Formal Methods in System Design (2021) 57:87–115
https://doi.org/10.1007/s10703-020-00359-9

On solving quantified bit-vector constraints using invertibility conditions

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

Received: 1 June 2019 / Accepted: 19 December 2020 / Published online: 18 January 2021
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC part of Springer Nature 2021

Abstract
We present a novel approach for solving quantified bit-vector constraints 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.

Keywords Satisfiability modulo theories · Quantified bit-vectors · Invertibility conditions

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. This is witnessed by the number of applications that generate problems in that fragment, and by past editions of the annual SMT competition where the corresponding division consistently attracts the highest number of solver and benchmark entries. Modeling properties of programs and circuits, e.g., universal safety properties and program invariants, however, often requires the use of *quanti-*

This work was partially supported by DARPA under awards FA8750-15-C-0113 and FA8650-18-2-7861 and by the National Science Foundation under award 1656926.

✉ Aina Niemetz
niemetz@cs.stanford.edu

1 Department of Computer Science, Stanford University, Stanford, USA

2 Department of Computer Science, The University of Iowa, Iowa City, USA

fied bit-vector formulas. Despite a multitude of applications, reasoning efficiently about such formulas is still a major challenge for automated tools.

The majority of solvers that support quantified bit-vector logics rely on instantiation-based techniques [8,25,27,30], which aim to find conflicting quantifier-free instances of universally quantified formulas in the input problem, with the goal of proving the problem unsatisfiable. For that, it is crucial to select good instantiations for the quantified variables, or else the solver may be overwhelmed by the number of quantifier-free instances generated. For example, consider the (unsatisfiable) quantified formula

$$\psi = \forall x.\, (x + s \not\approx t)$$

where x, s, and t denote bit-vectors of, say, size 32, and x occurs neither in s nor in t. 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 term $t - s$, denoting the unique solution of the equation $x + s \approx t$ for the variable x, we can immediately conclude that ψ is unsatisfiable since $(t - s) + s \not\approx t$ simplifies to false.

If we look at the term $x + s$ above as the bit-vector function $\lambda x.\, x + s$, we observe that the function is invertible, with inverse $\lambda y.\, y - s$. Informally then, we say that the bit-vector operator $+$ is *invertible* in its first argument. In particular, it is *always* invertible in that argument since the equation $x + s \approx t$ is always solvable for x. Since the same argument applies to the term $s + x$, the operator $+$ is always invertible in its second argument as well.

Not all the operators in the theory of bit-vectors are invertible in this sense. However, it is possible to identify quantifier-free conditions on one of their arguments that precisely *characterize* when they are invertible in that argument. For example, the constraint $x \cdot s \approx t$ is solvable for x *if and only if* the constraint $(-s \mid s)\ \&\ t \approx t$ is satisfiable. This is in fact a general property of bit-vector multiplication that always holds (for any bit-vector size n) for x, s and t [23]. The *invertibility condition* $(-s \mid s)\ \&\ t \approx t$ essentially states that the value of t has at least as many right-most zeroes in its binary representation as the value of s.

We have identified invertibility conditions for a representative set of operators in the standard theory of bit-vectors supported by SMT solvers. We present a novel approach for quantifier instantiation of bit-vector formulas that utilizes those invertibility conditions to generate symbolic instantiations. As an example, consider the quantified formula $\varphi = \forall x.\, (x \cdot s \not\approx t)$. If the above invertibility condition $(-s \mid s)\ \&\ t \approx t$ is unsatisfiable, we conclude that φ is satisfiable since there is no x that makes its body $x \cdot s \not\approx t$ false. On the other hand, if the invertibility condition is satisfiable, we know there is some bit-vector value b such that $b \cdot s \approx t$ holds. Instantiating x in φ with any term k denoting b to produce the quantifier-free formula $k \cdot s \not\approx t$ suffices to show that φ is unsatisfiable. In general, the term k can be always generated automatically from the invertibility condition by using the Hilbert choice operator. Formally, we show that invertibility conditions can be embedded into quantifier instantiations using Hilbert's choice function in a sound manner. This approach has compelling advantages with respect to previous approaches, as demonstrated by our experimental results.

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 the theory of bit-vectors defined by the SMT-LIB standard. [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 substantial improvements over state-of-the-art solvers for quantified bit-vector constraints.

An earlier version of this work appeared at CAV 2018, held as part of FLOC, in Oxford, UK [22]. This article extends that work with a more thorough description and listing of invertibility conditions in Sect. 3, formal proofs of all lemmas and theorems throughout, new implementation details in Sect. 4.2 and further details on our evaluation in Sect. 5.

Related work. Quantified bit-vector logics are currently supported by the state-of-the-art SMT solvers Boolector [19], CVC4 [2], Yices 2 [7], and Z3 [6] and a Binary Decision Diagram (BDD)-based tool called Q3B [16]. 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 2, 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) [27], where concrete models of a set of quantifier-free instances and the negation of the input formula (the counterexamples) serve as instantiations for the universal variables. In Z3, model-based quantifier instantiation (MBQI) [12] is combined with a template-based model finding procedure [30]. 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 quantifier-free terms with the same model value. Boolector employs a syntax-guided synthesis approach to synthesize interpretations for Skolem functions based on a set of quantifier-free instances of the formula, and uses a counterexample refinement loop similar to MBQI [25]. Other counterexample-guided approaches for quantified formulas in SMT solvers have been considered by Bjørner and Janota [4] and by Reynolds et al. [28], 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 [15]. We do not explore approaches for quantifier elimination in this paper.

2 Preliminaries

We assume the usual notions and terminology of many-sorted first-order logic with equality (denoted by $\approx$) (see, e.g., [11,18]). 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 $\sigma_1, \ldots, \sigma_n, \sigma \in \Sigma^s$. We assume that a signature Σ includes a Boolean sort Bool and the Boolean constants $\top$ (true) and $\bot$ (false). A Σ-*interpretation* maps: each $\sigma \in \Sigma^s$ to a non-empty set $\sigma^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$), with $\mathsf{Bool}^{\mathcal{I}} = \{\top, \bot\}$; 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 $f^{\mathcal{I}}\colon \sigma_1^{\mathcal{I}} \times \ldots \times \sigma_n^{\mathcal{I}} \to \sigma^{\mathcal{I}}$ 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 assume the usual definition of well-sorted terms, literals, and formulas (where formulas are terms of sort Bool) with variables in X and symbols in Σ, and refer to them as Σ-terms, Σ-atoms, and so on. We also assume the usual definition of free variables of a formula. A *closed* formula is a Σ-formula without free

variables. We adopt the usual inductive definition of the meaning, or *value*, $t^{\mathcal{I}}$ of a Σ-term t in a Σ-interpretation $\mathcal{I}$ and of the satisfiability relation $\models$ between Σ-interpretations and Σ-formulas.

2.1 Notation

We denote by $\mathrm{Lit}(\varphi)$ the set of Σ-literals that either appear in, or are negations of atomic formulas appearing in, Σ-formula φ. Let $\boldsymbol{x} = (x_1, ..., x_n)$ be a (possibly empty) tuple of distinct variables. We write $Q\boldsymbol{x}.\varphi$ with $Q \in \{\forall, \exists\}$ for a *quantified* formula $Qx_1 \cdots Qx_n.\varphi$. For a Σ-term or Σ-formula e, we denote the *free variables* of e (defined as usual) as $FV(e)$; we write $e[\boldsymbol{x}]$ to indicate that the variables of $\boldsymbol{x}$ occur free in e (i.e., $\boldsymbol{x} \subseteq FV(e)$); in addition, if $\boldsymbol{t} = (t_1, ..., t_n)$ is a tuple of Σ-terms, $e\{\boldsymbol{x} \mapsto \boldsymbol{t}\}$ denotes the term or formula obtained from e by simultaneously replacing each occurrence of x_i in e by t_i for $i = 1, \ldots, n$. Abusing the notation, when the relevant free variables $\boldsymbol{x}$ of e are clear from context, we write $e[\boldsymbol{t}]$ as an abbreviation of $e\{\boldsymbol{x} \mapsto \boldsymbol{t}\}$. When convenient, we identify a finite set of formulas with the conjunction of its elements.

Given a Σ-formula $\varphi[x]$ with $x \in X_\sigma$, we use Hilbert's *choice* operator ε [14] to describe values (i.e. domain elements) for x for which $\varphi[x]$ is satisfiable. Formally, we define a *choice term* $\varepsilon x.\varphi$ as a term of sort σ where x is bound by ε. In every interpretation $\mathcal{I}$, $\varepsilon x.\varphi$ denotes some value $v \in \sigma^{\mathcal{I}}$ such that $\mathcal{I}[x \mapsto v]$ satisfies φ if such values exist, and denotes an arbitrary element of $\sigma^{\mathcal{I}}$ otherwise. This means that the formula $\exists x.\varphi[x] \Leftrightarrow \varphi[\varepsilon x.\varphi]$ is satisfied by every interpretation. For example, in any interpretation $\mathcal{I}$ extending the standard interpretation of integer arithmetic, $\varepsilon x.x > 0$ denotes some integer greater than 0; $\varepsilon x.x > y$ denotes some value greater than $y^{\mathcal{I}}$. We remark that in the standard definition of the choice binder, the meaning of $\varepsilon x.\varphi$ is invariant modulo logical equivalence, that is, the equality $\varepsilon x.\varphi \approx \varepsilon x.\psi$ is satisfied by every interpretation in which the formulas φ and ψ are equivalent. However, the correctness of the work we present here does not rely on this invariant property.

We say that a formula is *quantifier-free* if it contains no quantifiers and is *binder-free* if it contains no quantifiers and no choice operators.

2.2 Theories

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. Two Σ-formulas φ and ψ are *T-equivalent* if the formula $\varphi \Leftrightarrow \psi$ is T-valid. A set Γ of Σ-formulas is *T-satisfiable* (resp., *T-unsatisfiable*) if there is an interpretation (resp., no interpretation) in I that satisfies all the formulas in Γ. The set Γ *T-entails* φ if $\Gamma \cup \{\neg\varphi\}$ is T-unsatisfiable. A theory T is a *complete* theory if for all closed Σ-formulas φ, φ is either T-valid or T-unsatisfiable. A choice term $\varepsilon x.\varphi$ is *T-valid* if $\exists x.\varphi$ is T-valid. We refer to a term t as *ε-T-valid* if all occurrences of choice terms in t are T-valid. We will sometimes omit T from the definitions above when the theory T is understood from context.

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

Symbol	SMT-LIB Syntax	Sort
$\approx, <_{\mathrm{u}}, >_{\mathrm{u}}, <_{\mathrm{s}}, >_{\mathrm{s}}$	=, bvult, bvugt, bvslt, bvsgt	$\sigma_{[n]} \times \sigma_{[n]} \rightarrow \mathsf{Bool}$
$\sim, -$	bvnot, bvneg	$\sigma_{[n]} \rightarrow \sigma_{[n]}$
$\&, \mid, <<, >>, >>_a$	bvand, bvor, bvshl, bvlshr, bvashr	$\sigma_{[n]} \times \sigma_{[n]} \rightarrow \sigma_{[n]}$
$+, \cdot, \mathrm{mod}, \div$	bvadd, bvmul, bvurem, bvudiv	$\sigma_{[n]} \times \sigma_{[n]} \rightarrow \sigma_{[n]}$
$\circ$	concat	$\sigma_{[n]} \times \sigma_{[m]} \rightarrow \sigma_{[n+m]}$
$[u : l]$	extract	$\sigma_{[n]} \rightarrow \sigma_{[u-l+1]}, 0 \leq l \leq u < n$

2.3 The theory of fixed-size bit-vectors

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 and represent those constants as bit-strings of 0s and 1s. All interpretations $\mathcal{I} \in I_{BV}$ are identical except for the value they assign to variables.[1] They interpret sort and function symbols as specified in SMT-LIB 2. All function symbols in Σ^f_{BV} are overloaded for every $\sigma_{[n]} \in \Sigma^s_{BV}$. 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 value* of width n, e.g., $\max_{s[4]} = 0111$ and $\min_{s[4]} = 1000$. To simplify the notation we will sometimes denote bit-vector constants by the corresponding natural number in $\{0, \ldots, 2^{n-1}\}$ (with a number c corresponding to a bit-string b if b has value c when seen as a binary number), adding the bit-width as a subscript; e.g., we will use $2_{[4]}$ for 0010. The width of a bit-vector sort or term is given by the function κ, e.g., $\kappa(\sigma_{[n]}) = n$ and $\kappa(t_{[n]}) = n$. We will omit the bit-width from the notation when it is clear from the context or not important. We will use the letters s and t to denote bit-vector terms in general and the letters x and y to denote bit-vector variables.

Without loss of generality, we consider a restricted set of bit-vector function symbols (or *bit-vector operators*) Σ^f_{BV} 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. This means that our approach is not really restricted to this particular set of operators as it can be lifted to any other set of bit-vector operators.

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 satisfy a syntactic condition we call *linearity*.

Definition 1 (Linear Literal) A Σ_{BV}-literal $\ell[x]$ is *linear* in variable x if it has a *single* occurrence of x.

[1] Note that this makes this theory complete in the sense of Sect. 2.2.

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, 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 is not always possible. For example, recalling that the semantics of arithmetic operators over bit-vectors of size n is that of integer arithmetic modulo n, for literals of the form $x \cdot s \approx t$, an inverse always exists provided that s always evaluates to a bit-vector denoting an odd integer.[2] Otherwise, there are values for s and t where no such inverse exists, e.g., $x \cdot 2 \approx 3$.

Even if a bit-vector operation may admit no unconditional inverse in general, it is possible to identify the condition under which the operation is invertible. For instance, as we mentioned earlier, it is possible to prove that the multiplication constraint $x \cdot s \approx t$ with $x \notin FV(s) \cup FV(t)$ is invertible for x exactly when the number of right-most zeroes in s is no greater than the number of right-most zeroes in t. This *invertibility condition* can be expressed by the formula $(-s \mid s)$ & $t \approx t$. Note that the operators $-$, $\sim$, and $+$ for which a general inverse always exists can be seen as having $\top$ as their invertibility condition.

Definition 2 (Invertibility Condition) Let $\ell[x]$ be a Σ_{BV}-literal. A binder-free Σ_{BV}-formula ϕ is an *invertibility condition* for x in ℓ if $FV(\phi) \subseteq FV(\ell) \setminus \{x\}$ and the formula $\phi \Leftrightarrow \exists x.\, \ell$ is T_{BV}-valid.

An invertibility condition for a literal $\ell[x]$ provides the *exact conditions* under which ℓ is solvable for x. We call it an "invertibility" condition because we can use Hilbert choice terms 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 ℓ. Recall that a choice term $\varepsilon x.\, \varphi$ denotes a solution of the formula $\varphi[x]$ for x if the formula is satisfiable, and denotes an arbitrary value otherwise. We can use the choice term $\varepsilon x.\, (\phi \Rightarrow \ell)$ to describe inverse solutions of a literal $\ell[x]$ with invertibility condition ϕ. For example, for the general case of bit-vector multiplication over equality the choice term is defined as $\varepsilon x.\, ((-s \mid s)$ & $t \approx t \;\Rightarrow\; x \cdot s \approx t)$. We favor the choice term $\varepsilon x.\, (\phi \Rightarrow \ell)$ over the simpler choice term $\varepsilon x.\, \ell$ because $\phi \Rightarrow \ell$ has a solution in every model of T_{BV}, whereas ℓ may have no solution.

Lemma 3 *Let ϕ be an invertibility condition for an ε-valid Σ_{BV}-literal $\ell[x]$ and let r be the term $\varepsilon x.\, (\phi \Rightarrow \ell)$. The term r is ε-valid and $\ell[r] \Leftrightarrow \exists x.\, \ell$ is T_{BV}-valid.*

Proof First, we show that $r = \varepsilon x.\, (\phi \Rightarrow \ell)$ is ε-valid, where ϕ is an invertibility condition for x in $\ell[x]$. To do so, since ℓ is ε-valid, we only need to show that $\exists x.\, \phi \Rightarrow \ell$ holds in all models of T_{BV}. Since ϕ is an invertibility condition for ℓ, we have that $x \notin FV(\phi)$, and hence, this formula is equivalent to $\phi \Rightarrow \exists x.\, \ell$. Let $\mathcal{I}$ be any model of T_{BV} that satisfies ϕ. Since ϕ is an invertibility condition for ℓ, by Definition 2, $\mathcal{I}$ satisfies $\exists x.\, \ell$ as well. Thus, $\exists x.\, \phi \Rightarrow \ell$ holds in all models of T_{BV}, and hence, r is ε-valid.

To show $\ell[r] \Leftrightarrow \exists x.\, \ell$ where r is $\varepsilon x.\, (\phi \Rightarrow \ell)$, first consider direction $\exists x.\, \ell \Rightarrow \ell[r]$. Let $\mathcal{I}$ be any model of T_{BV} that satisfies $\exists x.\, \ell$. By definition of ε, $\mathcal{I}$ also satisfies $\ell[\varepsilon x.\, \ell]$. Since ϕ is an invertibility condition for ℓ, from Def. 2 we have that $\phi \Leftrightarrow \exists x.\, \ell$ holds in all models of T_{BV}, and thus $\mathcal{I}$ also satisfies ϕ. Hence, since $\mathcal{I}$ satisfies $\ell[\varepsilon x.\, \ell]$, it also satifies $\ell[\varepsilon x.\, (\phi \Rightarrow \ell)]$, which is $\ell[r]$. Thus, $\exists x.\, \ell \Rightarrow \ell[r]$ is T_{BV}-valid. The other direction $\ell[r] \Rightarrow \exists x.\, \ell$ trivially holds in all models of T_{BV}.

□

Intuitively, when $\ell[x]$ is satisfiable in some model of the theory, the value of the choice term $\varepsilon x.\, (\phi \Rightarrow \ell)$ in that model is a value for x that is a solution of ℓ. Conversely, if ℓ is

[2] That is, provided that s is equivalent to a term of the form $2_{[n]} \cdot s' + 1_{[n]}$.

$\mathsf{solve}(x, e[x] \bowtie t)$:
  If $e = x$
    If $\bowtie \in \{\approx\}$ then return t
    else return $\varepsilon y.\ (\mathsf{getIC}(x, x \bowtie t) \Rightarrow y \bowtie t)$.
  else $e = \diamond(e_1, \ldots, e_i[x], \ldots, e_n)$ with $n > 0$ and $x \notin FV(e_j)$ for all $j \neq i$.
    Let $d[x'] = \diamond(e_1, \ldots, e_{i-1}, x', e_{i+1}, \ldots, e_n)$ where x' is a fresh variable.
    If $\bowtie \in \{\approx, \not\approx\}$ and $\diamond \in \{\sim, -, +\}$
    then let $t' = \mathsf{getInverse}(x', d[x'] \approx t)$ and return $\mathsf{solve}(x, e_i \bowtie t')$
    else let $\phi = \mathsf{getIC}(x', d[x'] \bowtie t)$ and return $\mathsf{solve}(x, e_i \approx \varepsilon y.\ (\phi \Rightarrow d[y] \bowtie t))$.

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

satisfiable under no conditions in some model, then the value of $\varepsilon x.\ (\phi \Rightarrow \ell)$ is still a solution for $\phi \Rightarrow \ell$ because ϕ is false in that model.

3.1 Solving linear invertible constraints

Now, suppose a Σ_{BV}-literal ℓ is linear in variable x, but x occurs arbitrarily deep in it. 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 term

$$x' = \varepsilon y.\ ((-s_1 \mid s_1)\ \&\ t \approx t \Rightarrow s_1 \cdot y \approx t)\ .$$

Then, we solve $s_2 + x \approx x'$ for x, which yields the solution

$$x = x' - s_2 = \varepsilon y.\ ((-s_1 \mid s_1)\ \&\ t \approx t \Rightarrow s_1 \cdot y \approx t) - s_2\ .$$

Figure 1 describes in pseudocode 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 an analytic solution by computing (conditional) inverses as follows. Let function $\mathsf{getInverse}(x, \ell[x])$ (see Table 2) return a term t' that is the inverse of x in ℓ, i.e., such that the equivalence $\ell \Leftrightarrow x \approx t'$ is valid. Furthermore, let function $\mathsf{getIC}(x, \ell[x])$ return the invertibility condition ϕ for x in ℓ. 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 $\mathsf{getInverse}(x', d[x'] \approx t)$, if it exists. Note that for a disequality $e \not\approx t$, 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 $t' = \mathsf{getInverse}(x', x' + s \approx t) = t - s$ and recurse on $e_i[x] \not\approx t'$.) If no inverse for $e[x] \bowtie t$ exists, we determine the invertibility condition ϕ for $d[x']$ via $\mathsf{getIC}(x', d[x'] \bowtie t)$, construct the choice term $\varepsilon y.\ (\phi \Rightarrow d[y] \bowtie t)$, and set it equal to $e_i[x]$, before recursively solving for x. If $e = x$ and the given literal is an equality, we have reached the base case and return t as the solution for x. Note that in Fig. 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.

Definition 4 (Linear Invertible Literal) Literal $\diamond(e_1, \ldots, e_i[x], \ldots, e_n) \bowtie t$ is *linear invertible* in x if it is linear in x and if for each function application in $e_i[x]$, if one of its arguments contains x, then the function operator is in $\{-, +, \sim\}$.

In other words, only the topmost function symbol $\diamond$ in a linear invertible literal may be conditionally invertible. For example, $(s + x) \cdot t \approx r$ and $s \cdot (t + x) <_{\mathrm{u}} r$ are linear invertible in x whereas $(s \mid x) \cdot t \approx r$ and $(s \cdot x) + t <_{\mathrm{u}} r$ are not. Note that some equalities such as $(x \cdot s) + t \approx r$ are not linear invertible by this definition, but can easily be shown to be equivalent to linear invertible ones, i.e. $x \cdot s \approx r - t$.

Lemma 5 *If x occurs only beneath operators that are always invertible in $e[x]$, then for any term t not containing x, $\exists x.\, e[x] \approx t$ is equivalent to $\top$.*

Proof The proof is by induction on the structure of $e[x]$. In the base case, if $e[x]$ is x, then the statement holds trivially. For the inductive case, assume it is true for some $e[x]$. Let $e'[x]$ be $\diamond(\ldots, e[x], \ldots)$. Since $\diamond$ is always invertible, $\exists x.\, e'[x] \approx t$ is equivalent to $\exists x.\, e[x] \approx t'$, for some t' not containing x. But this is equivalent to $\top$ by the induction hypothesis.

□

The procedure solve is correct in the following sense.

Theorem 6 *Let $\ell[x]$ be an ε-valid Σ_{BV}-literal linear in x, and let $r = \mathsf{solve}(x, \ell)$ be the term returned by function* solve. *Then (i) r is ε-valid, (ii) $FV(r) \subseteq FV(\ell) \setminus \{x\}$ and (iii) $\ell[r] \Leftrightarrow \exists x.\, \ell$ is T_{BV}-valid when $\ell[x]$ is linear invertible in x.*

Proof We assume without loss of generality that $\ell[x]$ is of the form $e[x] \bowtie t$. Since ℓ 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 is x, which implies that $e \bowtie t$ is linear invertible in 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, i.e. is satisfied, by 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 $\mathsf{getIC}(x', x' \bowtie t)$. 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 3, r is ε-valid and $r \bowtie t \Leftrightarrow \exists x.\, x \bowtie t$ is T_{BV}-valid.

Otherwise, e 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 $\mathsf{solve}(x, e_i[x] \bowtie_i t_i)$ for some relation $\bowtie_i$ and term t_i, where t_i is $\mathsf{getInverse}(x', d[x'] \approx t)$ or $\varepsilon y.\, (\mathsf{getIC}(x', d[x'] \bowtie t) \Rightarrow d[y] \bowtie t)$. In both cases, by definition of getInverse (see Table 2) and getIC, we have that t_i is ε-valid due to Lemma 3 and since ℓ is ε-valid. Also, in both cases, we have that $FV(t_i) \subseteq FV(t) \cup FV(d[x']) \setminus \{x'\} \subseteq FV(\ell) \setminus \{x\}$. Since $e \bowtie t$ is ε-valid and linear invertible in x and $x \notin FV(t_i)$, the literal $e_i \bowtie_i t_i$ is ε-valid and linear invertible in 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}.

Property (i) holds since r is ε-valid. To show property (ii), since $FV(r) \subseteq FV(e_i \bowtie_i t_i) \setminus \{x\}$ and since $FV(e_i) \subseteq FV(\ell)$ and $FV(t_i) \subseteq FV(\ell)$, we have that $FV(r) \subseteq FV(\ell) \setminus \{x\}$.

It remains to show property (iii), that $e[r] \bowtie t \Leftrightarrow \exists x.\, e \bowtie t$ is T_{BV}-valid when $e[x] \bowtie t$ is linear invertible in x. In the case that $\bowtie \in \{\approx, \not\approx\}$ and $\diamond \in \{\sim, -, +\}$, we have that $\bowtie_i$ is $\bowtie$ and t_i is $\mathsf{getInverse}(x', d[x'] \approx t)$. By definition of getInverse and since $\bowtie \in \{\approx, \not\approx\}$,

we have that $e_i \bowtie t_i$ and $d[e_i] \bowtie t$ are equivalent. Since $d[e_i] = e$, the latter literal is $e \bowtie t$. Thus, since $e_i[r] \bowtie t_i \Leftrightarrow \exists x.\, e_i \bowtie t_i$, we have that $e[r] \bowtie t \Leftrightarrow \exists x.\, e \bowtie t$. Otherwise, let $\psi = \mathsf{getIC}(x', d[x'] \bowtie t)$; we have that $\bowtie_i$ is $\approx$ and t_i is $\varepsilon y.\, (\psi \Rightarrow d[y] \bowtie t)$. Since ψ is an invertibility condition for x' in $d[x'] \bowtie t$, by Lemma 3, we have that $d[t_i] \bowtie t \Leftrightarrow \exists x'.\, d[x'] \bowtie t$ holds in all models of T_{BV}. Clearly $e[r] \bowtie t \Rightarrow \exists x.\, e \bowtie t$ holds in all models of T_{BV}. To show the other direction of the implication, consider any model $\mathcal{I}$ of T_{BV} that satisfies $\exists x.\, e \bowtie t$. Since $e = d[e_i]$, we have that $\mathcal{I}$ satisfies $\exists x'.\, d[x'] \bowtie t$ as well. Thus, since $d[t_i] \bowtie t \Leftrightarrow \exists x'.\, d[x'] \bowtie t$ holds in all models of T_{BV}, we have that $\mathcal{I}$ satisfies $d[t_i] \bowtie t$. Since $e[x]$ is linear invertible in x, we have that x occurs only under operators that are always invertible in $e_i[x]$, and thus by Lemma 5, $\exists x.\, e_i[x] \approx t_i$ is equivalent to $\top$. Since $e_i[r] \approx t_i \Leftrightarrow \exists x.\, e_i[x] \approx t_i$ holds by the induction hypothesis, we have that $\mathcal{I}$ satisfies $e_i[r] \approx t_i$. Since $\mathcal{I}$ satisfies $d[t_i] \bowtie t$, we have that it also satisfies $d[e_i[r]] \bowtie t$, which is $e[r] \bowtie t$. Thus, $e[r] \bowtie t \Leftarrow \exists x.\, e \bowtie t$ holds in all models of T_{BV}. □

We will use solve for choosing terms for quantifier instantiation, where the above properties are important for the correctness and completeness properties of our algorithm. Notice that this method may be applied to any literal that is linear in x, although it only has the third property when that literal is linear invertible. In the context of quantifier instantiation, the solve method is generally only effective when used on literals that are linear invertible.

3.2 Invertibility conditions

Table 2 shows the rules for inverse computation for bit-vector operators $-$, $\sim$, $+$, and $\cdot$ over equality. Note that for the first three operators, these inverses are general inverses. For bit-vector multiplication, however, the given inverse is not, since it is tied to the condition that s evaluates to an odd value.

Tables 3, 4, 5, 6 and 7 list the invertibility conditions for linear literals in x with bit-vector operators from the set $\{-, \sim, +, \cdot, \mathrm{mod}, \div, \&, |, \gg, \gg_a, \ll, \circ\}$ and relational operators $\bowtie$ from the set $\{\approx, \not\approx, <_\mathrm{u}, >_\mathrm{u}, \leq_\mathrm{u}, \geq_\mathrm{u}, <_\mathrm{s}, >_\mathrm{s}, \leq_\mathrm{s}, \geq_\mathrm{s}\}$. Some invertibility conditions are $\top$, meaning that the literal is always solvable for x. Others have an invertibility condition that is the same as the literal itself but with x replaced by a term not containing x. For example, the invertibility condition for equality with division $x \div s \approx t$ is $(s \cdot t) \div s \approx t$. In other words, $x \div s \approx t$ is solvable for x if and only if $(s \cdot t)$ is a solution. Other invertibility conditions have no structural similarities with their corresponding literal. For example, the invertibility condition for multiplication with equality does not itself involve multiplication. A majority of the invertibility conditions for disequalities state explicit corner cases where the constraint is not solvable, whereas for inequalities the majority encode boundary checks. We discuss in Sect. 3.3 how we determined these invertibility conditions.

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. [20]. There, target values are propagated via inverse value computation. In contrast, our approach does not determine single inverse values based on concrete assignments, but rather aims at finding analytic, i.e., symbolic, solutions through the generation of conditional inverses. In an extended version of that work [21], the same authors present rules for inverse value computation over equality but they provide no proof of correctness for them. Here, 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.

Table 2 Inverse computation for bit-vector operators $\{-, \sim, +, \cdot\}$ over equality (given modulo commutativity for addition and multiplication)

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

Table 3 Conditions for the invertibility of bit-vector operators over equality and disequality (given modulo commutativity for operators $+$, $\cdot$, & and $|$)

$\ell[x]$	$\approx$	$\not\approx$
$x \bowtie t$ $-x \bowtie t$ $\sim x \bowtie t$ $x + s \bowtie t$	$\top$	$\top$
$x \cdot s \bowtie t$	$(-s \mid s) \mathbin{\&} t \approx t$	$s \not\approx 0 \lor t \not\approx 0$
$x \bmod s \bowtie t$	$\sim(-s) \geq_{\mathrm{u}} t$	$s \not\approx 1 \lor t \not\approx 0$
$s \bmod x \bowtie t$	$(t + t - s) \mathbin{\&} s \geq_{\mathrm{u}} t$	$s \not\approx 0 \lor t \not\approx 0$
$x \div s \bowtie t$	$(s \cdot t) \div s \approx t$	$s \not\approx 0 \lor t \not\approx \sim 0$
$s \div x \bowtie t$	$s \div (s \div t) \approx t$	$\begin{cases} s \mathbin{\&} t \approx 0 & \text{for } \kappa(s) = 1 \\ \top & \text{otherwise} \end{cases}$
$x \mathbin{\&} s \bowtie t$	$t \mathbin{\&} s \approx t$	$s \not\approx 0 \lor t \not\approx 0$
$x \mid s \bowtie t$	$t \mid s \approx t$	$s \not\approx \sim 0 \lor t \not\approx \sim 0$
$x \gg s \bowtie t$	$(t \ll s) \gg s \approx t$	$t \not\approx 0 \lor s <_{\mathrm{u}} \kappa(s)$
$s \gg x \bowtie t$	$\bigvee_{i=0}^{\kappa(s)} s \gg i \approx t$	$s \not\approx 0 \lor t \not\approx 0$
$x \gg_a s \bowtie t$	$(s <_{\mathrm{u}} \kappa(s) \Rightarrow (t \ll s) \gg_a s \approx t) \land$ $(s \geq_{\mathrm{u}} \kappa(s) \Rightarrow (t \approx \sim 0 \lor t \approx 0))$	$\top$
$s \gg_a x \bowtie t$	$\bigvee_{i=0}^{\kappa(s)} s \gg_a i \approx t$	$(t \not\approx 0 \lor s \not\approx 0) \land$ $(t \not\approx \sim 0 \lor s \not\approx \sim 0)$
$x \ll s \bowtie t$	$(t \gg s) \ll s \approx t$	$t \not\approx 0 \lor s <_{\mathrm{u}} \kappa(s)$
$s \ll x \bowtie t$	$\bigvee_{i=0}^{\kappa(s)} s \ll i \approx t$	$s \not\approx 0 \lor t \not\approx 0$
$x \circ s \bowtie t$	$s \approx t[\kappa(s) - 1 : 0]$	$\top$
$s \circ x \bowtie t$	$s \approx t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	$\top$

3.3 Synthesizing invertibility conditions

We have identified 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 us do that. The reason is that the problem of finding invertibility conditions for a literal of the form $x \diamond s \bowtie t$ (or, symmetrically, $s \diamond x \bowtie t$) linear in x can be recast as a SyGuS problem by asking whether

Table 4 Conditions for the invertibility of bit-vector operators over unsigned inequalities $<_{\mathrm{u}}$ and $>_{\mathrm{u}}$ (given modulo commutativity for operators $+$, $\cdot$, & and |)

$\ell[x]$	$<_{\mathrm{u}}$	$>_{\mathrm{u}}$
$x \bowtie t$ $-x \bowtie t$ $\sim x \bowtie t$ $x + s \bowtie t$	$t \not\approx 0$	$t \not\approx \sim 0$
$x \cdot s \bowtie t$	$t \not\approx 0$	$t <_{\mathrm{u}} -s \mid s$
$x \bmod s \bowtie t$	$t \not\approx 0$	$t <_{\mathrm{u}} \sim(-s)$
$s \bmod x \bowtie t$	$t \not\approx 0$	$t <_{\mathrm{u}} s$
$x \div s \bowtie t$	$0 <_{\mathrm{u}} s \wedge 0 <_{\mathrm{u}} t$	$\sim 0 \div s >_{\mathrm{u}} t$
$s \div x \bowtie t$	$0 <_{\mathrm{u}} \sim(-t \,\&\, s) \wedge 0 <_{\mathrm{u}} t$	$t <_{\mathrm{u}} \sim 0$
$x \,\&\, s \bowtie t$	$t \not\approx 0$	$t <_{\mathrm{u}} s$
$x \mid s \bowtie t$	$s <_{\mathrm{u}} t$	$t <_{\mathrm{u}} \sim 0$
$x \gg s \bowtie t$	$t \not\approx 0$	$t <_{\mathrm{u}} \sim s \gg s$
$s \gg x \bowtie t$	$t \not\approx 0$	$t <_{\mathrm{u}} s$
$x \gg_a s \bowtie t$	$t \not\approx 0$	$t <_{\mathrm{u}} \sim 0$
$s \gg_a x \bowtie t$	$(s <_{\mathrm{u}} t \vee s \geq_{\mathrm{s}} 0) \wedge t \not\approx 0$	$s <_{\mathrm{s}} (s \gg \sim t) \vee t <_{\mathrm{u}} s$
$x \ll s \bowtie t$	$t \not\approx 0$	$t <_{\mathrm{u}} \sim 0 \ll s$
$s \ll x \bowtie t$	$t \not\approx 0$	$\bigvee_{i=0}^{\kappa(s)} (s \ll i) >_{\mathrm{u}} t$
$x \circ s \bowtie t$	$t_x \approx 0 \Rightarrow s <_{\mathrm{u}} t_s$	$t_x \approx \sim 0 \Rightarrow s >_{\mathrm{u}} t_s$
	where $t_x = t[\kappa(t) - 1 : \kappa(t) - \kappa(x)], t_s = t[\kappa(s) - 1 : 0]$	
$s \circ x \bowtie t$	$s \leq_{\mathrm{u}} t_s \wedge (s \approx t_s \Rightarrow t_x \not\approx 0)$	$s \geq_{\mathrm{u}} t_s \wedge s \approx t_s \Rightarrow t_x \not\approx \sim 0$
	where $t_x = t[\kappa(x) - 1 : 0], t_s = t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	

there exists a binary Boolean function C such that the (second-order) bit-vector formula

$$\exists C \forall s \forall t.\, (\exists x.\, x \diamond s \bowtie t) \Leftrightarrow C(s, t)$$

is satisfiable. If a SyGuS solver succeeds in synthesizing the function C, we can use it 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

$$\exists C \forall s \forall t.\, \left(\bigvee_{i=0}^{15} i \diamond s \bowtie t \right) \Leftrightarrow C(s, t) \tag{1}$$

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 Boolean connectives but different sets of bit-vector operators:

$$O_r = \{\neg, \wedge, \approx, <_{\mathrm{u}}, <_{\mathrm{s}}, 0, \min_s, \max_s, s, t, \sim, -, \&, |\}$$
$$O_g = \{\neg, \wedge, \vee, \approx, <_{\mathrm{u}}, <_{\mathrm{s}}, \geq_{\mathrm{u}}, \geq_{\mathrm{s}}, 0, \min_s, \max_s, s, t, \sim, +, -, \&, |, \gg, \ll\}$$

Table 5 Conditions for the invertibility of bit-vector operators over unsigned inequalities $\leq_{\mathrm{u}}$ and $\geq_{\mathrm{u}}$ (given modulo commutativity for operators $+$, $\cdot$, & and |)

$\ell[x]$	$\leq_{\mathrm{u}}$	$\geq_{\mathrm{u}}$
$x \bowtie t$ $-x \bowtie t$ $\sim x \bowtie t$ $x + s \bowtie t$	$\top$	$\top$
$x \cdot s \bowtie t$	$\top$	$-s \mid s \geq_{\mathrm{u}} t$
$x \bmod s \bowtie t$	$\top$	$\sim(-s) \geq_{\mathrm{u}} t$
$s \bmod x \bowtie t$	$\top$	$(t + t - s) \,\&\, s \geq_{\mathrm{u}} t \vee t <_{\mathrm{u}} s$
$x \div s \bowtie t$	$s \mid t \geq_{\mathrm{u}} \sim(-s)$	$(s \cdot t) \div t \,\&\, s \approx s$
$s \div x \bowtie t$	$0 <_{\mathrm{u}} \sim s \mid t$	$\top$
$x \,\&\, s \bowtie t$	$\top$	$s \geq_{\mathrm{u}} t$
$x \mid s \bowtie t$	$t \geq_{\mathrm{u}} s$	$\top$
$x \gg s \bowtie t$	$\top$	$(t \ll s) \gg s \approx t$
$s \gg x \bowtie t$	$\top$	$s \geq_{\mathrm{u}} t$
$x \gg_a s \bowtie t$	$\top$	$\top$
$s \gg_a x \bowtie t$	$s <_{\mathrm{u}} \min_s \vee t \geq_{\mathrm{u}} s$	$s \geq_{\mathrm{u}} \sim s \vee s \geq_{\mathrm{u}} t$
$x \ll s \bowtie t$	$\top$	$\sim 0 \ll s \geq_{\mathrm{u}} t$
$s \ll x \bowtie t$	$\top$	$\bigvee_{i=0}^{\kappa(s)} (s \ll i) \geq_{\mathrm{u}} t$
$x \circ s \bowtie t$	$t_x \approx 0 \Rightarrow s \leq_{\mathrm{u}} t_s$ where $t_x = t[\kappa(t) - 1 : \kappa(t) - \kappa(x)], t_s = t[\kappa(s) - 1 : 0]$	$t_x \approx \sim 0 \Rightarrow s \geq_{\mathrm{u}} t_s$
$s \circ x \bowtie t$	$s \leq_{\mathrm{u}} t_s$ where $t_x = t[\kappa(x) - 1 : 0], t_s = t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	$s \geq_{\mathrm{u}} t_s$

Note that s and t, which we have used until now as metavariables denoting arbitrary terms, as in (1), are to be understood as free (i.e., uninterpreted) constants in the two sets above. We call O_r the *restricted* grammar and O_g the *general* grammar.

The selection of bit-vector constants in the grammar turned out to be crucial for finding solutions; for example, 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 $\diamond \in \{\cdot, \bmod, \div, \&, |, \gg, \gg_a, \ll\}$ over relation $\bowtie \in \{\approx, \not\approx, <_{\mathrm{u}}, \leq_{\mathrm{u}}, >_{\mathrm{u}}, \geq_{\mathrm{u}}, <_{\mathrm{s}}, \leq_{\mathrm{s}}, >_{\mathrm{s}}, \geq_{\mathrm{s}}\}$, and used the SyGuS extension of the CVC4 solver [26] 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 involved the smaller number of operator applications. 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, in the sense of containing a smaller number of operator applications, than those we had manually crafted. For example, we manually defined the invertibility condition for $x \cdot s \approx t$ as $(t \approx 0) \vee ((t \,\&\, -t) \geq_{\mathrm{u}} (s \,\&\, -s) \wedge (s \not\approx 0))$. With SyGuS we obtained $((-s \mid s) \,\&\, t) \approx t$. 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

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

Table 6 Conditions for the invertibility of bit-vector operators over signed inequalities $<_{\mathrm{s}}$ and $>_{\mathrm{s}}$ (given modulo commutativity for operators $+$, $\cdot$, & and $|$)

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

$\sim(-s) \mid t \not\approx 0$. 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 and a memory limit of 8GB. Interestingly, the most time-consuming synthesis task (over 107 hours of CPU time) was finding the condition $((t+t)-s)\ \&\ s \geq_{\mathrm{u}} t$ for $s \bmod x \approx t$. A small number of synthesized solutions were correct only for bit-width 4 (we explain how solutions were checked in Sect. 3.4 below). An example incorrect solution is $(\sim s \ll s) \ll s <_{\mathrm{s}} t$ for $x \div s <_{\mathrm{s}} t$. 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.

Table 7 Conditions for the invertibility of bit-vector operators over signed inequalities $\leq_{\mathrm{s}}$ and $\geq_{\mathrm{s}}$ (given modulo commutativity for operators $+$, $\cdot$, & and $|$)

$\ell[x]$	$\leq_{\mathrm{s}}$	$\geq_{\mathrm{s}}$
$x \bowtie t$ $-x \bowtie t$ $\sim x \bowtie t$ $x + s \bowtie t$	$\top$	$\top$
$x \cdot s \bowtie t$	$\sim(s \approx 0 \wedge t <_{\mathrm{s}} s)$	$(-s \mid s)$ & $\max_s \geq_{\mathrm{s}} t$
$x \bmod s \bowtie t$	$\sim 0 <_{\mathrm{s}} -s$ & t	$t <_{\mathrm{s}} s \vee 0 \geq_{\mathrm{s}} s$
$s \bmod x \bowtie t$	$t <_{\mathrm{u}} \min_s \vee t \geq_{\mathrm{s}} s$	$(s \geq_{\mathrm{s}} 0 \Rightarrow s \geq_{\mathrm{s}} t) \wedge$ $((s <_{\mathrm{s}} 0 \wedge t \geq_{\mathrm{s}} 0) \Rightarrow s - t >_{\mathrm{u}} t)$
$x \div s \bowtie t$	$((s \cdot t) \div s \approx t) \vee$ $(t \leq_{\mathrm{s}} 0 \Rightarrow \min_s \div s <_{\mathrm{s}} t)$	$(\sim 0 \div s \geq_{\mathrm{s}} t) \vee (\max_s \div s \geq_{\mathrm{s}} t)$
$s \div x \bowtie t$	$t \geq_{\mathrm{s}} \sim 0 \vee t \geq_{\mathrm{s}} s$	$(s \geq_{\mathrm{s}} 0 \Rightarrow s \geq_{\mathrm{s}} t) \wedge$ $(s <_{\mathrm{s}} 0 \Rightarrow s \gg 1 \geq_{\mathrm{s}} t)$
x & $s \bowtie t$	$s \geq_{\mathrm{u}} t$ & $\min_s$	s & $t \approx t \vee t <_{\mathrm{s}} (t - s)$ & s
$x \mid s \bowtie t$	$t \geq_{\mathrm{s}} s \mid \min_s$	$s \geq_{\mathrm{s}} s$ & t
$x \gg s \bowtie t$	$t \geq_{\mathrm{s}} t \gg s$	$s \not\approx 0 \Rightarrow \sim 0 \gg s \geq_{\mathrm{s}} t$
$s \gg x \bowtie t$	$t <_{\mathrm{u}} \min_s \vee t \geq_{\mathrm{s}} s$	$(s <_{\mathrm{s}} 0 \Rightarrow s \gg 1 \geq_{\mathrm{s}} t) \wedge$ $(s \geq_{\mathrm{s}} 0 \Rightarrow s \geq_{\mathrm{s}} t)$
$x \gg_a s \bowtie t$	$t \geq_{\mathrm{s}} \sim(\max_s \gg s)$	$\max_s \gg s \geq_{\mathrm{s}} t$
$s \gg_a x \bowtie t$	$t \geq_{\mathrm{s}} 0 \vee t \geq_{\mathrm{s}} s$	$t \geq_{\mathrm{u}} \sim t \vee s \geq_{\mathrm{s}} t$
$x \ll s \bowtie t$	$t \gg (t \gg s) <_{\mathrm{u}} \min_s$	$(\max_s \ll s)$ & $\max_s \geq_{\mathrm{s}} t$
$s \ll x \bowtie t$	$t \gg s <_{\mathrm{u}} \min_s$	$\bigvee_{i=0}^{\kappa(s)} (s \ll i) \geq_{\mathrm{s}} t$
$x \circ s \bowtie t$	$t_x \approx \min_s \Rightarrow s \leq_{\mathrm{u}} t_s$	$t_x \approx \max_s \Rightarrow s \geq_{\mathrm{u}} t_s$
	where $t_x = t[\kappa(t) - 1 : \kappa(t) - \kappa(x)], t_s = t[\kappa(s) - 1 : 0]$	
$s \circ x \bowtie t$	$s \leq_{\mathrm{s}} t_s$	$s \geq_{\mathrm{s}} t_s$
	where $t_x = t[\kappa(x) - 1 : 0], t_s = t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	

3.4 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 (quantified) formula

$$\neg(\phi \Leftrightarrow \exists x \,.\, \ell[x])$$

where ℓ ranges over the literals in Tables 3, 4, 5, 6 and 7 with s and t replaced by fresh variables, and ϕ is the corresponding invertibility condition.

In total, we generated 12,980 verification problems. To verify them we used all participating solvers of the quantified bit-vector division of SMT-competition 2017, namely Boolector [19], CVC4, Q3B [16], and Z3 [6]. 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. All solvers that did not time out on a given benchmark agreed on the satisfiability status of that benchmark. We considered 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 $\{\sim, -, +, \&, |, >>, >>_a, <<, \circ\}$, over all relations, and for operators $\{\cdot, \div, \text{mod}\}$ 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 $x \text{ mod } s <_s t$ with a bit-width of 54 and $s \text{ mod } x <_u t$ 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 $\{\cdot, \div, \text{mod}\}$ 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.

Further work Formally proving that our invertibility conditions are correct for *all* bit-widths is beyond the scope of this paper. We have tackled this challenge in other work [23] by recasting the problem to that of checking the unsatisfiability of certain quantified formulas over the combined theory of non-linear integer arithmetic and uninterpreted functions, based on several encodings of bit-vectors in that theory. In that work too, we used SMT solvers to prove the generated formulas. The approach was only partially successful as it failed to verify about a quarter of the invertibility conditions. Most of these remaining conditions, however, were later verified interactively in the Coq proof assistant by Ekici et al. [10], leveraging previous work [9] that had developed a formalization in Coq of the SMT-LIB theory of bit-vectors.

4 Counterexample-guided instantiation for bit-vectors

In this section, we define a novel instantiation-based technique for quantified bit-vector formulas. At a high level, we are interested in determining the satisfiability of a quantified formula φ in some background theory T. We use a *counterexample-guided* approach for quantifier instantiation [27] that adds new instances of φ to a set of quantifier-free clauses based on models for the negation of $\neg\varphi$. The procedure terminates if it constructs a set of instances that is T-unsatisfiable or is T-satisfiable and entails φ.

Approaches for quantifier instantiation crucially depend on having a good strategy for choosing terms to be used in instantiations. We leverage techniques from the previous section to develop one such strategy for the theory of bit-vectors. Specifically, recall that the procedure solve in Fig. 1 returns a symbolic solution to a (linear) bit-vector literal. This procedure uses invertibility conditions for expressing the conditions under which such a solution exists. Intuitively, our quantifier instantiation procedure identifies literals whose solved forms correspond to relevant instantiations, and uses the terms returned by solve to instantiate quantified formulas. These symbolic instantiations are combined as necessary with other instantiation techniques.

Our counterexample-guided approach for quantifier instantiation is exemplified by the procedure $\mathsf{CEGQI}_{\mathcal{S}}$ in Fig. 2. To simplify the exposition here, we focus on input problems

$\mathsf{CEGQI}_{\mathcal{S}}(\forall \boldsymbol{x}.\, \psi[\boldsymbol{x}])$

$\Gamma := \emptyset$

Repeat:

1. If Γ is T-unsatisfiable, then return "unsat".
2. Otherwise, let $\Gamma' = \Gamma \cup \{\neg\psi\}$.
   If Γ' is T-unsatisfiable, then return "sat".
3. Otherwise, let $\mathcal{I}$ be a model of T and Γ' and let $\boldsymbol{t} = \mathcal{S}(\boldsymbol{x}, \psi, \mathcal{I}, \Gamma)$.
   $\Gamma := \Gamma \cup \{\psi[\boldsymbol{t}]\}$.

Fig. 2 A counterexample-guided quantifier instantiation procedure $\mathsf{CEGQI}_{\mathcal{S}}$, parameterized by a selection function $\mathcal{S}$, for determining the T-satisfiability of $\forall \boldsymbol{x}.\, \psi$ with ψ binder-free

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 any theory T with signature Σ and a decidable binder-free (and hence quantifier-free) fragment. The procedure checks via instantiation the T-satisfiability of a quantified input formula φ of the form $\forall \boldsymbol{x}.\, \psi[\boldsymbol{x}]$ where ψ is binder-free.[4] It maintains an evolving set Γ, initially empty, of binder-free instances of the input formula.

During each iteration of the procedure's loop, there are three possible cases:

1. Γ is T-unsatisfiable: then the input formula φ is also T-unsatisfiable and "unsat" is returned;
2. Γ is T-satisfiable but not together with $\neg\psi$, the negated body of φ: then Γ T-entails φ, hence φ is T-satisfiable and "sat" is returned;
3. $\Gamma \cup \{\neg\psi\}$ is T-satisfiable: Γ is extended with an instance of ψ obtained by replacing the variables $\boldsymbol{x}$ with some terms $\boldsymbol{t}$, and the computation continues.

The procedure CEGQI is parametrized by a *selection function* $\mathcal{S}$ that generates the terms $\boldsymbol{t}$.

Definition 7 (Selection Function [28]) A *selection function* takes as input a tuple of variables $\boldsymbol{x}$, a model $\mathcal{I}$ of T, a binder-free Σ-formula $\psi[\boldsymbol{x}]$, and a set Γ of Σ-formulas such that $\boldsymbol{x} \cap FV(\Gamma) = \emptyset$ and $\mathcal{I} \models \Gamma \cup \{\neg\psi\}$. It returns a tuple $\boldsymbol{t}$ of ε-valid terms of the same type as $\boldsymbol{x}$ such that $FV(\boldsymbol{t}) \subseteq FV(\psi) \setminus \boldsymbol{x}$.

Definition 8 Let $\psi[\boldsymbol{x}]$ be a binder-free Σ-formula. A selection function is:

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

Above, we refer to a *legal* input as one where $\mathcal{I}$ is a model for $T \wedge \Gamma \wedge \neg\psi$, and Γ does not contain any free variables in $\boldsymbol{x}$. An invariant of the procedure $\mathsf{CEGQI}_{\mathcal{S}}$ is that it calls $\mathcal{S}$ only with legal inputs. This procedure is refutation-sound and model-sound for any selection function $\mathcal{S}$, and terminating for selection functions that are finite and monotonic.[5] The following theorem is adapted from previous work [28].

[4] Note that ψ may have free variables besides those in $\boldsymbol{x}$ which are then also free variables of φ.

[5] Note that in order for a selection function that is finite on x and ψ to also be monotonic on the same, it must be the case that $\bigwedge_{t \in \mathcal{S}^*} \psi[t]$ T-entails ψ, so that no more legal inputs exist by the time the set $\mathcal{S}^*$ is exhausted.

Theorem 9 (Correctness of $\mathsf{CEGQI}_{\mathcal{S}}$) *Let $\mathcal{S}$ be a selection function and let $\varphi = \forall \boldsymbol{x}.\, \psi$ with ψ-binder-free. 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-satisfiable and T-equivalent to Γ.*
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. Let $\boldsymbol{y} = FV(\psi) \setminus \boldsymbol{x}$. 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 instances of $\psi[\boldsymbol{x}]$ of the form $\psi[\boldsymbol{t}]$ 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 construction, Γ consists of instances $\psi[\boldsymbol{t}]$ of ψ. Since Γ is T-unsatisfiable, it follows by the semantics of $\forall$ that $\forall \boldsymbol{x}.\, \psi[\boldsymbol{x}]$ is T-unsatisfiable.

Part 2) By definition of $\mathsf{CEGQI}_{\mathcal{S}}$, if $\mathsf{CEGQI}_{\mathcal{S}}(\varphi)$ returns "sat," then Γ is T-satisfiable and $\Gamma' = \Gamma \cup \{\neg\psi[\boldsymbol{x}]\}$ is T-unsatisfiable. The latter implies that Γ T-entails $\psi[\boldsymbol{x}]$. Since, by construction of Γ, no variables of $\boldsymbol{x}$ occur in Γ we have that Γ T-entails $\forall \boldsymbol{x}.\, \psi[\boldsymbol{x}]$. The equivalence between the two follows from the fact that, again by construction of Gamma, $\forall \boldsymbol{x}.\, \psi[\boldsymbol{x}]$ T-entails Γ.

Part 3) Assume $\mathcal{S}$ is monotonic and finite for $\psi[\boldsymbol{x}]$. 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.

□

Thanks to this theorem, it suffices to define a selection function satisfying the criteria of Definitions 7 and 8 to define a T-satisfiability procedure for quantified Σ-formulas. We do that in the following section for T_{BV}.

4.1 Selection functions for bit-vectors

In this subsection, we provide several selection functions to be used for bit-vector formulas in the framework described above. Some of these selection functions may return terms containing choice terms based on the procedure solve from Sect. 3. Recall that this procedure uses invertibility conditions to express symbolic solutions to bit-vector literals. There are several possible strategies that one can use in the design of a selection function; we discuss a few alternatives next.

Figure 3 describes a class of selection functions $\mathcal{S}_c^{BV}$ for binder-free bit-vector formulas depending on a *configuration* c. The parameter c ranges over the enumeration type $\{\mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}\}$ whose values are short respectively for "model value", "keep", "slack", and "boundary." We consider multiple configurations because there are many possible choices of terms to use for quantifier instantiation.[6]

The selection function of Fig. 3 collects in the set M all the literals occurring in ψ 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,

which transforms its input literal into a form suitable for procedure solve from Fig. 1. We discuss the intuition for projection operations in more detail below.

[6] We evaluate the effectiveness of these configurations in Sect. 5.

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

Let $M = \{\ell \mid \mathcal{I} \models \ell, \ell \in \mathrm{Lit}(\psi)\}$ and $N = \{\mathsf{project}_c(\mathcal{I}, \ell) \mid \ell \in M\}$

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 $t_i = \begin{cases} \mathsf{solve}(x_i, \mathsf{choose}(N_i)) & \text{if } N_i \text{ is non-empty} \\ k & \text{otherwise, with } k \text{ bit-vector constant s.t. } k^{\mathcal{I}} = x_i^{\mathcal{I}} \end{cases}$

For $j = 1, \ldots, i-1$:

$t_j := t_j\{x_i \mapsto t_i\}$

Return $(t_1, \ldots, t_n)$.

$$\begin{array}{lcl} \mathsf{project}_{\mathbf{m}}(\mathcal{I}, s \bowtie t) & = & \top \\ \mathsf{project}_{\mathbf{k}}(\mathcal{I}, s \bowtie t) & = & s \bowtie t \\ \mathsf{project}_{\mathbf{s}}(\mathcal{I}, s \bowtie t) & = & s \approx t + k \quad \text{with } k \text{ bit-vector constant s.t. } k = (s-t)^{\mathcal{I}} \\ \mathsf{project}_{\mathbf{b}}(\mathcal{I}, s \bowtie t) & = & \begin{cases} s \approx t & \text{if } s^{\mathcal{I}} = t^{\mathcal{I}} \\ s \approx t+1 & \text{if } s^{\mathcal{I}} > t^{\mathcal{I}} \\ s \approx t-1 & \text{if } s^{\mathcal{I}} < t^{\mathcal{I}} \end{cases} \end{array}$$

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

Example 10 Consider the Σ_{BV}-literal $a \geq_{\mathrm{u}} b$ and the interpretation $\mathcal{I}$, where $a^{\mathcal{I}} = 5$ and $b^{\mathcal{I}} = 3$. With input $\mathcal{I}$ and $a \geq_{\mathrm{u}} b$, the function $\mathsf{project}_c$ returns the literals: $\top$ for $c = \mathbf{m}$; $a \geq_{\mathrm{u}} b$ for $c = \mathbf{k}$; $a \approx b + 2$ for $c = \mathbf{s}$; and $a \approx b + 1$ for $c = \mathbf{b}$. In the context of Fig. 3, the above choices impact whether a solved form can be computed for a in the literals returned above, and what that solved form is. In the case of $c = \mathbf{m}$, the literal $\top$ cannot be solved for a, and hence its model value must be selected in the function $\mathcal{S}_c^{BV}$. On the other hand, in the case of $c = \mathbf{b}$, the literal $a \approx b + 1$ leads to the solved form $b + 1$ for a. △

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 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 $\mathsf{linearize}$ 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 $\mathcal{I}$.[7]

Example 11 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 $\{1 \cdot (x + a) \approx b, x \cdot (1 + a) \approx b\}$; $\mathsf{linearize}(x, \mathcal{I}, x \geq_{\mathrm{u}} a)$ returns the singleton set $\{x \geq_{\mathrm{u}} a\}$; $\mathsf{linearize}(x, \mathcal{I}, a \not\approx b)$ returns the empty set. △

If the set N_i is non-empty, the selection function heuristically chooses a literal from N_i, indicated in Fig. 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. By construction of N_i, the chosen literal is guaranteed to be linear in x_i; our heuristic is most effective if the chosen literal is moreover linear invertible in x_i. If N_i is empty, t_i is simply (the constant

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

denoting) 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 configuration of selection function $\mathcal{S}_c^{BV}$ determines 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 **m**, the selection function effectively 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 of variable x_i, for each i. On the other end of the spectrum, the configuration **k** *keeps* all literals in M unchanged. The remaining two configurations have an effect on how disequalities and inequalities are handled by $\mathsf{project}_c$. They are are inspired by quantifier elimination techniques for linear arithmetic [5,17]. With configuration **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 **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 1 to t if s is greater than t in $\mathcal{I}$, it subtracts 1 if s is smaller than t, and returns $s \approx t$ if their value is the same. In the following, we provide an end-to-end example of our technique for quantifier instantiation that makes use of selection function

Example 12 Consider formula $\varphi = \forall x_1. (x_1 \cdot a \leq_{\mathrm{u}} b)$ 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 $\Gamma' = \emptyset \cup \{x_1 \cdot a >_{\mathrm{u}} b\}$[8] 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 $\mathcal{S}_c^{BV}((x_1), \psi, \mathcal{I}, \Gamma)$, where $\psi = (x_1 \cdot a \leq_{\mathrm{u}} b)$, and first compute $M = \{x_1 \cdot a >_{\mathrm{u}} b\}$, the subset of $\mathrm{Lit}(\psi)$ that is satisfied by $\mathcal{I}$. The table below summarizes the values of the internal variables of $\mathcal{S}_c^{BV}$ for the various configurations: In each

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

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

$$((b <_{\mathrm{u}} -a \mid a) \Rightarrow v \cdot a >_{\mathrm{u}} b) \wedge v \cdot a \leq_{\mathrm{u}} b \tag{2}$$

where v is a fresh variable. This formula is T_{BV}-satisfiable if and only if literal $\ell = \neg(b <_{\mathrm{u}} -a \mid a)$ is T_{BV}-satisfiable. In the second iteration of the loop in $\mathsf{CEGQI}_{\mathcal{S}_c^{BV}}$, set Γ contains formula (2) above.

We have two possible outcomes, depending on the satisfiability of ℓ: (i) if ℓ is T_{BV}-unsatisfiable, then (2) and hence Γ are T_{BV}-unsatisfiable, and the procedure terminates with "unsat"; (ii) if ℓ is satisfied by some model $\mathcal{J}$ of T_{BV}, then the formula $\exists z. z \cdot a >_{\mathrm{u}} b$ is false in $\mathcal{J}$, since the invertibility condition of $z \cdot a >_{\mathrm{u}} b$ is false in $\mathcal{J}$. Hence, $\Gamma' = \Gamma \cup \{x_1 \cdot a >_{\mathrm{u}} b\}$ is unsatisfiable, and the algorithm terminates with "sat".

[8] We are using $x_1 \cdot a >_{\mathrm{u}} b$ here instead of $\neg(x_1 \cdot a \leq_{\mathrm{u}} b)$ for conciseness.

As we argue later, quantified bit-vector formulas like φ above, which contain only one occurrence of a universal variable in a linear invertible literal, require at most one instantiation before $\mathsf{CEGQI}_{\mathcal{S}_{\mathbf{k}}^{BV}}$ terminates. The same guarantee does not hold with the other configurations, however. In particular, configuration **m** generates the instance where t_1 is 1, which simplifies to $a \leq_{\mathrm{u}} b$. 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. △

Example 13 As the previous example demonstrates, $\mathsf{CEGQI}_{\mathcal{S}_{\mathbf{k}}^{BV}}$ may terminate after 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 that example to a disjunction of two literals for some literal $\ell[x_1]$: $\forall x_1.\, (x_1 \cdot a \leq_{\mathrm{u}} b \vee \ell[x_1])$. Assume that our selection function chooses the same t_1 as in the previous example. The corresponding instance is equisatisfiable with:

$$((b <_{\mathrm{u}} -a \mid a) \Rightarrow v \cdot a >_{\mathrm{u}} b) \wedge (v \cdot a \leq_{\mathrm{u}} b \vee \ell[v]) \tag{3}$$

with v a fresh variable. In contrast to Example 12, the second iteration of the loop from Fig. 2 is now not guaranteed to terminate. Formula (3) may be satisfied by a model $\mathcal{J}$ where $v \cdot a >_{\mathrm{u}} b$ and $\ell[v]$ hold. Note that $\mathcal{J}$ may also satisfy $b <_{\mathrm{u}} -a \mid a$, meaning it may still be the case that $\neg(x_1 \cdot a \leq_{\mathrm{u}} b)$ together with the above instance is satisfied by $\mathcal{J}$. In such a case, we may invoke $\mathsf{CEGQI}_{\mathcal{S}_{\mathbf{k}}^{BV}}$ 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 $x_1 \cdot a \leq_{\mathrm{u}} b$. By the terminology from Definition 8, this means that the selection function $\mathcal{S}_{\mathbf{k}}^{BV}$ is not monotonic for quantified formulas with more than one occurrence of a universal variable. △

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 Fig. 3 depend on the model value of x_1, and hence more than one possible instance may be considered in loop in Fig. 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 a linear invertible literal with respect to x. We say a selection function is *n-finite* for a quantified formula if the number of possible instantiations it returns is at most n for some positive integer n.

Theorem 14 *Let $\psi[\boldsymbol{x}]$ be a binder-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 $\forall \boldsymbol{x}.\, \psi$ is unit linear invertible.*
4. *$\mathcal{S}_{\mathbf{k}}^{BV}$ is monotonic if $\forall \boldsymbol{x}.\, \psi$ is unit linear invertible.*

Proof Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be a tuple of variables and let ψ be a binder-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 $\mathcal{I} \models \Gamma \cup \{\neg\psi\}$, and let t_1 be $\mathcal{S}_c^{BV}((x_1), \mathcal{I}, \psi, \Gamma)$.

Part 1) To show that $\mathcal{S}_c^{BV}$ is a selection function, we must show that t_1 is ε-valid and $FV(t_1) \subseteq FV(\psi) \setminus \{x_1\}$. 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 $\mathsf{solve}(x_1, \ell')$ where ℓ' is the result of calling $\mathsf{linearize}$ and $\mathsf{project}_c$ on some $\ell \in \mathrm{Lit}(\psi)$. 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 6, and since ℓ' is ε-valid and linear by definition of $\mathsf{linearize}$ and $\mathsf{project}_c$, we have that t_1 is ε-valid and $FV(t_1) \subseteq FV(\ell') \backslash \{x_1\}$. By the definition of $\mathsf{linearize}$ and $\mathsf{project}_c$ for each configuration c, and since each element of M is from $\mathrm{Lit}(\psi)$, we have that $FV(\ell') \subseteq FV(\psi)$. Thus, in either case, we have $FV(t_1) \subseteq FV(\psi) \backslash \{x_1\}$. 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 ψ, note that the number of terms of form $x_1^{\mathcal{I}}$ is finite (because a bit-vector sort only has a finite number of possible values). Also note that the number of literals in $\mathrm{Lit}(\psi)$ is finite. For c and $\ell \in \mathrm{Lit}(\psi)$, the set of literals of the form $\mathsf{project}_c(\mathcal{I}, \ell)$, call this set N', is finite. Now, for any specific $N \subseteq N'$, consider the loop in Fig. 3. The loop consists of substitutions, calls to $\mathsf{linearize}$, choices from N_i, and calls to solve. Given that we start with a finite set, each of these opearations still has only a finite number of possible outcomes. The number of possible return values of $\mathcal{S}_c^{BV}$ is thus finite for (x_1) and ψ for $c = \mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}$.

Part 2) Assume that $\mathsf{project}_{\mathbf{m}}$ is not monotonic for (x_1), meaning that $\psi[t_1] \in \Gamma$. Since $\mathsf{project}_{\mathbf{m}}$ is a selection function by Part 1, and $\mathcal{I}$, Γ are legal inputs to $\mathsf{project}_{\mathbf{m}}$, it must be the case that $\mathcal{I} \models \Gamma \wedge \neg\psi$. So $\mathcal{I} \models \psi[t_1]$. However, since $\mathsf{project}_{\mathbf{m}}(\mathcal{I}, \ell) = \top$, it must be the case that $t_1 = x_1^{\mathcal{I}}$, and we have that $\mathcal{I} \models \neg\psi[x_1]$, which is a contradiction. Hence, $\mathsf{project}_{\mathbf{m}}$ is monotonic for ψ.

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

Part 4) Assume ψ is the linear invertible literal $\ell[x_1]$. 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 $\mathsf{solve}(x_1, \neg\ell[x_1])$. Note that the negation of a linear invertible literal is still linear invertible as the set of relational operators we are using is closed under negation. Thus, by Theorem 6, and since ℓ is linear invertible with respect to x_1, we have that $\neg\ell[t_1] \Leftrightarrow \exists z. \neg\ell[z]$ holds in all models of T_{BV}. Now, assume that $\mathsf{project}_{\mathbf{k}}$ is not monotonic for (x_1) and $\ell[x_1]$, meaning that $\ell[t_1] \in \Gamma$. Since $\mathsf{project}_{\mathbf{k}}$ is a selection function by Part 1 and $\mathcal{I}$, Γ are legal inputs to $\mathsf{project}_{\mathbf{k}}$, it must be the case that $\mathcal{I} \models \Gamma \wedge \neg\psi$. So, $\mathcal{I} \models \ell[t_1]$. But then, 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 $\mathcal{I} \models \neg\ell[x_1]$, which is a contradiction. Hence, it must instead be the case that $\ell[t_1] \notin \Gamma$ and thus $\mathsf{project}_{\mathbf{k}}$ is monotonic for ψ.

□

Theorem 14 implies that the application of counterexample-guided instantiation to arbitrary bit-vector formulas using selection function $\mathcal{S}_{\mathbf{m}}^{BV}$ is a decision procedure for quantified bit-vectors. Unfortunately, the worst-case number of instances considered for a variable $x_{[n]}$ by this selection function is proportional to the number of its possible values (2^n), which makes the decision procedure impractical for sufficiently large n. More interestingly, counterexample-guided instantiation using selection function $\mathcal{S}_{\mathbf{k}}^{BV}$ is a decision procedure for quantified formulas that are unit linear invertible. Moreover, using this selection function has the guarantee that at most one instantiation is returned. 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 $\mathsf{CEGQI}_{\mathcal{S}}$ in Fig. 2. This was demonstrated by the use of this selection function in Example 12.

4.2 Implementation

We implemented the new instantiation techniques described in this section as an extension of CVC4, a DPLL(T)-based SMT solver [24] with support for quantifier-free bit-vector constraints, (arbitrarily nested) quantified formulas, and choice expressions. In CVC4, all choice terms $\varepsilon x.\, \varphi[x]$ are eliminated from assertions by replacing them with a fresh variable v of the same type and adding $\varphi[v]$ as a new assertion. This is sound since all choice expressions we consider are ε-valid. We point out that, since v is fresh, this treatment does *not* enforce that choice terms are unique up to logical equivalence. In other words, $\varepsilon x.\, \varphi[x]$ and $\varepsilon x.\, \psi[x]$ may be replaced by distinct variables even when φ and ψ are logically equivalent. This is done for performance reasons since the correctness of our procedure does not rely on this property. In the following, we discuss important implementation details of this extension.

4.2.1 Handling duplicate instantiations

The selection functions $\mathcal{S}^{BV}_{\mathbf{s}}$ and $\mathcal{S}^{BV}_{\mathbf{b}}$ are not guaranteed to be monotonic and neither is $\mathcal{S}^{BV}_{\mathbf{k}}$ for quantified formulas that are not unit linear invertible. Hence, when applying these strategies to arbitrary quantified formulas, we use a two-tiered strategy that invokes $\mathcal{S}^{BV}_{\mathbf{m}}$ as a second resort if the instance returned by a selection function already exists in Γ.

4.2.2 Linearizing rewrites

Our selection function in Fig. 3 uses the function linearize to compute literals that are linear in the variable x_i. 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, which apply basic algebraic manipulations with the goal of reducing, when possible, the number of occurrences of x_i to one.

As a trivial example, consider a literal $x_i + x_i \approx a$, which can be rewritten to $2 \cdot x_i \approx a$. The rewritten literal is linear in x_i if a does not contain x_i. If that is the case, there exists an invertibility condition for this literal as discussed in Sect. 3.

4.2.3 Variable elimination

We use procedure solve from Sect. 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 $\mathsf{solve}(x, \ell)$ returns a term s not containing ε-expressions, we can replace this formula by $\forall \boldsymbol{y}.\, \varphi[s, \boldsymbol{y}]$. When ℓ is an equality, this is known as *destructive equality resolution (DER)* in the literature and is an important implementation-level optimization in state-of-the-art bit-vector solvers [30].

As shown in Fig. 1, we use the getInverse function to increase the likelihood that solve returns a term that contains no ε-expressions. A common example is a literal $x \cdot c \approx t$ where c is an odd constant and $\kappa(c) = w$. The only solution for x in this case is $c^{-1} \cdot t$, with c^{-1} the (unique) multiplicative inverse of c modulo 2^w, which can be determined with the Extended Euclidean algorithm.

4.2.4 Handling extract

Consider formula $\forall x_{[32]}.\,(x[31:16] \not\approx a_{[16]} \vee x[15:0] \not\approx b_{[16]})$. 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.

4.2.5 Handling propositional structure and nested quantifiers

Figure 2 describes counterexample-guided quantifier instantiation for universal formulas of the form $\forall \boldsymbol{x}.\,\psi$ with ψ quantifier-free. Since ψ can contain additional free variables besides those in $\boldsymbol{x}$, this means effectively that the procedure can deal with formulas with one level of quantifier alternation; specifically, of the form $\exists \boldsymbol{y}.\,\forall \boldsymbol{x}.\,\psi$ where $\boldsymbol{x} \cup \boldsymbol{y} = FV(\psi)$. However, in practice, our techniques can be extended to problems with more than one level of quantifier alternation, and to problems not in prenex normal form. A thorough description of this is beyond the scope of this paper. In the following, we thus only provide some high level details. Further details can be found in Section 6 of [28].

In the DPLL(T) setting, the SMT solver incrementally builds a truth assignment with the goal of finding a set M of literals that is T-satisfiable and, when seen as a truth assignment, propositionally satisfies all the formulas in the input set Δ. The SMT solver considers all quantified formulas $\forall \boldsymbol{x}.\,\psi[\boldsymbol{x}]$ in the current set M, and for each of these formulas, it may add formulas of these forms to Δ:

(i) *instantiation clauses* $A \Rightarrow \psi[\boldsymbol{t}]$
(ii) *Skolemization clauses* $B \Rightarrow \neg\psi[\boldsymbol{v}]$, with $\boldsymbol{v}$ a tuple of fresh variables, and
(iii) *connecting clauses* $\forall \boldsymbol{x}.\,\psi[\boldsymbol{x}] \Rightarrow A$ and $\neg\forall \boldsymbol{x}.\,\psi[\boldsymbol{x}] \Rightarrow B$,

where A and B are fresh Boolean constants, which we call the *positive* and *negative instantiation* guards of $\forall \boldsymbol{x}.\,\psi$. Note that none of these clauses changes the satisfiability of Δ. The second and third kinds of clauses are added once, at the time the quantified formula first occurs in an assignment M. From then on, every occurrence of $\forall \boldsymbol{x}.\,\psi$ in a formula of Δ or M is treated abstractly, as if it were a propositional variable. The solver detects that the negation of a quantified formula along with the current set of clauses Δ is T-unsatisfiable by checking which negative guards must be assigned $\bot$ (false). In practice, this is determined by a decision heuristic which, when faced with the choice of guessing the value of a negative instantiation guard, always tries $\top$ first. When a quantified formula $\forall \boldsymbol{x}.\,\psi$ and its corresponding negative guard both end up being assigned value $\top$, the SMT solver adds an instantiation lemma to Δ for $\forall \boldsymbol{x}.\,\psi$ based on the selection function from Fig. 3. The process terminates as usual when the set Δ is proved T-unsatisfiable, or when the solver finds a T-satisfiable satisfying assignment where either the quantified formula or its negative guard are assigned $\bot$.

This scheme allows us to handle multiple quantified formulas simultaneously. It further allows us to handle quantified formulas with arbitrary nesting by simply allowing formulas like $\psi[\boldsymbol{x}]$ above to contain quantifiers. The resulting quantifiers in instantiation and Skolemization lemmas $A \Rightarrow \psi[\boldsymbol{t}]$ and $B \Rightarrow \neg\psi[\boldsymbol{v}]$ are recursively handled by introducing instantiation and Skolemization lemmas for quantified formulas that appear in subsequent satisfying assignments as a result of processing those lemmas.

4.2.6 Negating the input formula

Our version of counterexample-guided quantifier instantiation is most effective for checking the T-satisfiability of closed universal formulas of the form $\forall \boldsymbol{x}.\, \psi[\boldsymbol{x}]$. In the theory of bit-vectors T_{BV}, for a formula of the form $\exists \boldsymbol{y}.\, \forall \boldsymbol{x}.\, \psi[\boldsymbol{x}, \boldsymbol{y}]$ (with $\boldsymbol{y}$ non-empty) we consider instead the (closed) universal formula corresponding to its negation $\forall \boldsymbol{y}.\, \exists \boldsymbol{x}.\, \neg\psi[\boldsymbol{x}, \boldsymbol{y}]$. Although this does not permit Skolemization of the top-level quantification on $\boldsymbol{y}$, the negated version of this formula may be significantly easier to solve, since our techniques may find an instantiation for $\boldsymbol{y}$ that quickly leads to a proof of unsatisfiability.

Note that checking the satisfiability of the negated input formula is meaningful because T_{BV} is complete, which guarantees that either the formula or its negation is T_{BV}-unsatisfiable. This means that if our procedure determines the negation of a formula is T_{BV}-unsatisfiable, we can conclude the original formula is T_{BV}-satisfiable, and vice versa.

5 Evaluation

We implemented our techniques in the solver **CVC4** and considered four configurations $\mathbf{CVC4}_c$ with $\mathbf{c}$ one of $\{\mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}\}$, corresponding to the four selection function configurations described in Sect. 4. Out of these four configurations, $\mathbf{CVC4_m}$ 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 Sect. 4.2 when applicable. We compared them against all entrants of the quantified bit-vector division of the 2017 SMT competition SMT-COMP: Boolector [19], CVC4 [2], Q3B [16] 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 [29] with a 300 second CPU and wall clock time limit, and a 100 GB memory limit. None of the solvers hit the memory limit on any benchmark.

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 8 and Figs. 4, 5. Configuration $\mathbf{CVC4_b}$ solves the highest number of unsatisfiable benchmarks (4, 399), which is 30 more than the next best configuration $\mathbf{CVC4_s}$ and 37 more than the next best external solver, Z3. Compared to the instantiation-based solvers Boolector, CVC4 and Z3, the performance of $\mathbf{CVC4_b}$ is particularly strong on the h-uauto family, which consists of verification conditions from the Ultimate Automizer tool [13]. For satisfiable benchmarks, Boolector solves the most (581), which is 36 more than our best configuration $\mathbf{CVC4_b}$.

Overall, our best configuration $\mathbf{CVC4_b}$ solved 335 more benchmarks than our base configuration $\mathbf{CVC4_m}$. A more detailed runtime comparison between all configurations is provided by the scatter plots in Fig. 6. We add that $\mathbf{CVC4_b}$ solved 24 more benchmarks than the best external solver, Z3. Looking at uniquely solved instances, $\mathbf{CVC4_b}$ was able to solve 139 benchmarks that were not solved by Z3, whereas Z3 solved 115 benchmarks that $\mathbf{CVC4_b}$ did not. Overall, $\mathbf{CVC4_b}$ 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.

Table 8 Results of the four CVC4 configurations {**m**, **k**, **s**, **b**} and the SMT solvers Boolector, CVC4, Q3B, and Z3 on all 5151 BV benchmarks with a 300 second time limit

	Boolector	**CVC4**	**Q3B**	**Z3**	**CVC4$_m$**	**CVC4$_k$**	**CVC4$_s$**	**CVC4$_b$**
unsat								
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								
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**

The results are grouped by sat/unsat answers, where each row corresponds to a benchmark family. Bold numbers indicate the solver that solved the most instances

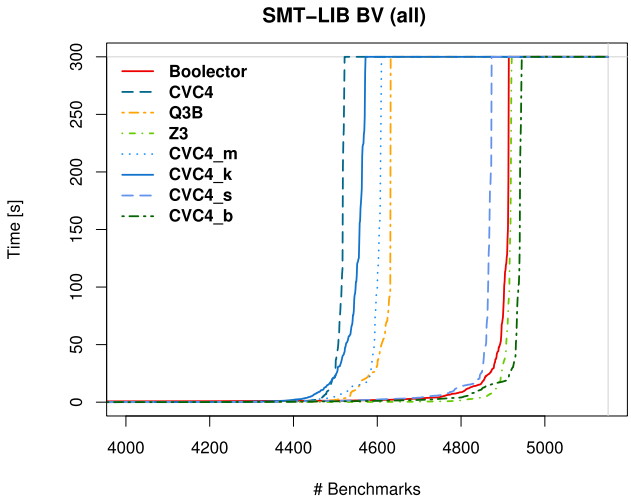


Fig. 4 Results on all BV benchmarks with a 300 second time limit

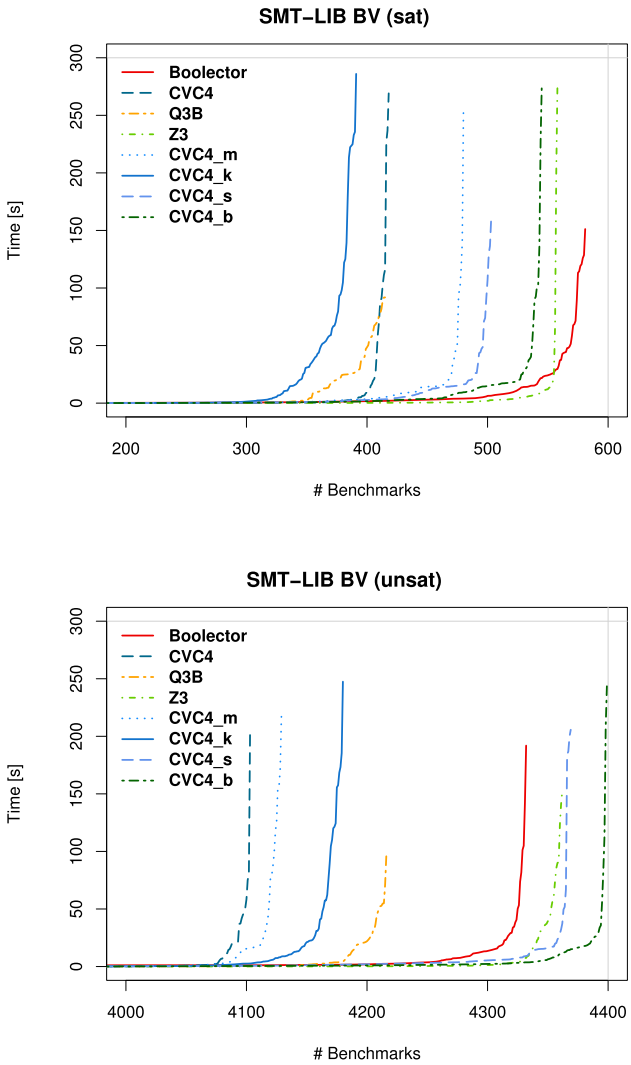
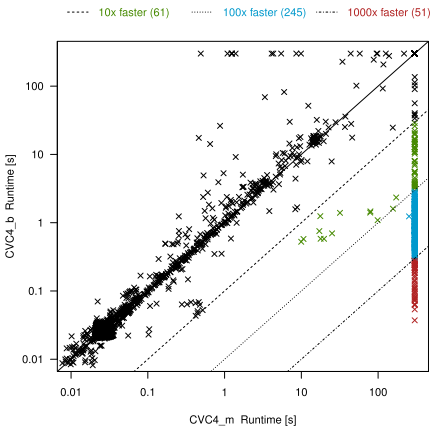


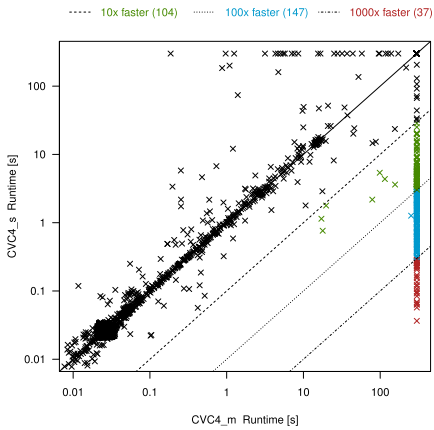
Fig. 5 (Un)satisfiable results on all BV benchmarks with a 300 second time limit

Interestingly, configuration $\mathbf{CVC4_k}$, despite having the strong guarantees given by Theorem 14, 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 considered 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 $\mathbf{CVC4_s}$ and $\mathbf{CVC4_b}$ which use invertibility conditions but in a less monolithic way.

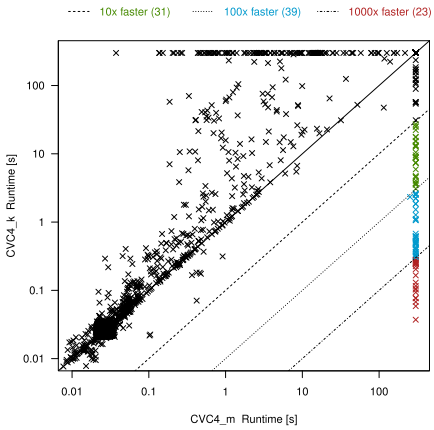
For some intuition on this, consider the problem $\forall x. (x > a \vee x < b)$ where a and b are such that $a > b$ is T_{BV}-valid. Intuitively, to show that this formula is unsatisfiable requires the solver to find an x between b and a. This is apparent when considering the dual problem



(a) Configuration $\mathbf{CVC4_m}$ vs. $\mathbf{CVC4_b}$.



(b) Configuration $\mathbf{CVC4_m}$ vs. $\mathbf{CVC4_s}$.



(c) Configuration $\mathbf{CVC4_m}$ vs. $\mathbf{CVC4_k}$.

Fig. 6 Runtime comparison of base configuration $\mathbf{CVC4_m}$ against configurations $\mathbf{CVC4_b}$, $\mathbf{CVC4_s}$, and $\mathbf{CVC4_k}$

$\exists x.\,(x \leq a \wedge x \geq b)$. Configuration $\mathbf{CVC4_b}$ 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 $\mathbf{CVC4_k}$, on the other hand, would instead consider the instantiation of x for two terms that witness ε-expressions: some k_1 that 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 work on deriving invertibility conditions that are optimal for linear constraints, in the sense of admitting the simplest propositional encoding. We also intend to investigate conditions that cover additional bit-vector operators, some cases of non-linear literals, as well as conditions that cover multiple constraints. While this is a challenging task, we believe that 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 quantifier elimination in bit-vector constraints. This will most likely require a procedure for generating concrete witnesses from choice expressions.

References

1. Alur R, Bodík R, Juniwal G, Martin MMK, Raghothaman M, Seshia SA, Singh R, Solar-Lezama A, Torlak E, Udupa A (2013) Syntax-guided synthesis. In: Formal methods in computer-aided design, FMCAD 2013, Portland, OR, USA, pp 1–8
2. Barrett C, Conway CL, Deters M, Hadarean L, Jovanović D, King T, Reynolds A, Tinelli C (2011) CVC4. In: Proceedings of the 23rd international conference on computer aided verification, CAV'11, pp 171–177. Springer-Verlag. http://dl.acm.org/citation.cfm?id=2032305.2032319
3. Barrett C, Stump A, Tinelli C (2010) 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)
4. Bjørner N, Janota M (2015) Playing with quantified satisfaction. In: 20th International conferences on logic for programming, artificial intelligence and reasoning: short presentations, LPAR 2015, Suva, Fiji, pp 15–27
5. Cooper DC (1972) Theorem proving in arithmetic without multiplication. In: Meltzer B, Michie D (eds) Machine intelligence, vol 7. Edinburgh University Press, Edinburgh, pp 91–100
6. De Moura L, Bjørner N (2008) 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, TACAS'08/ETAPS'08, pp 337–340. Springer. http://dl.acm.org/citation.cfm?id=1792734.1792766
7. Dutertre B (2014) Yices 2.2. In: Proceedings of 26th international conference, computer aided verification, CAV 2014, held as part of the vienna summer of logic, VSL 2014, Vienna, Austria, pp 737–744
8. Dutertre B (2015) Solving exists/forall problems in Yices. Workshop on Satisfiability Modulo Theories
9. Ekici B, Mebsout A, Tinelli C, Keller C, Katz G, Reynolds A, Barrett C (2017) 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
10. Ekici B, Viswanathan A, Zohar Y, Barrett C, Tinelli C (2019) Verifying bit-vector invertibility conditions in Coq (extended abstract). In: Reis G, Barbosa H (eds) Proceedings sixth workshop on proof exchange for theorem proving, electronic proceedings in theoretical computer science, vol 301, pp 57–89. Open Publishing Association. https://doi.org/10.4204/EPTCS.301
11. Enderton HB (2001) A mathematical introduction to logic, 2nd edn. Academic Press, New York
12. Ge Y, de Moura LM (2009) Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: Bouajjani A, Maler O (eds) Proceedings, lecture notes in computer science 21st international conference computer aided verification, CAV 2009, Grenoble, France, June 26–July 2, 2009, vol 5643, pp 306–320. Springer. https://doi.org/10.1007/978-3-642-02658-4_25
13. Heizmann M, Chen Y, Dietsch D, Greitschus M, Nutz A, Musa B, Schätzle C, Schilling C, Schüssele F, Podelski A (2017) Ultimate automizer with an on-demand construction of floyd-hoare automata—(competition contribution). In: A. Legay, T. Margaria (eds) Tools and algorithms for the construction and analysis of systems; Proceedings of 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, Part II, Lecture Notes in Computer Science, vol 10206, pp 394–398. https://doi.org/10.1007/978-3-662-54580-5_30
14. Hilbert D, Bernays P (1934) Grundlagen der Mathematik. Verlag von Julius Springer, Die Grundlehren der mathematischen Wissenschaften
15. John AK, Chakraborty S (2016) A layered algorithm for quantifier elimination from linear modular constraints. Formal Methods Syst Des 49(3):272–323. https://doi.org/10.1007/s10703-016-0260-9
16. Jonás M, Strejcek J (2016) Solving quantified bit-vector formulas using binary decision diagrams. In: Proceedings of 19th international conference theory and applications of satisfiability testing SAT 2016, Bordeaux, France, pp 267–283
17. Loos R, Weispfenning V (1993) Applying linear quantifier elimination
18. Manzano M (1993) Introduction to many-sorted logic. Many-sorted logic and its applications. Wiley, New York, pp 3–86
19. Niemetz A, Preiner M, Biere A (2014 (published 2015)) Boolector 2.0 system description. J Satisfiabil Boolean Model Comput 9: 53–58
20. Niemetz A, Preiner M, Biere A (2016) Precise and complete propagation based local search for satisfiability modulo theories. In: Proceedings of 28th International conference computer aided verification, CAV 2016, Toronto, ON, Canada, Part I, pp 199–217
21. Niemetz A, Preiner M, Biere A (2017) Propagation based local search for bit-precise reasoning. Formal Methods Syst Des 51(3):608–636. https://doi.org/10.1007/s10703-017-0295-6
22. Niemetz A, Preiner M, Reynolds A, Barrett C, Tinelli C (2018) Solving quantified bit-vectors using invertibility conditions. In: Proceedings of the 30th international conference on computer aided verification (CAV 2018), Oxford, pp 236–255. https://doi.org/10.1007/978-3-319-96142-2_16
23. Niemetz A, Preiner M, Reynolds A, Zohar Y, Barrett CW, Tinelli C (2019) Towards bit-width-independent proofs in SMT solvers. In: Fontaine P (ed) Proceedings of the 27th international conference on automated deduction (CADE-27), Lecture Notes in Computer Science, vol 11716, pp 366–384. Springer. https://doi.org/10.1007/978-3-030-29436-6_22
24. Nieuwenhuis R, Oliveras A, Tinelli C (2006) Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). J ACM 53(6):937–977
25. Preiner M, Niemetz A, Biere A (2017) 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
26. Reynolds A, Barbosa H, Nötzli A, Barrett CW, Tinelli C (2019) CVC4SY: Smart and fast term enumeration for syntax-guided synthesis. In: Dillig I, Tasiran S (eds) Proceedings of the 31st international conference on computer aided verification (CAV 2019), Lecture Notes in Computer Science, vol 11562, pp 74–83. Springer. https://doi.org/10.1007/978-3-030-25543-5_5
27. Reynolds A, Deters M, Kuncak V, Tinelli C, Barrett CW (2015) Counterexample-guided quantifier instantiation for synthesis in SMT. In: Proceedings of 27th international conference computer aided verification, CAV 2015, San Francisco, CA, USA, Part II, pp 198–216
28. Reynolds A, King T, Kuncak V (2017) Solving quantified linear arithmetic by counterexample-guided instantiation. Formal Methods Syst Des 51(3):500–532. https://doi.org/10.1007/s10703-017-0290-y
29. Stump A, Sutcliffe G, Tinelli C (2014) 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
30. Wintersteiger CM, Hamadi Y, de Moura LM (2013) Efficiently solving quantified bit-vector formulas. Formal Methods Syst Des 42(1):3–23

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.