where c is an odd constant. The only solution for x in this case is $c^{-1} \cdot t$ where c^{-1} denotes the (unique) multiplicative inverse of c modulo the bit-width of the type of c, which can be computed by Euclid's algorithm.

Handling Extract Consider formula $\forall x_{[32]}. (x[31 : 16] \not\approx a_{[16]} \lor x[15 : 0] \not\approx b_{[16]})$. Since all invertibility conditions for the extract operator are $\top$, rather than producing choice expressions we have found it more effective to eliminate extracts via rewriting. As a consequence, we independently solve constraints for *regions* of quantified variables when they appear underneath applications of extract operations. In this example, we let the solved form of x be $y_{[16]} \circ z_{[16]}$ where y and z are fresh variables, and subsequently solve for these variables in $y \approx a$ and $z \approx b$. Hence, we may instantiate x with $a \circ b$, a term that we would not have found by considering the two literals independently in the negated body of the formula above.

Handling Propositional Structure and Nested Quantifiers Notice that Figure 2 describes counterexample-guided quantifier instantiation for an input formula with one level of quantifier alternation. In practice, our technique can be used for problems containing more than one level of quantifier alternation and that are not in prenex normal form. A thoroughout description of this technique is beyond the scope of the paper; we provide some high level details here. Recall that in the DPLL(T) setting, the SMT solver incrementally builds a truth assignment in the form of a set of literals, with the goal of finding a set that propositionally satisfies all clauses in Γ and is consistent with respect to the background theory. In this setting, we consider all quantified formulas $\forall \boldsymbol{x}.\varphi[\boldsymbol{x}]$ in the current set M. For each of these formulas, we may add clauses of three

forms: *instantiation lemmas* of the form $(\neg A \vee \varphi[\boldsymbol{t}])$, *Skolemization lemmas* of the form $(\neg B \vee \neg\varphi[\boldsymbol{k}])$ where $\boldsymbol{k}$ are fresh constants of the same type as $\boldsymbol{x}$, and the connecting clauses $(\forall\boldsymbol{x}.\varphi[\boldsymbol{x}] \Rightarrow A)$ and $(\neg\forall\boldsymbol{x}.\varphi[\boldsymbol{x}] \Rightarrow B)$ Here, A and B are fresh Boolean constants which we call the *positive* and *negative* guard of $\forall\boldsymbol{x}.\varphi[\boldsymbol{x}]$ respectively. The second and third clauses are added once at the time when the quantified formula first occurs in an assignment M. We detect when the negation of a quantified formula along with the current set of clauses Γ is unsatisfiable by checking which negative guards B must be assigned to false. In practice, this is determined by a decision heuristic which insists that negative guards must be decided with positive polarity first. If a quantified formula and its corresponding negative guard are both asserted true, then we add an instantiation lemma to Γ based on the selection function from Figure 3. We terminate as usual when the set Γ is unsatisfiable, or we find a consistent satisfying assignment where no quantified formula and its negative guard are both asserted. This scheme allows the SMT solver to handle multiple quantified formulas simultaneously, as well as handling quantified formulas with arbitrary nesting. Above, notice that $\varphi[\boldsymbol{k}]$ may contain quantifiers, which are recursively handled by introducing instantiation and Skolemization lemmas for quantified formulas that appear in subsequent satisfying assignments.

Negating the Input Formula Our version of counterexample-guided quantifier instantiation is most effective for input formulas that are closed and universal. Thus, when an input formula is of the form $\exists\boldsymbol{x}.\,\varphi$ where $\boldsymbol{x}$ is non-empty and φ may contain quantifiers, we consider its negation $\forall\boldsymbol{x}.\,\neg\varphi$ instead. The latter formula may be significantly easier to solve since our quantifier instantiation techniques may find an instantiation for $\boldsymbol{x}$ that quickly leads to a proof of unsatisfiability, whereas instantiating $\boldsymbol{x}$ is not possible for the former. Since the theory of bit-vectors is a complete theory, it follows that the original formula is satisfiable if and only if this formula is unsatisfiable.

Rewrite Rules for Quantifier-Free Constraints Finally, we have found that in a DPLL(T)-based SMT solver, the quantifier-free bit-vector solver is often the bottleneck when solving quantified bit-vector constraints. For this reason, we use aggressive rewriting techniques for quantifier-free bit-vector constraints with the goal of replacing constraints with expensive propositional encodings with those with simpler encodings.

5 Evaluation

We implemented our techniques in the solver **cegqi** and considered four configurations **cegqi$_c$**, where **c** is one of $\{\mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}\}$, corresponding to the four selection function configurations described in Section 4. Out of these four configurations, **cegqi$_m$** is the only one that does not employ our new techniques but uses only model values for instantiation. It can thus be considered our base configuration. All configurations enable the optimizations described in Section 4.2 when applicable. We compared them against all entrants of the quantified bit-vector division of the 2017 SMT competition SMT-COMP: Boolector [16], CVC4 [2], Q3B [14] and Z3 [6]. With the exception of Q3B, all solvers are related to our approach since they are instantiation-based. However, none of these solvers utilizes invertibility conditions when constructing instantiations. We ran all experiments on the StarExec logic solving service [23] with a 300 second CPU and wall clock time limit and 100 GB memory limit.

unsat	Boolector	CVC4	Q3B	Z3	$\mathbf{cegqi_m}$	$\mathbf{cegqi_k}$	$\mathbf{cegqi_s}$	$\mathbf{cegqi_b}$
h-uauto	14	12	93	24	10	103	105	**106**
keymaera	3917	3790	3781	**3923**	3803	3798	3888	3918
psyco	**62**	**62**	49	**62**	**62**	39	**62**	61
scholl	57	36	13	**67**	36	27	36	35
tptp	55	52	**56**	**56**	**56**	**56**	**56**	**56**
uauto	**137**	72	131	**137**	72	72	135	**137**
ws-fixpoint	74	71	**75**	74	**75**	74	**75**	**75**
ws-ranking	16	8	18	**19**	15	11	12	11
Total unsat	4332	4103	4216	4362	4129	4180	4369	**4399**
sat	**Boolector**	**CVC4**	**Q3B**	**Z3**	$\mathbf{cegqi_m}$	$\mathbf{cegqi_k}$	$\mathbf{cegqi_s}$	$\mathbf{cegqi_b}$
h-uauto	15	10	**17**	13	16	**17**	16	**17**
keymaera	**108**	21	24	**108**	20	13	36	75
psyco	131	**132**	50	131	**132**	60	**132**	129
scholl	**232**	160	201	204	203	188	208	211
tptp	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**
uauto	14	14	15	**16**	14	14	14	14
ws-fixpoint	45	49	**54**	36	45	51	49	50
ws-ranking	19	15	**37**	33	33	31	31	32
Total sat	**581**	418	415	558	480	391	503	545
Total (5151)	4913	4521	4631	4920	4609	4571	4872	**4944**

Table 9. Results on satisfiable and unsatisfiable benchmarks with a 300 second timeout.

We evaluated our approach on all 5,151 benchmarks from the quantified bit-vector logic (BV) of SMT-LIB [3]. The results are summarized in Table 9. Configuration $\mathbf{cegqi_b}$ solves the highest number of unsatisfiable benchmarks $(4, 399)$, which is 30 more than the next best configuration $\mathbf{cegqi_s}$ and 37 more than the next best external solver, Z3. Compared to the instantiation-based solvers Boolector, CVC4 and Z3, the performance of $\mathbf{cegqi_b}$ is particularly strong on the h-uauto family, which are verification conditions from the Ultimate Automizer tool [11]. For satisfiable benchmarks, Boolector solves the most (581), which is 36 more than our best configuration $\mathbf{cegqi_b}$.

Overall, our best configuration $\mathbf{cegqi_b}$ solved 335 more benchmarks than our base configuration $\mathbf{cegqi_m}$. A more detailed runtime comparison between the two is provided by the scatter plot in Figure 4. Moreover, $\mathbf{cegqi_b}$ solved 24 more benchmarks than the best external solver, Z3. In terms of uniquely solved instances, $\mathbf{cegqi_b}$ was able to solve 139 benchmarks that were not solved by Z3, whereas Z3 solved 115 benchmarks that $\mathbf{cegqi_b}$ did not. Overall, $\mathbf{cegqi_b}$ was able to solve 21 of the 79 benchmarks (26.6%) not solved by any of the other solvers. For 18 of these 21 benchmarks, it terminated after considering no more than 4 instantiations. These cases indicate that using symbolic terms for instantiation solves problems for which other techniques, such as those that enumerate instantiations based on model values, do not scale.

Interestingly, configuration $\mathbf{cegqi_k}$, despite having the strong guarantees given by Theorem 11, performed relatively poorly on this set (with $4, 571$ solved instances overall). We attribute this to the fact that most of the quantified formulas in this set are not unit linear invertible. In total, we found that only 25.6% of the formulas consid-

ered during solving were unit linear invertible. However, only a handful of benchmarks were such that *all* quantified formulas in the problem were unit linear invertible. This might explain the superior performance of $\mathbf{cegqi_s}$ and $\mathbf{cegqi_b}$ which use invertibility conditions but in a less monolithic way.

For some intuition on this, consider the problem $\forall x.\,(x > a \lor x < b)$ where a and b are such that $a > b$ is T_{BV}-valid. Intuitively, to show that this formula is unsatisfiable requires the solver to find an x between b and a. This is apparent when considering the dual problem $\exists x.\,(x \leq a \land x \geq b)$. Configuration $\mathbf{cegqi_b}$ is capable of finding such an x, for instance, by considering the instantiation $x \mapsto a$ when solving for the boundary point of the first disjunct. Configuration $\mathbf{cegqi_k}$, on the other hand, would instead consider the instantiation of x for two terms that witness ε-expressions: some k_1 that

Fig. 4. Configuration $\mathbf{cegqi_m}$ vs. $\mathbf{cegqi_b}$.

is never smaller than a, and some k_2 that is never greater that b. Neither of these terms necessarily resides in between a and b since the solver may subsequently consider models where $k_1 > b$ and $k_2 < a$. This points to a potential use for invertibility conditions that solve multiple literals simultaneously, something we are currently investigating.
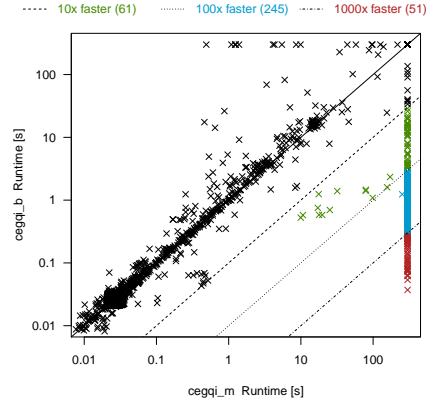
6 Conclusion

We have presented a new class of strategies for solving quantified bit-vector formulas based on invertibility conditions. We have derived invertibility conditions for the majority of operators in a standard theory of fixed-width bit-vectors. An implementation based on this approach solves over 25% of previously unsolved verification benchmarks from SMT LIB, and outperforms all other state-of-the-art bit-vector solvers overall.

In future work, we plan to develop a framework in which the correctness of invertibility conditions can be formally established independently of bit-width. We are working on deriving invertibility conditions that are optimal for linear constraints, in the sense of admitting the simplest propositional encoding. We also are investigating conditions that cover additional bit-vector operators, some cases of non-linear literals, as well as those that cover multiple constraints. While this is a challenging task, we believe efficient syntax-guided synthesis solvers can continue to help push progress in this direction. Finally, we plan to investigate the use of invertibility conditions for performing quantifier elimination on bit-vector constraints. This will require a procedure for deriving concrete witnesses from choice expressions.

References

1. Alur, R., Bodík, R., Juniwal, G., Martin, M.M.K., Raghothaman, M., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: Formal Methods in

Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013. pp. 1–8 (2013)

2. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Proceedings of the 23rd International Conference on Computer Aided Verification. pp. 171–177. CAV'11, Springer-Verlag (2011), http://dl.acm.org/citation.cfm?id=2032305.2032319

3. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. In: Gupta, A., Kroening, D. (eds.) Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK) (2010)

4. Bjørner, N., Janota, M.: Playing with quantified satisfaction. In: 20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations, LPAR 2015, Suva, Fiji, November 24-28, 2015. pp. 15–27 (2015)

5. Cooper, D.C.: Theorem proving in arithmetic without multiplication. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 7, pp. 91–100. Edinburgh University Press (1972)

6. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. TACAS'08/ETAPS'08, Springer-Verlag (2008), http://dl.acm.org/citation.cfm?id=1792734.1792766

7. Dutertre, B.: Yices 2.2. In: Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. pp. 737–744 (2014)

8. Dutertre, B.: Solving exists/forall problems in yices. Workshop on Satisfiability Modulo Theories (2015)

9. Ekici, B., Mebsout, A., Tinelli, C., Keller, C., Katz, G., Reynolds, A., Barrett, C.: Smtcoq: A plug-in for integrating smt solvers into coq. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. Lecture Notes in Computer Science, vol. 10427, pp. 126–133. Springer International Publishing (2017)

10. Ge, Y., de Moura, L.M.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5643, pp. 306–320. Springer (2009), https://doi.org/10.1007/978-3-642-02658-4_25

11. Heizmann, M., Chen, Y., Dietsch, D., Greitschus, M., Nutz, A., Musa, B., Schätzle, C., Schilling, C., Schüssele, F., Podelski, A.: Ultimate automizer with an on-demand construction of floyd-hoare automata - (competition contribution). In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10206, pp. 394–398 (2017), https://doi.org/10.1007/978-3-662-54580-5_30

12. Hilbert, D., Bernays, P.: Grundlagen der Mathematik. Die Grundlehren der mathematischen Wissenschaften, Verlag von Julius Springer (1934)

13. John, A.K., Chakraborty, S.: A layered algorithm for quantifier elimination from linear modular constraints. Formal Methods in System Design **49**(3), 272–323 (2016), https://doi.org/10.1007/s10703-016-0260-9

14. Jonás, M., Strejcek, J.: Solving quantified bit-vector formulas using binary decision diagrams. In: Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings. pp. 267–283 (2016)

15. Loos, R., Weispfenning, V.: Applying linear quantifier elimination (1993)

16. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0 system description. Journal on Satisfiability, Boolean Modeling and Computation **9**, 53–58 (2014 (published 2015))

17. Niemetz, A., Preiner, M., Biere, A.: Precise and complete propagation based local search for satisfiability modulo theories. In: Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I. pp. 199–217 (2016)

18. Niemetz, A., Preiner, M., Biere, A.: Propagation based local search for bit-precise reasoning. Formal Methods in System Design **51**(3), 608–636 (2017), https://doi.org/10.1007/s10703-017-0295-6

19. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: from an abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). Journal of the ACM **53**(6), 937–977 (Nov 2006)

20. Preiner, M., Niemetz, A., Biere, A.: Counterexample-guided model synthesis. In: Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I. pp. 264–280 (2017)

21. Reynolds, A., Deters, M., Kuncak, V., Tinelli, C., Barrett, C.W.: Counterexample-guided quantifier instantiation for synthesis in SMT. In: Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II. pp. 198–216 (2015)

22. Reynolds, A., King, T., Kuncak, V.: Solving quantified linear arithmetic by counterexample-guided instantiation. Formal Methods in System Design **51**(3), 500–532 (2017), https://doi.org/10.1007/s10703-017-0290-y

23. Stump, A., Sutcliffe, G., Tinelli, C.: Starexec: A cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Proceedings of the 7th International Joint Conference on Automated Reasoning. Lecture Notes in Computer Science, vol. 8562, pp. 367–373. Springer (2014)

24. Wintersteiger, C.M., Hamadi, Y., de Moura, L.M.: Efficiently solving quantified bit-vector formulas. Formal Methods in System Design **42**(1), 3–23 (2013)