Journal of Automated Reasoning (2021) 65:1001–1025
https://doi.org/10.1007/s10817-021-09598-9

Towards Satisfiability Modulo Parametric Bit-vectors

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

Received: 15 April 2020 / Accepted: 22 September 2020 / Published online: 18 June 2021

Abstract
Many SMT solvers implement efficient SAT-based procedures for solving fixed-size bit-vector formulas. These techniques, however, cannot be used directly to reason about bit-vectors of symbolic bit-width. To address this shortcoming, we propose a translation from bit-vector formulas with parametric bit-width to formulas in a logic supported by SMT solvers that includes non-linear integer arithmetic, uninterpreted functions, and universal quantification. While this logic is undecidable, our approach can still solve many formulas that arise in practice by capitalizing on advances in SMT solving for non-linear arithmetic and universally quantified formulas. We provide several case studies in which we have applied this approach with promising results, including the bit-width independent verification of invertibility conditions, compiler optimizations, and bit-vector rewrite rules.

Keywords Satisfiability Modulo Theories · Bit-precise Reasoning · Parametric Bit-vectors

1 Introduction

Satisfiability Modulo Theories (SMT) solving for the theory of fixed-size bit-vectors has received a lot of interest in recent years. Many applications rely on bit-precise reasoning as provided by SMT solvers, and the number of solvers that participate in the corresponding divisions of the annual SMT competition is high and increasing. Although the satisfiability problem in this domain is theoretically difficult (e.g., [19]), bit-vector solvers are in practice highly efficient and typically implement SAT-based procedures.

This work was supported in part by DARPA (Awards N66001-18-C-4012 and FA8650-18-2-7861), ONR (Award N68335-17-C-0558), NSF (Award 1656926), and the Stanford Center for Blockchain Research.
A preliminary version of this work was published in the proceedings of CADE-27 [24]. The current article includes proofs, concrete axiomatizations for bitwise operators, more details on the evaluation, and a list of conditional inverses for bit-vector literals.

✉ Yoni Zohar
yoni206@gmail.com

1 Stanford University, Stanford, USA

2 The University of Iowa, Iowa City, USA

Reasoning about *fixed-size*, i.e., fixed-width, bit-vectors suffices for many applications. For instance, in hardware verification the size of a circuit is usually known in advance. In software verification, machine integers are treated as fixed-width bit-vectors, with the width depending on the underlying architecture. Correspondingly, current solving approaches in SMT rely on this restriction and, as a consequence, cannot reason about parametric circuits or machine integers of arbitrary size. This is a serious limitation when proving properties that are bit-width independent, or when reasoning about machine integers of a fixed but large size. For example, in smart contract languages such as Solidity [33], 256-bit integers are widely used as addresses.

The current state of the art for solving bit-vector formulas involves a technique called *bit-blasting* [20], an eager translation to propositional logic. However, it does not scale well for larger bit-widths, in particular in the presence of operations that have a complex definition at the propositional level, such as multiplication. To address this limitation, we propose a general method for reasoning about bit-vector formulas with parametric bit-width. The essence of our method is to replace the translation from fixed-size bit-vectors to propositional logic with a translation to a quantified fragment of the combined theory of integer arithmetic and uninterpreted functions. We obtain a fully automated verification process by capitalizing on recent advances in SMT solving for these theories.

The reliability of our approach depends on the correctness of the SMT solvers in use. Interactive theorem provers, or proof assistants, such as Isabelle and Coq [10,25], on the other hand, target applications where trust is of higher importance than automation, although substantial progress towards increasing the latter has been made in recent years [5]. Our long-term goal is an efficient automated framework for proving bit-width independent properties within a trusted proof assistant, which requires both a formalization of such properties in the language of the proof assistant and the development of efficient automated techniques to reason about these properties. Our encoding techniques make the latter feasible.

Translating a formula from the theory of parametric-width bit-vectors to the theory of integer arithmetic is not straightforward. This is due to the fact that the semantics of bit-vector operations of bit-width n are most naturally expressed using exponentiation terms 2^n. Most SMT solvers, however, do not support unrestricted exponentiation for integer arithmetic. Further, operators such as bitwise *and* and *or* do not have a natural representation in integer arithmetic. While they are definable in the theory of integer arithmetic using β-function encodings (e.g., [15]), such a translation is expensive as it requires an encoding of sequences into natural numbers. Instead, we introduce an uninterpreted function (UF) for each of these problematic operators and axiomatize them with quantified formulas, which shifts some of the reasoning burden from arithmetic to UF reasoning. We consider two alternative axiomatization approaches: a complete one relying on induction, and a partial (hand-crafted) one that can be understood as an under-approximation of the original problem.

To evaluate the potential of our approach, we examine three case studies that arise from real applications where reasoning about bit-width independent properties is essential. Niemetz et al. [23] introduced the notion of *invertibility condition* for bit-vector operators, and defined a large number of such conditions which they then used in a new procedure for quantified bit-vector formulas. Because of the fixed-size setting, the correctness of those conditions was only checked for specific range of bit-widths: from 1 to 65. As a first case study, we consider here the bit-width independent verification of those invertibility conditions, which Niemetz et al. [23] left to future work. As a second case study, we examine the bit-width independent verification of compiler optimizations in LLVM. For that, we use the Alive tool [22], which generates verification conditions for such optimizations in the theory of fixed-size bit-vectors. Proving the correctness of these optimizations for arbitrary bit-widths

Table 1 Considered bit-vector operators with SMT-LIB 2 syntax

Symbol	SMT-LIB Syntax	Sort
$\approx, \not\approx$	$=$, distinct	$\sigma_{[n]} \times \sigma_{[n]} \to \mathsf{Bool}$
$<_{\mathrm{u}}^{\mathrm{BV}}, >_{\mathrm{u}}^{\mathrm{BV}}, <_{\mathrm{s}}^{\mathrm{BV}}, >_{\mathrm{s}}^{\mathrm{BV}}$	bvult, bvugt, bvslt, bvsgt	$\sigma_{[n]} \times \sigma_{[n]} \to \mathsf{Bool}$
$\leq_{\mathrm{u}}^{\mathrm{BV}}, \geq_{\mathrm{u}}^{\mathrm{BV}}, \leq_{\mathrm{s}}^{s\mathrm{BV}}, \geq_{\mathrm{s}}^{\mathrm{BV}}$	bvule, bvuge, bvsle, bvsge	$\sigma_{[n]} \times \sigma_{[n]} \to \mathsf{Bool}$
$\sim^{\mathrm{BV}}, -^{\mathrm{BV}}$	bvnot, bvneg	$\sigma_{[n]} \to \sigma_{[n]}$
$\&^{\mathrm{BV}}, \mid^{\mathrm{BV}}, \oplus^{\mathrm{BV}}$	bvand, bvor, bvxor	$\sigma_{[n]} \times \sigma_{[n]} \to \sigma_{[n]}$
$<<^{\mathrm{BV}}, >>^{\mathrm{BV}}, >>_{a}^{\mathrm{BV}}$	bvshl, bvlshr, bvashr	$\sigma_{[n]} \times \sigma_{[n]} \to \sigma_{[n]}$
$+^{\mathrm{BV}}, .^{\mathrm{BV}}, \mathrm{mod}^{\mathrm{BV}}, \mathrm{div}^{\mathrm{BV}}$	bvadd, bvmul, bvurem, bvudiv	$\sigma_{[n]} \times \sigma_{[n]} \to \sigma_{[n]}$
$[u:l]^{\mathrm{BV}}$	extract $(0 \leq l \leq u < n)$	$\sigma_{[n]} \to \sigma_{[u-l+1]}$
$\circ^{\mathrm{BV}}$	concatenation	$\sigma_{[n]} \times \sigma_{[m]} \to \sigma_{[n+m]}$

would ensure their correctness for any language or underlying architecture rather than just for specific ones. As a third case study, we consider the bit-width independent verification of rewrite rules for the theory of fixed-size bit-vectors. SMT solvers for this theory heavily rely on such rules to simplify the input. Verifying their correctness is essential and is typically done by hand, which is both tedious and error-prone.

To summarize, this paper makes the following contributions.

- In Sect. 4, we study complete and incomplete encodings of bit-vector formulas with parametric bit-width into integer arithmetic.
- In Sect. 5, we evaluate the effectiveness of these encodings in three case studies.
- As part of the first case study, we introduce *conditional inverses* for bit-vector constraints, thus augmenting the work of [23] with concrete parametric solutions.

1.1 Related Work

Bit-width independent bit-vector formulas were studied by Picora [27], who introduced a formal language for bit-vectors of parametric width, together with semantics and a decision procedure. The language we use here is a simplified variant of that language. A unification-based algorithm for bit-vectors of symbolic lengths is discussed by Bjørner and Picora in [4]. Bit-width independent formulas are related to parametric Boolean functions and circuits. An inductive approach for reasoning about such formalisms was developed by Gupta and Fisher [16,17] by considering a Boolean function for the base case of a circuit and another one for its inductive step. Reasoning about equivalence of such circuits can be embedded in Picora's formal framework of [27]. Our technique is based on a translation of parametric bit-vectors to integers. For the case of fixed-width bit-vectors, translations of arithmetic bit-vector operations to integers have been studied, e.g., in [6,7,36].

2 Preliminaries

We briefly review the usual notions and terminology of many-sorted first-order logic with equality. See [15,34] for more detailed information.

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. A *signature* Σ consists of a set $\Sigma^s \subseteq S$ of sort symbols and a set Σ^f of function symbols. Arities of function symbols are defined in the usual many-sorted way. Constants are treated as nullary functions. We assume that every signature Σ includes a Boolean sort Bool and two constant symbols $\top$ (true) and $\bot$ (false) of sort Bool. Furthermore, we assume that Σ has an equality symbol $\approx$ of arity $\sigma \times \sigma \to \mathsf{Bool}$ for each sort $\sigma \in \Sigma^s$. We call *predicate* symbol any function symbol whose return sort is Bool. We use the usual definitions of well-sorted terms, literals, and formulas as terms of sort Bool, and refer to them as Σ-terms, Σ-literals, and Σ-formulas, respectively. We include the *if–then–else* operator $\mathrm{ite}(\_, \_, \_)$ of arity $\mathsf{Bool} \times \sigma \times \sigma \to \sigma$ for each $\sigma \in \Sigma^s$.

We define $\boldsymbol{x} = (x_1, ..., x_n)$ as a tuple of variables and write $Q\boldsymbol{x}\varphi$ with $Q \in \{\forall, \exists\}$ for a *quantified* formula $Qx_1 \cdots Qx_n\varphi$. For a Σ-term s, we denote the *free variables* of s (defined as usual) as $\mathrm{FV}(s)$ and use $s[\boldsymbol{x}]$ to denote that the free variables of s occur in $\boldsymbol{x}$; if Σ-terms $\boldsymbol{t} = (t_1, ..., t_n)$ is a tuple of terms we write $s[\boldsymbol{t}]$ for the term obtained from s by simultaneously replacing each occurrence of x_i in s by t_i. A *Σ-interpretation* $\mathcal{I}$ maps: each sort $\sigma \in \Sigma^s$ to a distinct non-empty set of values $\sigma^\mathcal{I}$ (the *domain* of σ in $\mathcal{I}$); each variable $x \in X_\sigma$ to an element $x^\mathcal{I} \in \sigma^\mathcal{I}$; and each function symbol $f^{\sigma_1 \cdots \sigma_n \sigma} \in \Sigma^f$ to a total function $f^\mathcal{I}\colon \sigma_1^\mathcal{I} \times ... \times \sigma_n^\mathcal{I} \to \sigma^\mathcal{I}$ if $n > 0$, and to an element in $\sigma^\mathcal{I}$ if $n = 0$. We use the usual inductive definition of a satisfiability relation $\models$ between Σ-interpretations and Σ-formulas. Given a Σ-interpretation $\mathcal{I}$ and a sub-signature Σ' of Σ, the *reduct* of $\mathcal{I}$ to Σ' is obtained from $\mathcal{I}$ by restricting it to the sorts and symbols of Σ'.

A *theory* T is a pair (Σ, I), where Σ is a signature and I is a non-empty class of Σ-interpretations that is closed under variable reassignment, i.e., if interpretation $\mathcal{I}'$ only differs from an $\mathcal{I} \in I$ in how it interprets variables, then also $\mathcal{I}' \in I$. Moreover, each interpretation in I interprets $\approx$ as the identity relation and Bool as the two-element set $\{\top^\mathcal{I}, \bot^\mathcal{I}\}$ and interprets a term of the form $\mathrm{ite}(c, s, t)$ as $s^\mathcal{I}$ if $c^\mathcal{I} = \top^\mathcal{I}$ and as $t^\mathcal{I}$ otherwise. A Σ-formula φ is *T-satisfiable* (resp. *T-unsatisfiable*) if it is satisfied by some (resp. no) interpretation in I; it is *T-valid* if it is satisfied by all interpretations in I. We will sometimes omit T when the theory is understood from context.

The theory $T_{\mathrm{BV}} = (\Sigma_{\mathrm{BV}}, I_{\mathrm{BV}})$ of fixed-size bit-vectors as defined in the SMT-LIB 2 standard [3] consists of the class of interpretations I_{BV} and signature Σ_{BV}, which includes a unique sort for each positive integer n (representing the bit-vector width), denoted here as $\sigma_{[n]}$. We consider a restricted set of bit-vector function and predicate symbols (or *bit-vector operators*) 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. For a given positive integer n, the domain $\sigma_{[n]}^\mathcal{I}$ of sort $\sigma_{[n]}$ in $\mathcal{I}$ is the set of all bit-vectors of size n. We assume that Σ_{BV} includes all *bit-vector constants* of sort $\sigma_{[n]}$ for each n, represented as bit-strings. However, to simplify the notation we will sometimes denote them by the corresponding natural number in $\{0, \ldots, 2^{n-1}\}$. All interpretations $\mathcal{I} \in I_{\mathrm{BV}}$ are identical except for the value they assign to variables. They interpret sort and function symbols as specified in SMT-LIB 2. In particular, all function symbols (of non-zero arity) in Σ_{BV}^f are overloaded for every $\sigma_{[n]} \in \Sigma_{\mathrm{BV}}^s$. We denote a Σ_{BV}-term (or *bit-vector term*) t of width n as $t_{[n]}$ when we want to specify its bit-width explicitly.

The SMT-LIB 2 definition of division by zero is worth mentioning. The expression $s\mathrm{div}^{\mathrm{BV}}0$ is defined as evaluating to a bit-vector with the same bit-width as s with all bits set to 1. Similarly, expression $s\mathrm{mod}^{\mathrm{BV}}0$ is defined as evaluating to s. Details on these semantics and their justification are given in the latest version of the SMT-LIB 2 standard.

We refer to the ith bit of $t_{[n]}$ as $t[i]$ with $0 \leq i < n$. We interpret $t[0]$ as the least significant bit (LSB), and $t[n-1]$ as the most significant bit (MSB), and denote bit ranges over k from index j down to i as $t[j:i]$. The unsigned interpretation of a bit-vector $t_{[n]}$ as a natural number is given by $[t]_{\mathbb{N}} = \Sigma_{i=0}^{n-1} t[i] \cdot 2^i$, and its signed interpretation as an integer is given by $[t]_{\mathbb{Z}} = -t[n-1] \cdot 2^{n-1} + \left[t[n-2:0]^{\mathrm{BV}}\right]_{\mathbb{N}}$.

The theory $T_{\mathrm{IA}} = (\Sigma_{\mathrm{IA}}, I_{\mathrm{IA}})$ of integer arithmetic is also defined as in the SMT-LIB 2 standard. The signature Σ_{IA} includes a single sort Int, function and predicate symbols $\{+, -, \cdot, \div, \mathrm{mod}, |\_|, <, \leq, >, \geq\}$, and a constant symbol for every non-negative integer value. We further extend Σ_{IA} to include exponentiation, denoted in the usual way as a^b. Unlike in the theory of bit-vectors, $\div$ and mod are *partial* functions, as they are not defined when their second argument is 0. Their semantics in this case is left unspecified in the SMT-LIB 2 standard, and they are thus treated as uninterpreted functions. Hence, I_{IA} is the set of all Σ_{IA}-interpretations $\mathcal{I}$ that interpret the arithmetic operators as the usual integer operators, and differ only for the values they assign to variables and to $\div$ and mod when their second argument is zero. For a set $\mathcal{F}$ of function symbols whose arities have the form $\mathsf{Int} \times \cdots \times \mathsf{Int} \to \mathsf{Int}$, we write $T^{\mathcal{F}}_{\mathrm{UFIA}}$ to denote the (combined) theory of uninterpreted functions of $\mathcal{F}$ with integer arithmetic. Its signature $\Sigma^{\mathcal{F}}_{\mathrm{UFIA}}$ is the union of the signature of T_{IA} with a disjoint signature containing the set $\mathcal{F}$ of function symbols, called *uninterpreted functions*. $T^{\mathcal{F}}_{\mathrm{UFIA}}$ consists of all $\Sigma^{\mathcal{F}}_{\mathrm{UFIA}}$-interpretations whose reduct to the signature Σ_{IA} is an interpretation of T_{IA}.

3 Parametric Bit-Vector Formulas

We are interested in reasoning about (classes of) Σ_{BV}-formulas that hold independently from the specific width of the sorts assigned to their variables or terms. We formalize the notion of *parametric* Σ_{BV}-formulas in the following.

We fix two disjoint sets X^* and Z^* of variable and constant symbols, respectively, of a new bit-vector sort of undetermined bit-width. The bit-width is provided by the first component of a separate function pair $\omega = (\omega^b, \omega^N)$ which maps symbols $\mathsf{x} \in X^* \cup Z^*$ to Σ_{IA}-terms without $\div$ and mod. These operators (which we did not encounter in mappings ω of practical problems) are easily expressible by introducing more variables and using multiplication and addition, without having to deal with issues like division by zero. We refer to $\omega^b(\mathsf{x})$ as the *symbolic bit-width* assigned by ω to x. The second component of ω is a map ω^N from symbols $\mathsf{z} \in Z^*$ to Σ_{IA}-terms without $\div$ and mod. We call $\omega^N(\mathsf{z})$ the *symbolic value* assigned by ω to z. Let $\boldsymbol{v} = \mathrm{FV}(\omega)$ be the set of free (integer) variables occurring in the range of either ω^b or ω^N. We say that ω is *admissible* if for every interpretation $\mathcal{I} \in I_{\mathrm{IA}}$ that interprets each variable in $\boldsymbol{v}$ as a positive integer, and for every $\mathsf{x} \in X^* \cup Z^*$, $\mathcal{I}$ also interprets $\omega^b(\mathsf{x})$ as a positive integer.

Let t be a term (possibly a formula), built from the function symbols of Σ_{BV} and $X^* \cup Z^*$, ignoring their sorts. We refer to t as a *parametric Σ_{BV}-term (formula)*. One can interpret t as a class of fixed-size bit-vector terms as follows. For each symbol $\mathsf{x} \in X^*$ and integer $n > 0$, we associate a unique variable x_n of (fixed) bit-vector sort $\sigma_{[n]}$. Given an admissible ω with $\boldsymbol{v} = \mathrm{FV}(\omega)$ and an interpretation $\mathcal{I}$ that maps each variable in $\boldsymbol{v}$ to a positive integer, let $t|_{\omega[\mathcal{I}]}$ be the result of replacing all symbols $\mathsf{x} \in X^*$ in t by the corresponding bit-vector variable $x_{[k]}$ and all symbols $\mathsf{x} \in Z^*$ in t by the bit-vector constant of sort $\sigma_{[k]}$ corresponding to $\omega^N(\mathsf{x})^{\mathcal{I}} \bmod 2^k$, where in both cases k is $\omega^b(\mathsf{x})^{\mathcal{I}}$. We say that t is *well-sorted under* ω if ω is admissible and $t|_{\omega[\mathcal{I}]}$ is a well-sorted Σ_{BV}-term for all $\mathcal{I}$ that map variables in $\boldsymbol{v}$ to positive

integers. Note that in the most general case, well-sortedness and admissibility conditions may include non-linear multiplication and even exponentiation, which makes the problem of determining them undecidable. As we shall see in Sect. 5, however, there are interesting benchmarks that occur in practice and induce very simple conditions that can be trivial to check.

Example 1 Let X^* be the set $\{\mathsf{x}\}$ and Z^* be the set $\{\mathsf{z}_0, \mathsf{z}_1\}$, where $\omega^N(\mathsf{z}_0) = 0$ and $\omega^N(\mathsf{z}_1) = 1$. Let φ be the formula $(\mathsf{x} +^{\mathrm{BV}} \mathsf{x}) +^{\mathrm{BV}} \mathsf{z}_1 \not\approx \mathsf{z}_0$. We have that φ is well-sorted under (ω^b, ω^N) with $\omega^b = \{\mathsf{x} \mapsto a, \mathsf{z}_0 \mapsto a, \mathsf{z}_1 \mapsto a\}$ or $\omega^b = \{\mathsf{x} \mapsto 3, \mathsf{z}_0 \mapsto 3, \mathsf{z}_1 \mapsto 3\}$. It is not well-sorted when $\omega^b = \{\mathsf{x} \mapsto a_1, \mathsf{z}_0 \mapsto a_1, \mathsf{z}_1 \mapsto a_2\}$ since $\varphi|_{\omega[\mathcal{I}]}$ is not a well-sorted Σ_{BV}-formula whenever $a_1^{\mathcal{I}} \neq a_2^{\mathcal{I}}$. Note that an ω where $\omega^b(\mathsf{x}) = a_1 - a_2$ is not admissible, since $(a_1 - a_2)^{\mathcal{I}} \leq 0$ is possible even when $a_1^{\mathcal{I}} > 0$ and $a_2^{\mathcal{I}} > 0$.

Notice that symbolic constants such as the maximum unsigned constant of a symbolic length w can be represented by introducing $\mathsf{z} \in Z^*$ with $\omega^b(\mathsf{z}) = w$ and $\omega^N(\mathsf{z}) = 2^w - 1$. Furthermore, recall that signature Σ_{BV} includes the (postfix) bit-vector extract operator $[u : l]^{\mathrm{BV}}$ whose name is parameterized by two natural numbers u and l. We do not lift the above definitions to handle extract operations. This is for simplicity and comes at no loss of expressive power, since constraints involving extract can be equivalently expressed using constraints involving concatenation. For example, showing that every instance of a constraint $s \approx t[u : l]^{\mathrm{BV}}$ is satisfiable, where $0 < l \leq u < n - 1$, is equivalent to showing that $\forall y_1 \forall y_2 \forall y_3 (t \approx y_1 \circ^{\mathrm{BV}} (y_2 \circ^{\mathrm{BV}} y_3) \Rightarrow s \approx y_2)$ is satisfiable where y_1, y_2, y_3 are fresh variables of sort $\sigma_{[n-1-u]}$, $\sigma_{[u-l+1]}$, $\sigma_{[l]}$, respectively. We may reason about a formula involving a symbolic range $\{l, \ldots, u\}$ of t by considering a parametric bit-vector formula that encodes a formula of the latter form, where the appropriate symbolic bit-widths are assigned to symbols introduced for y_1, y_2, y_3.

For every admissible ω, we extend ω to bit-vector terms t that are well sorted under ω so that $t|_{\omega[\mathcal{I}]}$ has sort $\sigma_{[\omega^b(t)^{\mathcal{I}}]}$ for all interpretations $\mathcal{I}$ that map variables in $\mathrm{FV}(\omega)$ to positive integers. Intuitively, $\omega(t)$ is computed recursively by computing ω for each immediate subterm of t and then applying the typing rules of the operators in Σ_{BV}. The formal definition of ω^b is by induction on terms. The base cases are already defined. $\omega^b(\diamond t) = \omega^b(t)$ for $\diamond \in \{-^{\mathrm{BV}}, \sim^{\mathrm{BV}}\}$. For the binary operators of Table 1 (except for extraction which can be eliminated as described above), we set $\omega^b(\diamond(t_1, t_2)) = \omega^b(t_2)$ if $\diamond$ is not $\circ^{\mathrm{BV}}$. For concatenation, $\omega^b(t_1 \circ^{\mathrm{BV}} t_2) = \omega^b(t_1) + \omega^b(t_2)$. In turn, ω^N is extended to complex terms by evaluating them according to the semantics given by the SMT-LIB 2 standard. For example, $\omega^N(t_1 +^{\mathrm{BV}} t_2)$ is $\omega^N(t_1) + \omega^N(t_2) \bmod 2^{\omega^b(t_2)}$.

Finally, we extend the notion of validity to parametric bit-vector formulas. Given a formula φ well sorted under ω, we say φ is T_{BV}-*valid* under ω if $\varphi|_{\omega[\mathcal{I}]}$ is T_{BV}-valid for all $\mathcal{I}$ that map variables in $\mathrm{FV}(\omega)$ to positive integers.

4 Encoding Parametric Bit-Vector Formulas in SMT

Current SMT solvers do not support reasoning about parametric bit-vector formulas. In this section, we present a technique for encoding such formulas into formulas involving non-linear integer arithmetic, uninterpreted functions, and universal quantifiers. In SMT-LIB parlance, these are formulas in the UFNIA logic. Given a quantifier-free formula φ that is well-sorted under some mapping ω, we describe this encoding in terms of a translation $\mathcal{T}$, which returns a formula ψ that is valid in the theory of uninterpreted functions with integer arithmetic only

if φ is T_{BV}-valid under ω. We describe several variations on this translation and discuss their relative strengths and weaknesses.

4.1 Introducing the Encoding

4.1.1 Overall Approach

At a high level, our translation produces an implication whose antecedent requires the integer variables to be in the correct ranges (e.g., $k > 0$ for every bit-width variable k), and whose conclusion is the result of converting each (parametric) bit-vector term of bit-width k to an integer term. Operations on parametric bit-vector terms are converted to operations on the integers modulo 2^k, where k can be a symbolic (i.e., uninterpreted) constant. We first introduce uninterpreted functions that will be used in our translation. Note that SMT solvers may not support the full set of functions in our extended signature Σ_{IA}, since they typically do not support exponentiation. Our translation, however, requires a limited form of exponentiation. Therefore, we introduce an uninterpreted function symbol $\mathsf{pow2}$ of arity $\mathsf{Int} \to \mathsf{Int}$, whose intended semantics is the function $\lambda x.2^x$ when the argument x is non-negative. Second, for each (non-predicate) n-ary function f^{BV} (with $n > 0$) of arity $\sigma_1 \times \cdots \times \sigma_n \to \sigma$ in the signature of fixed-size bit-vectors Σ_{BV} (excluding bit-vector extraction), we introduce an uninterpreted function $f^{\mathbb{N}}$ of arity $\mathsf{Int} \times \mathsf{Int} \times \cdots \times \mathsf{Int} \to \mathsf{Int}$, where the extra argument is used to specify the bit-width. For example, for $+^{\mathrm{BV}}$ with arity $\sigma_{[n]} \times \sigma_{[n]} \to \sigma_{[n]}$, we introduce $+^{\mathbb{N}}$ of arity $\mathsf{Int} \times \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$. In its intended semantics, this function adds the second and third arguments, both integers, and returns the result modulo 2^k, where k is the first argument. The signature Σ_{BV} contains one function, bit-vector concatenation $\circ^{\mathrm{BV}}$, whose two arguments may have different sorts. For this case, the first argument of $\circ^{\mathbb{N}}$ indicates the bit-width of the third argument, i.e., $\circ^{\mathbb{N}}(k, x, y)$ is interpreted as the concatenation of x and y, where y is an integer that encodes a bit-vector of bit-width k; the bit-width for x is not specified by an argument, as it is not needed for the elimination of this operator we perform later. We introduce uninterpreted functions for each bit-vector predicate symbol in a similar fashion. For instance, ${\geq_u}^{\mathbb{N}}$ has arity $\mathsf{Int} \times \mathsf{Int} \times \mathsf{Int} \to \mathsf{Bool}$ and encodes whether its second argument is greater than or equal to its third argument when these two arguments are interpreted as unsigned bit-vector values whose bit-width is given by its first argument. Most of these uninterpreted functions are eliminated as described below. However, $\mathsf{pow2}$, $\&^{\mathbb{N}}$, $|^{\mathbb{N}}$ and $\oplus^{\mathbb{N}}$ are not eliminated. Depending on the variation of the encoding, our translation may introduce quantified formulas that fully axiomatize the behavior of these remaining uninterpreted functions or add (quantified) lemmas that state some key properties about them, or both.

4.1.2 Translation Function

Figure 1 defines our translation function $\mathcal{T}_A$, which is parameterized by an axiomatization mode A. Given an input formula φ that is well-sorted under ω, it returns the implication whose antecedent is an *axiomatization* formula $\mathrm{AX}_A(\varphi, \sigma)$ and whose conclusion is the result of converting φ to its encoded version via the conversion function Conv. The former is dependent upon the axiomatization mode A which we discuss later. We assume without loss of generality that φ contains no applications of bit-vector extract, which can be eliminated as described in the previous section, nor does it contain concrete bit-vector constants, since these can be equivalently represented by introducing a symbol in Z^* with the appropriate concrete

$\mathcal{T}_A(\varphi, \omega)$:
  Return $\mathrm{AX}_A(\varphi, \omega) \Rightarrow \textsc{Conv}(\varphi, \omega)$.

$\textsc{Conv}\,(e, \omega)$:
  Match e:

x	$\rightarrow$	$\chi(\mathsf{x})$	if $\mathsf{x} \in X^*$
z	$\rightarrow$	$\omega^N(\mathsf{z}) \bmod \mathsf{pow2}(\omega^b(\mathsf{z}))$	if $\mathsf{z} \in Z^*$
$t_1 \approx t_2$	$\rightarrow$	$\textsc{Conv}(t_1, \omega) \approx \textsc{Conv}(t_2, \omega)$	
$f^{\mathrm{BV}}(t_1, \ldots, t_n)$	$\rightarrow$	$\textsc{Elim}(f^{\mathbb{N}}(\omega^b(t_n), \textsc{Conv}(t_1, \omega), \ldots, \textsc{Conv}(t_n, \omega)))$	
$\bowtie(\varphi_1, \ldots, \varphi_n)$	$\rightarrow$	$\bowtie(\textsc{Conv}(\varphi_1, \omega), \ldots, \textsc{Conv}(\varphi_n, \omega))$	$\bowtie \in \{\wedge, \vee, \Rightarrow, \neg, \Leftrightarrow\}$

$\textsc{Elim}\,(e)$:
  Match e:

$+^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$(x + y) \bmod \mathsf{pow2}(k)$	
$-^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$(x - y) \bmod \mathsf{pow2}(k)$	
$\cdot^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$(x \cdot y) \bmod \mathsf{pow2}(k)$	
$\mathrm{div}^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$\mathrm{ite}(y \approx 0, \mathsf{pow2}(k) - 1, x \operatorname{div} y)$	
$\mathrm{mod}^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$\mathrm{ite}(y \approx 0, x, x \bmod y)$	
$\sim^{\mathbb{N}}(k, x)$	$\rightarrow$	$\mathsf{pow2}(k) - (x + 1)$	
$-^{\mathbb{N}}(k, x)$	$\rightarrow$	$(\mathsf{pow2}(k) - x) \bmod \mathsf{pow2}(k)$	
$\ll^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$(x \cdot \mathsf{pow2}(y)) \bmod \mathsf{pow2}(k)$	
$\gg^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$(x \operatorname{div} \mathsf{pow2}(y)) \bmod \mathsf{pow2}(k)$	
$\circ^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$x \cdot \mathsf{pow2}(k) + y$	
$\bowtie_u^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$x \bowtie y$	$\bowtie \in \{<, \leq, >, \geq\}$
$\bowtie_s^{\mathbb{N}}(k, x, y)$	$\rightarrow$	$\mathsf{uts}_k(x) \bowtie \mathsf{uts}_k(y)$	$\bowtie \in \{<, \leq, >, \geq\}$

$\mathsf{uts}_k(x)$:
  Return $2 \cdot (x \bmod \mathsf{pow2}(k-1)) - x$.

Fig. 1 Translation $\mathcal{T}_A$ for parametric bit-vector formulas, parameterized by axiomatization mode A

mappings in ω^b and ω^N. For simplicity, we present only the case where φ is quantifier-free. Quantifiers can be handled in the expected way: every bound bit-vector variable is replaced with a bound integer variable. The needed range-constraints are then added conjunctively to the matrix of an existential quantifier, and as a premise of an implication in the case of a universal quantifier.[1]

In the translation, we use an auxiliary function Conv which converts parametric bit-vector expressions into integer expressions with uninterpreted functions. Parametric bit-vector variables x (that is, symbols from X^*) are replaced by unique integer variables of sort Int, where we assume a mapping χ that maintains this correspondence and is such that the range of χ does not include any variable occurring in $\mathrm{FV}(\omega)$. Parametric bit-vector constants z (that is, symbols from set Z^*) are replaced by the term $\omega^N(\mathsf{z}) \bmod \mathsf{pow2}(\omega^b(\mathsf{z}))$. Observe that the ranges of the maps in ω may contain arbitrary Σ_{IA}-symbols (other than $\div$ and $\bmod$). In practice, however, our translation handles only cases where these terms contain symbols supported by the SMT solver, as well as terms of the form 2^t, which we assume are replaced by $\mathsf{pow2}(t)$ during this translation. For instance, if $\omega^b(\mathsf{z}) = w + v$ and $\omega^N(\mathsf{z}) = 2^w - 1$, then $\textsc{Conv}(z, \omega)$ returns $(\mathsf{pow2}(w) - 1) \bmod \mathsf{pow2}(w + v)$.

Equalities are processed by Conv by recursively running the translation on both sides. The next case handles applications of symbols from the signature Σ_{BV}, where symbols f^{BV} are replaced with the corresponding uninterpreted function $f^{\mathbb{N}}$. The first argument of $f^{\mathbb{N}}$ is $\omega^b(t_n)$, indicating the symbolic bit-width of the last argument of f^{BV}. The remaining arguments are obtained by recursively calling Conv on $t_1, \ldots, t_n$. In all cases, $\omega^b(t_n)$ corresponds to the bit-width that the uninterpreted function $f^{\mathbb{N}}$ expects, based on its intended semantics

[1] Our implementation of the translation does consider the general case since quantified formulas appear in the first of the case studies we discuss in Sect. 5.

(the bit-width of the second argument for bit-vector concatenation, or of an arbitrary argument for all other function and predicate symbols). Finally, CONV applies homomorphically to all terms whose top symbol is a Boolean connective.

CONV runs the auxiliary conversion function ELIM on all applications of uninterpreted functions $f^{\mathbb{N}}$ introduced during the conversion. ELIM eliminates a majority of functions corresponding to bit-vector operators as these functions are equivalently expressed using integer arithmetic and $\mathsf{pow2}$. Specifically, the ternary addition operation $+^{\mathbb{N}}$, which represents addition of two bit-vectors with their width k specified as the first argument, is translated to integer addition modulo $\mathsf{pow2}(k)$. The translation of subtraction $-^{\mathbb{N}}$ and multiplication $\cdot^{\mathbb{N}}$ applications is similar. For the division and remainder operations $\mathrm{div}^{\mathbb{N}}$ and $\mathrm{mod}^{\mathbb{N}}$, CONV handles the special case where the second argument is zero, consistently with the semantics of bit-vectors operators in the SMT-LIB 2 standard. The integer operators corresponding to unary (arithmetic) negation and bitwise negation can be eliminated in a straightforward way. The semantics of various bitwise shift operators can be defined arithmetically using division and multiplication with $\mathsf{pow2}(k)$. Arithmetic shift right ${>>_a}^{\mathbb{N}}$ can be defined in terms of other bit-vector operators and hence can be eliminated based on this definition. Concatenation can be eliminated by multiplying its first argument x by $\mathsf{pow2}(k)$ (recall that k is the bit-width of the second argument y) which has the effect of shifting x left by k bits, as expected. The unsigned relation symbols can be directly converted to the corresponding integer relation. For the elimination of signed relation symbols we use an auxiliary helper uts (unsigned to signed), also defined in Fig. 1, which returns the interpretation of its argument when seen as a signed value. The definition of uts can be derived based on the semantics of signed and unsigned bit-vector values in the SMT-LIB 2 standard. Based on this definition, we have that integers v and u that encode bit-vectors of bit-width k, satisfy ${<_s}^{\mathbb{N}}(k, u, v)$ if and only if they satisfy $\mathsf{uts}_k(u) < \mathsf{uts}_k(v)$.

Example 2 As an example of our translation, let $\varphi = (\mathsf{x} +^{\mathrm{BV}} \mathsf{x}) +^{\mathrm{BV}} \mathsf{z}_1 \not\approx \mathsf{z}_0$, $\omega^N(\mathsf{z}_0) = 0$, $\omega^N(\mathsf{z}_1) = 1$, and $\omega^b(\mathsf{x}) = \omega^b(\mathsf{z}_0) = \omega^b(\mathsf{z}_1) = a$ from Example 1. Then, the result of $\mathrm{CONV}(\varphi, (\omega^b, \omega^N))$ is

$$\mathrm{ELIM}(+^{\mathbb{N}}(a, \mathrm{ELIM}(+^{\mathbb{N}}(a, \chi(\mathsf{x}), \chi(\mathsf{x}))), 1 \bmod \mathsf{pow2}(a))) \not\approx 0 \bmod \mathsf{pow2}(a) \ .$$

After applying ELIM and simplifying, we get $(\chi(\mathsf{x}) + \chi(\mathsf{x}) + 1) \bmod \mathsf{pow2}(a) \not\approx 0$.

Thanks to the application of ELIM, we have that all formulas generated by CONV contain only uninterpreted function symbols in the set $\mathcal{F} = \{\mathsf{pow2}, \&^{\mathbb{N}}, |^{\mathbb{N}}, \oplus^{\mathbb{N}}\}$. Thus, we restrict our attention to these symbols only in our axiomatization AX_A, described next. From now on we denote $T^{\mathcal{F}}_{\mathrm{UFIA}}$ by T_{UFIA}.

4.1.3 Axiomatization Modes

We consider four different axiomatization modes A, which we call *full*, *partial*, *comb*, and *qf* (quantifier-free). These induce four different translations, namely: $\mathcal{T}_{full}$, $\mathcal{T}_{\partial}$, $\mathcal{T}_{comb}$, and $\mathcal{T}_{qf}$. For each of these modes, we define $\mathrm{AX}_A(\varphi, \omega)$ as the conjunction:11

$$\bigwedge_{x \in \mathrm{FV}(\varphi)} 0 \leq \chi(x) < \mathsf{pow2}(\omega^b(x)) \wedge \left(\bigwedge_{w \in \mathrm{FV}(\omega)} w > 0 \right) \wedge \mathrm{AX}^{\mathsf{pow2}}_A \wedge \mathrm{AX}^{\&^{\mathbb{N}}}_A \wedge \mathrm{AX}^{|^{\mathbb{N}}}_A \wedge \mathrm{AX}^{\oplus^{\mathbb{N}}}_A$$

The first conjunction states that all integer variables introduced for parametric bit-vector variables x reside in the range specified by their bit-width. The second conjunction states

Table 2 Full axiomatization of pow2, $\&^{\mathbb{N}}$, $|^{\mathbb{N}}$, and $\oplus^{\mathbb{N}}$

$\diamond$	$\mathrm{AX}^{\diamond}_{full}$
pow2	$\mathsf{pow2}(0) \approx 1 \wedge \forall k.\, k > 0 \Rightarrow \mathsf{pow2}(k) \approx 2 \cdot \mathsf{pow2}(k-1)$
$\&^{\mathbb{N}}$	$\forall k, x, y.\, \&^{\mathbb{N}}(k, x, y) \approx$ $\mathsf{ite}(k > 1, \&^{\mathbb{N}}(k-1, x \bmod \mathsf{pow2}(k-1), y \bmod \mathsf{pow2}(k-1)), 0) +$ $\mathsf{pow2}(k-1) \cdot \min(\mathsf{ex}_{k-1}(x), \mathsf{ex}_{k-1}(y))$
$\|^{\mathbb{N}}$	$\forall k, x, y.\, \|^{\mathbb{N}}(k, x, y) \approx$ $ite(k > 1, \|^{\mathbb{N}}(k-1, x \bmod \mathsf{pow2}(k-1), y \bmod \mathsf{pow2}(k-1)), 0)$ $+\mathsf{pow2}(k-1) \cdot \max(\mathsf{ex}_{k-1}(x), \mathsf{ex}_{k-1}(y))$
$\oplus^{\mathbb{N}}$	$\forall k, x, y.\, \oplus^{\mathbb{N}}(k, x, y) \approx$ $\mathsf{ite}(k > 1, \oplus^{\mathbb{N}}(k-1, x \bmod \mathsf{pow2}(k-1), y \bmod \mathsf{pow2}(k-1)), 0) +$ $\mathsf{pow2}(k-1) \cdot \lvert \mathsf{ex}_{k-1}(x) - \mathsf{ex}_{k-1}(y) \rvert$

We use $\mathsf{ex}_i(x)$ for $(x \div \mathsf{pow2}(i)) \bmod 2$, $\min(x, y)$ for $\mathsf{ite}(x < y, x, y)$, and $\max(x, y)$ for $\mathsf{ite}(x < y, y, x)$

Table 3 Partial axiomatization of pow2, $\&^{\mathbb{N}}$, and $\oplus^{\mathbb{N}}$

$\diamond$	axiom	$\mathrm{AX}^{\diamond}_{partial}$
pow2	base cases	$\mathsf{pow2}(0) \approx 1 \wedge \mathsf{pow2}(1) \approx 2 \wedge \mathsf{pow2}(2) \approx 4 \wedge \mathsf{pow2}(3) \approx 8$
	weak monotonicity	$\forall i \forall j.\, i \leq j \Rightarrow \mathsf{pow2}(i) \leq \mathsf{pow2}(j)$
	strong monotonicity	$\forall i \forall j.\, i < j \Rightarrow \mathsf{pow2}(i) < \mathsf{pow2}(j)$
	modularity	$\forall i \forall j \forall x.\, (x \cdot \mathsf{pow2}(i)) \bmod \mathsf{pow2}(j) \not\approx 0 \Rightarrow i < j$
	never even	$\forall i \forall x.\, \mathsf{pow2}(i) - 1 \not\approx 2 \cdot x$
	always positive	$\forall i.\, \mathsf{pow2}(i) \geq 1$
	div 0	$\forall i.\, i \div \mathsf{pow2}(i) \approx 0$
$\&^{\mathbb{N}}$	base case	$\forall x \forall y.\, \&^{\mathbb{N}}(1, x, y) \approx \min(\mathsf{ex}_0(x), \mathsf{ex}_0(y))$
	max	$\forall k \forall x.\, \&^{\mathbb{N}}(k, x, \max^{\mathbb{N}}_k) \approx x$
	min	$\forall k \forall x.\, \&^{\mathbb{N}}(k, x, 0) \approx 0$
	idempotence	$\forall k \forall x.\, \&^{\mathbb{N}}(k, x, x) \approx x$
	contradiction	$\forall k \forall x.\, \&^{\mathbb{N}}(k, x, \sim^{\mathbb{N}}(k, x)) \approx 0$
	symmetry	$\forall k \forall x \forall y.\, \&^{\mathbb{N}}(k, x, y) \approx \&^{\mathbb{N}}(k, y, x)$
	difference	$\forall k \forall x \forall y \forall z.\, x \not\approx y \Rightarrow \&^{\mathbb{N}}(k, x, z) \not\approx y \vee \&^{\mathbb{N}}(k, y, z) \not\approx x$
	range	$\forall k \forall x \forall y.\, 0 \leq \&^{\mathbb{N}}(k, x, y) \leq \min(x, y)$
$\|^{\mathbb{N}}$	base case	$\forall x \forall y.\, \|^{\mathbb{N}}(1, x, y) \approx \max(\mathsf{ex}_0(x), \mathsf{ex}_0(y))$
	max	$\forall k \forall x.\, \|^{\mathbb{N}}(k, x, \max^{\mathbb{N}}_k) \approx \max^{\mathbb{N}}_k$
	min	$\forall k \forall x.\, \|^{\mathbb{N}}(k, x, 0) \approx x$
	idempotence	$\forall k \forall x.\, \|^{\mathbb{N}}(k, x, x) \approx x$
	excluded middle	$\forall k \forall x.\, \|^{\mathbb{N}}(k, x, \sim^{\mathbb{N}}(k, x)) \approx \max^{\mathbb{N}}_k$
	symmetry	$\forall k \forall x \forall y.\, \|^{\mathbb{N}}(k, x, y) \approx \|^{\mathbb{N}}(k, y, x)$
	difference	$\forall k \forall x \forall y \forall z.\, x \not\approx y \Rightarrow \|^{\mathbb{N}}(k, x, z) \not\approx y \vee \|^{\mathbb{N}}(k, y, z) \not\approx x$
	range	$\forall k \forall x \forall y.\, \max(x, y) \leq \|^{\mathbb{N}}(k, x, y) \leq \max^{\mathbb{N}}_k$
$\oplus^{\mathbb{N}}$	base case	$\forall x \forall y.\, \oplus^{\mathbb{N}}(1, x, y) \approx \mathsf{ite}(\mathsf{ex}_0(x) \approx \mathsf{ex}_0(y), 0, 1)$
	zero	$\forall k \forall x.\, \oplus^{\mathbb{N}}(k, x, x) \approx 0$

Table 3 continued

$\diamond$	axiom	$\mathrm{AX}^{\diamond}_{partial}$
	one	$\forall k \forall x.\, \oplus^{\mathbb{N}}(k, x, \sim^{\mathbb{N}}(k, x)) \approx \max^{\mathbb{N}}_{k}$
	symmetry	$\forall k \forall x \forall y.\, \oplus^{\mathbb{N}}(k, x, y) \approx \oplus^{\mathbb{N}}(k, y, x)$
	range	$\forall k \forall x \forall y.\, 0 \leq \oplus^{\mathbb{N}}(k, x, y) \leq \max^{\mathbb{N}}_{k}$

We use $\max^{\mathbb{N}}_{k}$ for $\mathsf{pow2}(k) - 1$

that all free variables in ω (denoting bit-widths) are positive. The remaining four conjuncts denote the axiomatizations for the four uninterpreted functions that may occur in the output of the conversion function. The definitions of these formulas are given in Tables 2 and 3 for axiomatizations *full* and *partial* respectively. For each axiom, i, j, k denote bit-widths and x, y denote integers that encode bit-vectors of size k. We assume guards on all quantified formulas (omitted for brevity) that constrain i, j, k to be positive and x, y to be in the range $\{0, \ldots, \mathsf{pow2}(k) - 1\}$. Each table entry lists a set of formulas (interpreted conjunctively) that state properties about the intended semantics of these operators. The formulas for axiomatization mode *full* assert the intended semantics of these operators, whereas those for *partial* assert several properties of them. Axiomatization *comb* asserts both, and mode *qf* takes only the formulas in axiomatization *partial* that are quantifier-free. In particular, $\mathrm{AX}^{\mathsf{pow2}}_{qf}$ corresponds to the base cases listed in *partial*, and $\mathrm{AX}^{\diamond}_{qf}$ for the other operators is simply $\top$. The partial axiomatization of these operations mainly includes natural properties of them. For example, we include some base cases for each operation, and also the ranges of its inputs and output. For some proofs, these are sufficient. For operators $\&^{\mathbb{N}}$, $|^{\mathbb{N}}$ and $\oplus^{\mathbb{N}}$ we also include behavior for specific cases, e.g., $\&^{\mathbb{N}}(k, a, 0) = 0$ and its variants. Other axioms (e.g., "never even") were added after analyzing specific benchmarks to identify sufficient axioms for their proofs.

4.2 Correctness

Our translation satisfies the following key properties.

Theorem 1 *Let φ be a parametric bit-vector formula that is well-sorted under ω and has no occurrences of operator bit-vector extract or concrete bit-vector constants. Then, the following hold.*

1. *φ is T_{BV}-valid under ω if and only if $\mathcal{T}_{full}(\varphi, \omega)$ is T_{UFIA}-valid.*
2. *φ is T_{BV}-valid under ω if and only if $\mathcal{T}_{comb}(\varphi, \omega)$ is T_{UFIA}-valid.*
3. *φ is T_{BV}-valid under ω if $\mathcal{T}_{\partial}(\varphi, \omega)$ is T_{UFIA}-valid.*
4. *φ is T_{BV}-valid under ω if $\mathcal{T}_{qf}(\varphi, \omega)$ is T_{UFIA}-valid.*

We prove Theorem 1 in the remainder of this section. We start with a proof of the first property described in the theorem, and then use this property in order to prove the others.

4.2.1 Proof of Property 1

The essence of the proof is to elevate the translation function described in Fig. 1 and Table 2 from formulas to interpretations. We start with the following definition that introduces this translation.

Definition 1 Given an interpretation $\mathcal{J}$ of T_{IA} that interprets the variables in $\boldsymbol{v} = \text{FV}(\omega)$ as positive integers, and an interpretation $\mathcal{I}$ of T_{BV}, we define a corresponding interpretation $\mathcal{I}\mathbb{N}$ of T_{UFIA} as follows:

- $\diamond^{\mathcal{I}\mathbb{N}}$ is set to satisfy $\text{AX}_{\diamond}^{full}$ for any $\diamond \in \{\mathsf{pow2}, \&^{\mathbb{N}}, |^{\mathbb{N}}, \oplus^{\mathbb{N}}\}$
- $v^{\mathcal{I}\mathbb{N}} = v^{\mathcal{J}}$ for every $v \in \boldsymbol{v}$
- $\chi(x)^{\mathcal{I}\mathbb{N}} = \left[(x_{[c]})^{\mathcal{I}}\right]_{\mathbb{N}}$, where $c = \omega^b(x)^{\mathcal{J}}$ for any $x \in X^*$

We will show that this translation between interpretations preserves satisfiability. The following lemmas will be useful for this purpose.

Lemma 1 *Let $\mathcal{I}$ be an interpretation of T_{UFIA} that satisfies AX_{full}^{pow2}. Let a be a bit-vector constant of bit-width k and $n = [a]_{\mathbb{N}}$. Then:*

1. *Over natural numbers, $\mathsf{pow2}^{\mathcal{I}}$ is identical to $\lambda x.2^x$.*
   In addition, $\mathcal{I}$ satisfies the following formulas:
2. $\left[i \circ^{\text{BV}} a\right]_{\mathbb{N}} \approx \mathsf{pow2}(k) \cdot i + [a]_{\mathbb{N}}$ *for any* $i \in \{0, 1\}$.
3. $n \bmod \mathsf{pow2}(k-1) \approx \left[a[k-2:0]^{\text{BV}}\right]_{\mathbb{N}}$
4. $n \div 2 \approx \left[a[k-1:1]^{\text{BV}}\right]_{\mathbb{N}}$
5. $\mathsf{ex}_i(n)^{\mathcal{I}} = a[i]$, *with* $\mathsf{ex}_i(n)$ *as* $(n \div \mathsf{pow2}(i)) \bmod 2$ *for every* $0 \leq i \leq k-1$.

Proof The first property follows from the standard inductive definition of the exponentiation function. The next three properties follow from the definition of $[\cdot]_{\mathbb{N}}$. For the last property, we use induction on k. If $k = 1$ then we must have $i = 0$. In this case, $n \in \{0, 1\}$ and $\mathsf{ex}_i(n)^{\mathcal{I}} = n = [a[0]]_{\mathbb{N}}$. Suppose $k > 1$. If $i = 0$, then this is shown similarly to the base case. Otherwise, $a[i]$ corresponds to the bit in index $i-1$ of the bit-vector $a[k-1:1]^{\text{BV}}$. By the induction hypothesis and previous items of this lemma, the latter is equal to $\mathsf{ex}_{i-1}(\left[a[k-1:1]^{\text{BV}}\right]_{\mathbb{N}})^{\mathcal{I}} = \mathsf{ex}_{i-1}([a]_{\mathbb{N}} \div 2)^{\mathcal{I}} = \mathsf{ex}_i(n)^{\mathcal{I}}$. □

In the following lemma, we show that the full axiomatization indeed captures (over the natural numbers) the intended meaning of the bitwise operators.

Lemma 2 *Let a and b be bit-vector constants of bit-width k, operator $\diamond \in \{\&, |, \oplus\}$, and $\mathcal{I}$ an interpretation of T_{UFIA} that satisfies $\text{AX}_{full}^{pow2} \wedge \text{AX}_{full}^{\diamond}$. Then $\mathcal{I}$ satisfies $\left[a \diamond^{\text{BV}} b\right]_{\mathbb{N}} \approx \diamond^{\mathbb{N}}(k, [a]_{\mathbb{N}}, [b]_{\mathbb{N}})$.*

Proof We prove the lemma for the case where $\diamond$ is & by induction on k. The other cases are shown similarly. If $k = 1$, then by Lemma 1, we have $(\left[a \&^{\text{BV}} b\right]_{\mathbb{N}})^{\mathcal{I}} = [\min(a, b)]_{\mathbb{N}} = \min(a, b) = \&^{\mathbb{N}}(1, [a]_{\mathbb{N}}, [b]_{\mathbb{N}})$. Now, suppose $k > 1$. Then,

$$\begin{aligned}
&\&^{\mathbb{N}}(k, [a]_{\mathbb{N}}, [b]_{\mathbb{N}})^{\mathcal{I}} \\
&\quad = (\&^{\mathbb{N}}(k-1, [a]_{\mathbb{N}} \bmod \mathsf{pow2}(k-1), [b]_{\mathbb{N}} \bmod \mathsf{pow2}(k-1)) \\
&\qquad + \mathsf{pow2}(k-1) \cdot \min(\mathsf{ex}_{k-1}(a), \mathsf{ex}_{k-1}(b)))^{\mathcal{I}}
\end{aligned}$$

By Lemma 1, we have that

$$\min(\mathsf{ex}_{k-1}(a), \mathsf{ex}_{k-1}(b)) = \left[a[k-1] \&^{\text{BV}} b[k-1]\right]_{\mathbb{N}}$$

By the induction hypothesis and Lemma 1, we obtain that $\&^{\mathbb{N}}(k, [a]_{\mathbb{N}}, [b]_{\mathbb{N}})$ is equal in $\mathcal{I}$ to

$$\left[a[k-2:0]^{\text{BV}} \&^{\text{BV}} b[k-2:0]^{\text{BV}}\right]_{\mathbb{N}} + \mathsf{pow2}(k-1) \cdot \left[a[k-1] \&^{\text{BV}} b[k-1]\right]_{\mathbb{N}}$$

which by Lemma 1 is equal in $\mathcal{I}$ to

$$\left[(a\,[k-1]\&^{\mathrm{BV}} b\,[k-1]) \circ^{\mathrm{BV}} (a[k-2:0]^{\mathrm{BV}}\&^{\mathrm{BV}} a[k-2:0]^{\mathrm{BV}})\right]_{\mathbb{N}} = \left[a\&^{\mathrm{BV}} b\right]_{\mathbb{N}}$$

□

Now, we present the main lemma of the proof. It establishes a tight correspondence between the translation of Fig. 1 and Table 2 for formulas, and that of Definition 1 for interpretations.

Lemma 3 $\textsc{Conv}(t,\omega)^{\mathcal{I}\mathbb{N}} = \left[(t|_{\omega[\mathcal{J}]})^{\mathcal{I}}\right]_{\mathbb{N}}$ *for any parametric* Σ_{BV}*-term* t*, interpretation* $\mathcal{I}$ *of* T_{BV}*, and* $\mathcal{J}$ *as in Definition 1.*

Proof First, notice that for any $x \in X^* \cup Z^*$ we have that $\omega^b(x)^{\mathcal{I}\mathbb{N}} = \omega^b(x)^{\mathcal{J}}$ and $\omega^N(x)^{\mathcal{I}\mathbb{N}} = \omega^N(x)^{\mathcal{J}}$. Using induction, the same holds for any parametric Σ_{BV}-term t. We prove the lemma by induction on t.

- If t is x for some $x \in X^*$: follows from the definition of $\mathcal{I}\mathbb{N}$ for this case.
- If t is z for some $z \in Z^*$:

$$\begin{aligned}&\textsc{Conv}(t,\omega)^{\mathcal{I}\mathbb{N}}\\ &= (\omega^N(z) \bmod \mathsf{pow2}(\omega^b(z)))^{\mathcal{I}\mathbb{N}}\\ &= \omega^N(z)^{\mathcal{J}} \bmod \mathsf{pow2}(\omega^b(z)^{\mathcal{J}})\end{aligned}$$

  In turn, $z|_{\omega[\mathcal{J}]}$ is the bit-vector constant of width $k = \omega^b(z)^{\mathcal{J}}$ whose integer value is $\omega^N(z)^{\mathcal{J}} \bmod 2^k$.
- If t is constructed from an operator other than $\&^{\mathrm{BV}}$, $|^{\mathrm{BV}}$, $\oplus^{\mathrm{BV}}$, then this follows by the semantics of the various operators as defined in the SMT-LIB 2 standard. We explicitly show the case where $t = t_1 +^{\mathrm{BV}} t_2$. In this case,

$$\textsc{Conv}(t,\omega) = (\textsc{Conv}(t_1,\omega) + \textsc{Conv}(t_2,\omega)) \bmod \mathsf{pow2}(\omega^b(t_2)),$$

  which by Lemma 1, is interpreted in $\mathcal{I}\mathbb{N}$ as

$$((\textsc{Conv}(t_1,\omega))^{\mathcal{I}\mathbb{N}} + (\textsc{Conv}(t_2,\omega))^{\mathcal{I}\mathbb{N}}) \bmod 2^k$$

  for $k = \omega^b(t_2)^{\mathcal{J}}$. By the induction hypothesis, the latter is equal to

$$(\left[t_1|_{\omega[\mathcal{J}]}{}^{\mathcal{I}}\right]_{\mathbb{N}} + \left[t_2|_{\omega[\mathcal{J}]}{}^{\mathcal{I}}\right]_{\mathbb{N}}) \bmod 2^k,$$

  which by the semantics of $+^{\mathrm{BV}}$ as defined in the SMT-LIB 2 standard, is equal to $\left[((t_1 +^{\mathrm{BV}} t_2)|_{\omega[\mathcal{J}]})^{\mathcal{I}}\right]_{\mathbb{N}}$.
- The operators $\&^{\mathrm{BV}}$, $|^{\mathrm{BV}}$ and $\oplus^{\mathrm{BV}}$ rely on Lemma 2, rather than on the SMT-LIB 2 standard. We explicitly show the case where $t = t_1 \&^{\mathrm{BV}} t_2$. In this case,

$$\textsc{Conv}(t,\omega)^{\mathcal{I}\mathbb{N}} = (\&^{\mathbb{N}}(\omega^b(t_2), \textsc{Conv}(t_1,\omega), \textsc{Conv}(t_2,\omega)))^{\mathcal{I}\mathbb{N}},$$

  which by the induction hypothesis, is equal to

$$(\&^{\mathbb{N}}(\omega^b(t_2), \left[t_1|_{\omega[\mathcal{J}]}{}^{\mathcal{I}}\right]_{\mathbb{N}}, \left[t_2|_{\omega[\mathcal{J}]}{}^{\mathcal{I}}\right]_{\mathbb{N}}))^{\mathcal{I}\mathbb{N}}$$

  By Lemma 2, we obtain $\left[((t_1 \&^{\mathrm{BV}} t_1)|_{\omega[\mathcal{J}]})^{\mathcal{I}}\right]_{\mathbb{N}}$.

□

Using Lemma 3, we prove the correctness of the translation between interpretations:

Lemma 4 *$\mathcal{I}$ satisfies $\varphi|_{\omega[\mathcal{J}]}$ iff $\mathcal{I}\mathbb{N}$ satisfies $\mathcal{T}_{full}(\varphi, \omega)$.*

Proof By induction on φ. We explicitly prove two base cases. The remaining cases are proved similarly. Further, the induction steps (for formulas composed using the logical connectives), follow directly from the induction hypothesis.

- If φ has the form $t_1 = t_2$, then $I \models \varphi|_{\omega[\mathcal{J}]}$ iff $t_1|^{\mathcal{I}}_{\omega[\mathcal{J}]} = t_2|^{\mathcal{I}}_{\omega[\mathcal{J}]}$ iff $\left[t_1|^{\mathcal{I}}_{\omega[\mathcal{J}]}\right]_{\mathbb{N}} = \left[t_2|^{\mathcal{I}}_{\omega[\mathcal{J}]}\right]_{\mathbb{N}}$ iff (Lemma 3) $\textsc{Conv}(t_1, \omega)^{\mathcal{I}\mathbb{N}} = \textsc{Conv}(t_2, \omega)^{\mathcal{I}\mathbb{N}}$ iff $\mathcal{I}\mathbb{N} \models \textsc{Conv}(t_1, \omega) = \textsc{Conv}(t_2, \omega)$ iff $\mathcal{I}\mathbb{N} \models \mathcal{T}_{full}(t_1 = t_2, \omega)$.
- If φ has the form $t_1 <_{\mathrm{s}}{}^{\mathrm{BV}} t_2$, then recall that the SMT-LIB 2 semantics of $x <_{\mathrm{s}}{}^{\mathrm{BV}} y$ are given by the formula $(x[k-1:k-1]^{\mathrm{BV}} = 1 \wedge y[k-1:k-1]^{\mathrm{BV}} = 0) \vee (x[k-1:k-1]^{\mathrm{BV}} = y[k-1:k-1]^{\mathrm{BV}} \wedge x <_{\mathrm{u}}{}^{\mathrm{BV}} y)$. It is easy to see that this condition is equivalent to $[x]_{\mathbb{Z}} < [y]_{\mathbb{Z}}$. Thus, in this case, we have that $I \models \varphi|_{\omega[\mathcal{J}]}$ iff

$$(*)\ \left[t_1|^{\mathcal{I}}_{\omega[\mathcal{J}]}\right]_{\mathbb{Z}} < \left[t_2|^{\mathcal{I}}_{\omega[\mathcal{J}]}\right]_{\mathbb{Z}}.$$

  Now, for any term t, $\mathsf{uts}_k(\left[(t|_{\omega[\mathcal{J}]})^{\mathcal{I}}\right]_{\mathbb{N}}) = \left[(t|_{\omega[\mathcal{J}]})^{\mathcal{I}}\right]_{\mathbb{Z}}$ for every parametric bit-vector term t with $k = \omega^b(t)$. Indeed, let $v = t|^{\mathcal{I}}_{\omega[\mathcal{J}]}$. Then, $[v]_{\mathbb{N}} = 2^{k-1} \cdot v[k-1] + \left[v[k-2:0]^{\mathrm{BV}}\right]_{\mathbb{N}}$ and $[v]_{\mathbb{Z}} = -2^{k-1} \cdot v[k-1] + \left[v[k-2:0]^{\mathrm{BV}}\right]_{\mathbb{N}}$. Adding these two equations and simplifying, we get $[v]_{\mathbb{Z}} = 2 \cdot \left[v[k-2:0]^{\mathrm{BV}}\right]_{\mathbb{N}} - [v]_{\mathbb{N}} = [v]_{\mathbb{N}} \bmod 2^{k-1} - [v]_{\mathbb{N}} = \mathsf{uts}_k([v]_{\mathbb{N}})$. Hence, condition $(*)$ above holds iff $\mathsf{uts}_k(\left[t_1|^{\mathcal{I}}_{\omega[\mathcal{J}]}\right]_{\mathbb{N}}) < \mathsf{uts}_k(\left[t_2|^{\mathcal{I}}_{\omega[\mathcal{J}]}\right]_{\mathbb{N}})$, with $k = \omega^b(t_1)^{\mathcal{J}}$ iff (by Lemma 3) $\mathsf{uts}_k(\textsc{Conv}(t_1, \omega)^{\mathcal{I}\mathbb{N}}) < \mathsf{uts}_k(\textsc{Conv}(t_2, \omega)^{\mathcal{I}\mathbb{N}})$ iff $\mathcal{I}\mathbb{N} \models \mathcal{T}_{full}(\varphi, \omega)$.

□

Finally, we can collect all the pieces and prove the correctness of the full translation between formulas.

Proof of Property 1 of Theorem 1

($\Leftarrow$): Suppose φ is not T_{BV}-valid under ω. Then there exists a T_{IA}-interpretation $\mathcal{J}$ that maps variables in $\boldsymbol{v}$ to positive integers such that $\varphi|_{\omega[\mathcal{J}]}$ is not T_{BV}-valid. Hence, there exists an interpretation $\mathcal{I}$ of T_{BV} such that $\mathcal{I} \not\models \varphi|_{\omega[\mathcal{J}]}$. By Lemma 4, the constructed interpretation $\mathcal{I}\mathbb{N}$ of T_{UFIA} does not satisfy $\mathcal{T}_{full}(\varphi, \omega)$. Hence, $\mathcal{T}_{full}(\varphi, \omega)$ is not T_{UFIA}-valid.

($\Rightarrow$): Suppose $\mathcal{T}_{full}(\varphi, \omega)$ is not T_{UFIA}-valid. Then there exists an interpretation $\mathcal{I}$ of T_{UFIA} such that $\mathcal{I} \not\models \mathcal{T}_{full}(\varphi, \omega)$. We define a IA-interpretation $\mathcal{J}$ such that $\varphi|_{\omega[\mathcal{J}]}$ is not T_{BV}-valid, by setting $x^{\mathcal{J}} = x^{\mathcal{I}}$ for each $x \in \mathrm{FV}(\omega)$. Translation $\mathcal{T}_{full}(\varphi, \omega)$ is an implication whose left size is a conjunction whose first two conjuncts are: $\bigwedge_{x \in \mathrm{FV}(\varphi)} 0 \leq \chi(x) < \mathsf{pow2}(\omega^b(x))$ (denoted A) and $\bigwedge_{w \in \mathrm{FV}(\omega)} w > 0$ (denoted B). Notice that $\mathcal{J}$ maps the variables in $\mathrm{FV}(\omega)$ to positive integers, as $\mathcal{I} \models B$. To show that $\varphi|_{\omega[\mathcal{J}]}$ is not T_{BV}-valid, we define a T_{BV}-interpretation $\mathcal{I}\mathrm{BV}$ by setting $x^{\mathcal{I}\mathrm{BV}}_{[c]} = \left[\chi(x)^{\mathcal{I}}\right]^{-1}_{\mathbb{N}_k}$, where $c = \omega^b(x)^{\mathcal{J}}$ and $[.]^{-1}_{\mathbb{N}_.}$ can be considered as an "inverse" of $[.]_{\mathbb{N}}$. Concretely, if $0 \leq y < 2^k$, then there is a unique bit-vector x with minimal width such that $[x]_{\mathbb{N}} = y$. If this minimal width is at most k, then $[y]^{-1}_{\mathbb{N}_k}$ is that bit-vector, padded with zeros to width k. Otherwise, $[y]^{-1}_{\mathbb{N}_k}$ is undefined. In the case of $\mathcal{I}$ above, every usage of $[.]^{-1}_{\mathbb{N}_.}$ is defined, thanks to the fact that $\mathcal{I} \models A$. It can be easily shown that $\mathcal{I}\mathrm{BV}\mathbb{N} = \mathcal{I}$. By Lemma 4, since $\mathcal{I} \not\models \mathcal{T}_{full}(\varphi, \omega)$, we have that $\mathcal{I}\mathrm{BV} \not\models \varphi|_{\omega[\mathcal{J}]}$.

□

4.2.2 Proof of Properties 2–4

The rest of the properties follow from Property 1 by showing that the axioms in Table 3 are valid in every interpretation of T_{UFIA} that satisfies $\mathrm{AX}_{full}(\varphi, \omega)$.

The axioms of pow2 easily follow from simple properties of exponentiation. As for the bitwise logical operators, we explicitly prove the validity of "difference" for $\&^{\mathbb{N}}$. The rest are shown similarly. Let $k > 0$, $0 \leq x, y, z < 2^k$, and let $\mathcal{I}$ be an interpretation of T_{UFIA} that satisfies $\mathrm{AX}_{full}(\varphi, \omega)$. Note that k, x, y and z are integer constants, and are therefore interpreted as themselves in $\mathcal{I}$. Let a, b and c be bit-vectors of width k such that $x = [a]_{\mathbb{N}}$, $y = [b]_{\mathbb{N}}$ and $z = [c]_{\mathbb{N}}$. Suppose $\mathcal{I}$ satisfies $x \not\approx y$. Then $a \not\approx b$, and so there exists some $0 \leq i \leq k-1$ such that $a[i] \not\approx b[i]$. First, suppose $a[i] = 0$. Then $b[i] = 1$ and $(a \&^{\mathrm{BV}} c)[i] = 0$, and hence, $a \&^{\mathrm{BV}} c \not\approx b$. This means that $\left[a \&^{\mathrm{BV}} c\right]_{\mathbb{N}} \not\approx [b]_{\mathbb{N}}$, and thus, by Lemma 2, we have that $\mathcal{I}$ satisfies $\&^{\mathbb{N}}(k, x, z) \not\approx y$. Otherwise, $a[i] = 1$ and $b[i] = 0$. If $a \&^{\mathrm{BV}} c \approx b$ then $c[i] = 0$, which means that $(b \&^{\mathrm{BV}} c)[i] = 0$ and so $b \&^{\mathrm{BV}} c \not\approx a$. This means that $\left[b \&^{\mathrm{BV}} c\right]_{\mathbb{N}} \not\approx [a]_{\mathbb{N}}$, and thus by Lemma 2, we have that $\mathcal{I}$ satisfies $\&^{\mathbb{N}}(k, y, z) \not\approx x$. □

5 Case Studies

We apply the techniques from Sect. 4 to three case studies:

1. the verification of invertibility conditions from Niemetz et al. [23];
2. the verification of compiler optimizations as generated by Alive [22]; and
3. the verification of rewrite rules that are used in SMT solvers.

For these case studies, we consider a set of verification conditions that originally use fixed-size bit-vectors, and exclude formulas involving multiple bit-widths. This restriction simplifies the implementation but is interesting enough as each case study includes many verification conditions of this sort. Generalizing the framework beyond this restriction is technical but straightforward, and is left for future work. In essence, such an extension amounts to including the well-sortedness conditions of the underlying mappings ω in the translated benchmarks (e.g., equalities between bit-width variables that are associated with parametric bit-vector variables that occur under the same operator). For the case studies that we consider here, such extra constraints are trivial.

For each formula ϕ, we first produce a parametric version φ by replacing each variable in ϕ by a fresh $\mathsf{x} \in X^*$ and each (concrete) bit-vector constant by a fresh $\mathsf{z} \in Z^*$. We define $\omega^b(\mathsf{x}) = \omega^b(\mathsf{z}) = k$ for a fresh integer variable k and let $\omega^N(\mathsf{z})$ be the integer value corresponding to the bit-vector constant it replaced. We then define $\omega = (\omega^b, \omega^N)$ and invoke our translation from Sect. 4 on the parametric bit-vector formula φ. If the resulting formula is valid, the original verification condition holds independently of the original bit-width. In each case study, we report on the success rates of determining the validity of these formulas for axiomatization modes *full*, *partial*, *comb*, and *qf*. Overall, axiomatization mode *comb* yields the best results.

All experiments described below require an SMT solver with support for the SMT-LIB 2 logic UFNIA. We used all three participants in the UFNIA division of the 2018 SMT competition: CVC4 [2] (GitHub master 6eb492f6), Z3 [12] (version 4.8.4), and Vampire [18] (GitHub master d0ea236). Z3 and CVC4 use various strategies and techniques for quantifier instantiation including E-matching [11], and enumerative [29] and conflict-based [32] instantiation. For non-linear integer arithmetic, CVC4 uses an approach based on incremental

linearization [8,9,31]. Vampire is a superposition-based theorem prover for first-order logic based on the AVATAR framework [35], which has been extended to support some theories, including integer arithmetic [28]. We performed all experiments on a cluster with Intel Xeon E5-2637 CPUs with 3.5 GHz and 32 GB of memory. We imposed a time limit of 300 seconds (wallclock) and a memory limit of 4GB for each solver/benchmark pair.[2] We consider a bit-width independent property to be proved if at least one solver proved it for at least one of the axiomatization modes. The solving time reported does not include the (negligible) translation time, as the translation for all benchmarks was performed before running the experiments on the translated benchmarks.[3]

5.1 Verifying Invertibility Conditions

Niemetz et al. [23] present a technique for solving quantified bit-vector formulas that utilizes *invertibility conditions* to generate symbolic instantiations. Intuitively, an invertibility condition ϕ_c for a literal $\ell[x]$ is the exact condition under which $\ell[x]$ has a solution for x, i.e., $\phi_c \Leftrightarrow \exists x.\ell[x]$ is T_{BV}-valid. For example, consider bit-vector literal $x \&^{\mathrm{BV}} s \approx t$ with $x \notin \mathrm{FV}(s) \cup \mathrm{FV}(t)$; then, the invertibility condition for x is $t \&^{\mathrm{BV}} s \approx t$. The authors define invertibility conditions for a representative set of literals having a single occurrence of x that involve the bit-vector operators listed in Table 1, excluding extraction, as the invertibility condition for the latter is trivially $\top$. A considerable number of these conditions were determined by leveraging Syntax-Guided Synthesis (SyGuS) techniques [1]. The authors further verified the correctness of all conditions for bit-widths 1 to 65. However, a bit-width-independent proof of correctness of these conditions was left to future work. In the following, we apply the techniques of Sect. 4 to tackle this problem. Note that for this case study, we exclude operators involving multiple bit-widths, namely bit-vector extraction and concatenation. For the former, all invertibility conditions are $\top$, and for the latter it is easy to produce a hand-written parametric proof of correctness of its invertibility conditions.

5.1.1 Proving Invertibility Conditions

Let $\ell[x]$ be a bit-vector literal of the form $\diamond x \bowtie t$ or $x \diamond s \bowtie t$ (dually, $s \diamond x \bowtie t$), with operators $\diamond$ and relations $\bowtie$ as defined in Table 1. To prove the correctness of an invertibility condition ϕ_c for x independent of the bit-width, we have to prove the validity of the formula:

$$\phi_c \Leftrightarrow \exists x.\ell[x], \tag{1}$$

where occurrences of the variables s and t are implicitly universally quantified. We then want to prove that Eq. (1) is T_{BV}-valid under ω. Considering the two directions of Eq. (1) separately, we get:

$$\exists x.\ell[x, s, t] \Rightarrow \phi_c[s, t] \tag{rtl}$$

and

$$\phi_c[s, t] \Rightarrow \exists x.\ell[x, s, t]. \tag{ltr}$$

[2] On preliminary experiments we observed that higher time limits did not increase the overall success rate of the case studies.

[3] All benchmarks, results, log files, and solver configurations are available at http://cvc4.cs.stanford.edu/papers/CADE2019-JAR/.

The validity of (rtl) is equivalent to the unsatisfiability of the quantifier-free formula:

$$\ell[x, s, t] \wedge \neg \phi_c[s, t]. \tag{rtl'}$$

Eliminating the quantifier in (ltr) is much trickier. It typically amounts to finding a symbolic value for x such that $\ell[x, s, t]$ holds provided that $\phi_c[s, t]$ holds. We refer to such a symbolic value as a *conditional inverse*.

5.1.2 Conditional Inverses

Given an invertibility condition ϕ_c for x in bit-vector literal $\ell[x]$, we say that a term α_c is a *conditional inverse* for x if $\phi_c \Rightarrow \ell[\alpha_c]$ is T_{BV}-valid. For example, in literal $(x \mid^{\mathrm{BV}} s) \leq_{\mathrm{u}}^{\mathrm{BV}} t$, the term s itself is a conditional inverse for x: given that there exists some x such that $(x \mid^{\mathrm{BV}} s) \leq_{\mathrm{u}}^{\mathrm{BV}} t$, we have that $(s \mid^{\mathrm{BV}} s) \leq_{\mathrm{u}}^{\mathrm{BV}} t$. When a conditional inverse α_c for x is found, we may replace (ltr) by:

$$\phi_c \Rightarrow \ell[\alpha_c] \tag{ltr'}$$

Clearly, (ltr') implies (ltr). However, the converse may not hold, i.e., there may be models that falsifiy (ltr') without falsifying (ltr). Notice that if the invertibility condition for x is $\top$, the conditional inverse is in fact unconditional. The problem of finding a conditional inverse for a bit-vector literal $x \diamond s \bowtie t$ (dually, $s \diamond x \bowtie t$) can be defined as a SyGuS problem by asking whether the (second-order) formula $\exists C \forall s \forall t . \phi_c \Rightarrow C(s, t) \diamond s \bowtie t$ is satisfiable, where C is a binary function symbol. If such a function C is found, then it is in fact a conditional inverse for x in $\ell[x]$. We synthesized conditional inverses for x in $\ell[x]$ for bit-width 4 with variants of the grammars used in [23] to synthesize invertibility conditions. For each grammar, we generated 160 SyGuS problems, one for each combination of bit-vector operator and relation from Table 1 except for extraction and concatenation, counting commutative cases only once. We used the SyGuS solver CVC4SY [30] to solve these problems, and out of 160, we were able to synthesize candidate conditional inverses for 143 invertibility conditions. For 12 out of these 143, we found that the synthesized terms were not conditional inverses for every bit-width by checking (ltr') for bit-widths up to 64.

Tables 4 and 5 list all verified conditional inverses that we found. Note that we omitted superscript BV from all bit-vector symbols for better readability. The synthesized conditional inverses are useful for bit-width independent verification of the correctness of the invertibility conditions. In addition, they are interesting in their own right: they provide more constructive information about bit-vector literals by showing a concrete term for a general solution, which is an improvement compared to [23], where a formula that states the existence of such a solution was synthesized, without necessarily showing how to construct such a solution. We plan to study the affect of these more specific solutions in the context of quantifier instantiation, and to compare it to the technique of [23].

5.1.3 Results

Table 6 summaries the results of verifying invertibility conditions. For each invertibility condition, it states whether it was fully proved, only one direction of it was proved, or none of the directions were proved. Table 7 provides detailed information on the results for the axiomatization modes *full*, *partial*, and *qf* discussed in Sect. 4.

Table 4 Conditional inverses for relations $\bowtie$ in $\{\approx, \not\approx, <_u, \leq_u, >_u, \geq_u\}$

Literal	$\approx$	$\not\approx$	$<_u$	$\leq_u$	$>_u$	$\geq_u$
$-x \bowtie t$	$-t$	$\sim t$	0	0	$\sim t$	$-t$
$\sim x \bowtie t$	$\sim t$	t	$-t$	$\sim t$	0	0
$x + s \bowtie t$	$t - s$	$\sim (s + t)$	$-s$	$-s$	$\sim s$	$\sim s$
$x \& s \bowtie t$	t	$\sim t$	0	t	s	s
$x >> s \bowtie t$		$\sim t$	0	0	~ 0	~ 0
$s >> x \bowtie t$		$t >> s - t$	$\sim (s \mid \max_s)$	$\sim (s \mid \max_s)$	$s \& \min_s$	$s \& \min_s$
$x >> s \bowtie t$	$t << s$	$\min_s << t$	s	s	$\sim s$	$\sim s$
$s >> x \bowtie t$		$-t$	s	s	0	0
$x \cdot s \bowtie t$		$\max_s << t$	0	0		
$x \mid s \bowtie t$	t	$\sim t$	s	s	$\sim s$	t
$x << s \bowtie t$	$t >> s$	$\max_s << t$	0	0	~ 0	~ 0
$s << x \bowtie t$		t	$\min_s$	$\min_s$		
$x \operatorname{div} s \bowtie t$	$s \cdot t$	$s >> t$	0	t	~ 0	
$s \operatorname{div} x \bowtie t$		$t \& \min_s$	~ 0	~ 0	0	0
$x \mod s \bowtie t$	t	$-\sim t$	s	s	$\sim -s$	t
$s \mod x \bowtie t$	$s - t$	t	s	s	0	0

Table 5 Conditional inverses for relations $\bowtie$ in $\{<_s, \leq_s, >_s, \geq_s\}$

Literal	$<_s$	$\leq_s$	$>_s$	$\geq_s$
$-x \bowtie t$	$\min_s$	$\min_s$	$\sim t$	$-t$
$\sim x \bowtie t$	$\max_s$	$\max_s$	$\min_s$	$\min_s$
$x + s \bowtie t$	$\min_s - s$	$t - s$	$\max_s - s$	$t - s$
$x \& s \bowtie t$	$\min_s$	t	$\max_s$	$\max_s$
$x >> s \bowtie t$	$\min_s$	$\min_s$	$\max_s$	$\max_s$
$s >> x \bowtie t$	$\sim (s \mid \max_s)$	$\sim (s \mid \max_s)$	$s \& \min_s$	$s \& \min_s$
$x >> s \bowtie t$	$\min_s << s$	t	$\max_s << s$	$t << s$
$s >> x \bowtie t$	$\sim (s \mid \max_s)$	$\sim (s \mid \max_s)$		
$x \cdot s \bowtie t$				
$x \mid s \bowtie t$	$\min_s$	$\min_s$	$\max_s$	t
$x << s \bowtie t$	$\min_s >> s$	$t >> s$	$\max_s >> s$	$\max_s >> s$
$s << x \bowtie t$				
$x \operatorname{div} s \bowtie t$	$\sim -t$	t		
$s \operatorname{div} x \bowtie t$				
$x \mod s \bowtie t$	$\sim (\max_s \mid -s)$	$t \& \min_s$	$-\sim t$	t
$s \mod x \bowtie t$	t	$s - t$	$(s \mid \min_s) - (\max_s \& t - \max_s)$	$(s \mid \min_s) - (t \& \max_s)$

Table 6 Invertibility conditions verification

$\ell[x]$	$\approx$	$\not\approx$	$<_{\mathrm{u}}^{\mathrm{BV}}$	$>_{\mathrm{u}}^{\mathrm{BV}}$	$\leq_{\mathrm{u}}^{\mathrm{BV}}$	$\geq_{\mathrm{u}}^{\mathrm{BV}}$	$<_{\mathrm{s}}^{\mathrm{BV}}$	$>_{\mathrm{s}}^{\mathrm{BV}}$	$\leq_{\mathrm{s}}^{\mathrm{BV}}$	$\geq_{\mathrm{s}}^{\mathrm{BV}}$
$-^{\mathrm{BV}} x \bowtie t$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\sim^{\mathrm{BV}} x \bowtie t$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$x \&^{\mathrm{BV}} s \bowtie t$	→	✓	✓	✓	✓	✓	→	→	×	→
$x \mid^{\mathrm{BV}} s \bowtie t$	→	✓	✓	✓	✓	✓	→	×	→	×
$x <<^{\mathrm{BV}} s \bowtie t$	→	←	✓	→	✓	→	→	×	←	×
$s <<^{\mathrm{BV}} x \bowtie t$	✓	✓	✓	✓	✓	✓	←	✓	←	✓
$x >>^{\mathrm{BV}} s \bowtie t$	✓	✓	✓	→	✓	✓	✓	→	✓	→
$s >>^{\mathrm{BV}} x \bowtie t$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$x >>_a^{\mathrm{BV}} s \bowtie t$	×	✓	✓	✓	✓	✓	→	✓	→	✓
$s >>_a^{\mathrm{BV}} x \bowtie t$	✓	✓	←	←	←	←	←	×	←	✓
$x +^{\mathrm{BV}} s \bowtie t$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$x -^{\mathrm{BV}} s \bowtie t$	×	←	✓	×	✓	×	×	×	←	×
$x \mathrm{div}^{\mathrm{BV}} s \bowtie t$	✓	✓	✓	✓	✓	←	✓	✓	✓	✓
$s \mathrm{div}^{\mathrm{BV}} x \bowtie t$	✓	←	✓	✓	✓	✓	✓	←	✓	←
$x \mathrm{mod}^{\mathrm{BV}} s \bowtie t$	✓	✓	✓	✓	✓	✓	×	✓	←	✓
$s \mathrm{mod}^{\mathrm{BV}} x \bowtie t$	→	✓	✓	✓	✓	✓	✓	←	✓	←

✓ means was fully proved, →means only left-to-right proved, ←means only right-to-left proved, ×means not proved

We use →and ←to indicate that only direction left-to-right (ltr or ltr') or right-to-left (rtl') were proved, and ✓ and ×to indicate that both or none of the directions were proved. Additionally, we use $\rightarrow_{\alpha_c}$ (resp. $\rightarrow_{\mathrm{no}\,\alpha_c}$) to indicate that for direction left-to-right, formula (ltr') (resp. (ltr)) was proved with (resp. without) plugging in a conditional inverse.

Overall, out of 160 invertibility conditions, we were able to fully prove 110, and for 19 (resp. 17) conditions we were able to prove only direction rtl' (resp. ltr'). For direction right-to-left, 129 formulas (rtl') overall were successfully proved to be unsatisfiable. Out of these 129, 32 formulas were actually trivial since the invertibility condition ϕ_c was $\top$. For direction left-to-right, overall, 127 formulas were proved successfully, and out of these, 102 (resp. 94) were proved using (resp. not using) a conditional inverse. Furthermore, 33 formulas could only be proved when using a conditional inverse. Thus, using conditional inverses was helpful for proving the correctness of invertibility conditions. Interestingly, the opposite was also observed: 11 invertibility conditions for which conditional inverses were found, were not proven using conditional inverses, but were proven without them. This could be explained by the fact that even though conditional inverses eliminate a quantifier (thus making the problem easier), they produce a stronger formula, as (ltr') implies (ltr) (thus making the problem harder).

Considering the different axiomatization modes, overall, with 104 fully proved and only 17 unproved instances, axiomatization *comb* performed best. Interestingly, even though axiomatization *qf* only includes some of the base cases of axiomatization *partial*, it still performs well. This may be due to the fact that in many cases, the correctness of the invertibility condition does not rely on any particular property of the operators involved. For example, the invertibility condition ϕ_c for literal $x \&^{\mathrm{BV}} s \approx t$ is $t \&^{\mathrm{BV}} s \approx t$. Proving the correctness of

Table 7 Invertibility condition verification using axiomatization modes *comb*, *full*, *partial*, and *qf*

Axiomatization	✓	←	→	×	$\rightarrow_{\alpha_c}$	$\rightarrow_{\text{no}\,\alpha_c}$
full	64	18	22	56	72	51
partial	76	14	26	44	78	81
qf	40	22	22	76	50	51
comb	104	21	18	17	99	79
Total (160)	110	19	17	14	102	94

Column $\rightarrow_{\alpha_c}$ ($\rightarrow_{\text{no}\,\alpha_c}$) counts left-to-right proved with (without) conditional inverse

ϕ_c amounts to coming up with the right substitution for x without relying on any particular axiomatization of $\&^{\mathbb{N}}$. In contrast, the invertibility condition ϕ_c for literal $x\&^{BV}s \not\approx t$ is $t \not\approx 0 \vee s \not\approx 0$. Proving the correctness of ϕ_c relies on axioms regarding $\&^{BV}$ and $\sim^{BV}$. Specifically, we have found that from axiomatization *partial*, it suffices to keep "min" and "idempotence" to prove ϕ_c in that case.

The project of verifying invertibility conditions for arbitrary bit-widths was recently extended by Ekici et al. [14], where a representative subset consisting of 11 invertibility conditions out of the 50 that were not proven by our approach were proven semi-automatically using the Coq proof assistant, building on a library for SMT-LIB 2 bit-vectors previously developed by Ekici et al. [13].

5.2 Verifying Alive Optimizations

Lopes et al. [22] introduces Alive, a tool for proving the correctness of compiler peephole optimizations. Alive has a high-level language for specifying optimizations. The tool takes as input a description of an optimization in this high-level language and then automatically verifies that applying the optimization to an arbitrary piece of source code produces optimized target code that is equivalent under a given precondition. It can also automatically translate verified optimizations into C++ code that can be linked into LLVM [21]. For each optimization, Alive generates four constraints that encode the following properties, assuming that the precondition of the optimization holds:

1. *Memory* Source and Target yield the same state of memory after execution.
2. *Definedness* The target is well-defined whenever the source is.
3. *Poison* The target produces so-called poison values (caused by LLVM's *nsw*, *nuw*, and *exact* attributes) only when the source does.
4. *Equivalence* Source and target yield the same result after execution.

From these verification tasks, Alive can generate benchmarks in SMT-LIB 2 format in the theory of fixed-size bit-vectors, with and without quantifiers. For each task, types are instantiated with all possible valid type assignments (for integer types up to a default bound of 64 bits). In the following, we apply our techniques from Sect. 4 to prove Alive verification tasks independently from the bit-width. For this, as in Lopes et al. [22], we consider the set of optimizations from the *instcombine* optimization pass of LLVM, provided as Alive translations (433 total).[4] Of these 433 optimizations, 113 are dependent on a specific bit-width; thus, we focus on the remaining 320. We further exclude optimizations that do not comply with the following criteria:

[4] At https://github.com/nunoplopes/alive/tree/master/tests/instcombine.

Table 8 Alive optimizations verification using axiomatizations *comb*, *full*, *partial* and *qf*

Family	Considered	**Proved**				
		full	*partial*	*qf*	*comb*	Total
AddSub (52)	16	7	7	7	9	9
MulDivRem (29)	5	1	2	1	3	3
AndOrXor (162)	124	57	55	53	60	60
Select (51)	26	15	11	11	16	16
Shifts (17)	9	0	0	0	0	0
LoadStoreAlloca (9)	0	0	0	0	0	0
Total (320)	180	80	75	72	88	88

– In each generated SMT-LIB 2 file, only a single bit-width is used.
– All SMT-LIB 2 files generated for a property (instantiated for all possible valid type assignments) must be identical modulo the bit-width (excluding, e.g., bit-width dependent constants other than 0, 1 and (un)signed min/max).

As a useful exception to the first criterion, we included instances where all terms of bit-width 1 can be interpreted as Booleans. Overall, we consider bit-width independent verification conditions 1–4 for 180 out of 320 optimizations. None of these include memory operations or poison values, and only some have definedness constraints (and those are simple). We thus only consider the equivalence verification conditions for these 180 optimizations.

5.2.1 Results

Table 8 summarizes the results of verifying the equivalence constraints for the selected 180 optimizations from the *instcombine* LLVM optimization pass. The first column lists all families with the number of bit-width independent optimizations per family (320 total). The second column indicates how many in each family were in the set of 180 considered optimizations, and the remaining columns show how many of those considered were proved with each axiomatization mode.

Overall, out of 180 equivalence verification conditions, we were able to prove 88. Our techniques were most successful for the AndOrXor family. This is not surprising, since many verification conditions of this family require only Boolean reasoning and basic properties of ordering relations that are already included in the theory T_{IA}. For example, given bit-vector term a and bit-vector constants C_1 and C_2, optimization AndOrXor:979 from *instcombine* essentially rewrites $(a <_{\mathrm{s}}^{\mathrm{BV}} C_1 \wedge a <_{\mathrm{s}}^{\mathrm{BV}} C_2)$ to $a <_{\mathrm{s}}^{\mathrm{BV}} C_1$, provided that precondition $C_1 <_{\mathrm{s}}^{\mathrm{BV}} C_2$ holds. To prove its correctness, it suffices to apply the transitivity of $<_{\mathrm{s}}^{\mathrm{BV}}$ with Boolean reasoning. The same holds when lifting this equivalence to the integers, deducing the transitivity of $<_s^{\mathbb{N}}$ from that of the builtin $<$ relation of T_{IA}.

None of the 9 instances from the Shifts family were proven. These instances are more complicated than others, since they combine bitwise and arithmetical operations and thus rely on the axiomatization of these operators. Solving the instances from the Shifts family is an interesting challenge for future work. Adding specialized axioms to axiomatization *partial* is one promising approach.

Interestingly, for this case study, the results from the different axiomatization modes are very similar. This can again be explained by the fact that many optimizations rely on properties

Table 9 Summary of solvers performance on all generated SMT-LIB 2 benchmarks

Solver	Invertibility (2696) (%)	Alive (720) (%)	Rewriting (8024) (%)
CVC4	50.3	42.6	64.2
Vampire	31.4	36.2	66.5
Z3	33.8	37.9	64.2
All solvers	23.5	32.5	63.8
Some solvers	66.3	58.0	66.8

of the integers that are already included in T_{IA}, without requiring any particular property of functions pow2, $\&^{\mathbb{N}}$, $|^{\mathbb{N}}$ and $\oplus^{\mathbb{N}}$ (as in the above example).

Note that we have also tried using our approach for proving the equivalence verification conditions for up to a bit-width of 64. This was done by adding an assertion of the form $0 < k \leq 64$ for the bit-width variable k. However, all optimizations that were proven correct this way were already proven correct for arbitrary bit-widths, which suggests that this restriction did not make the input problem easier for the solvers.

5.3 BV Rewriting

SMT solvers for the theory of fixed-size bit-vectors heavily rely on rewriting to reduce the size of the input formula prior to solving the problem. Since these rewrite rules are usually implemented independently of the bit-width, verifying that they hold for any bit-width is crucial for the soundness of the solver. For this case study, we used a feature of the SyGuS solver in CVC4 that allows us to enumerate equivalent bit-vector terms/formulas (rewrite candidates) for a certain bit-width up to a certain term depth (nesting level of operators) [26]. We generated 1575 pairs of equivalent bit-vector terms of depth three and 431 equivalent pairs of formulas of depth two for bit-width 4 and translated them to integer problems with axiomatization modes *full*, *partial*, *qf*, and *comb*, resulting in $6300 + 1724 = 8024$ benchmarks in total. Since rewrites that have been proved correct can be used to further axiomatize the integer translation, we collected all proven rewrites after each run, added them as axioms to the initial problems and reran the experiments. This was repeated until we reached a fixpoint, i.e., no further rewrites were proved. With this approach, we were able to prove 409 out of the 431 formula equivalences (94%) by reaching a fixpoint at the first iteration. For the equivalent terms, we initially proved 878 out of the 1575 equivalences, which increased to 935 (59%) after adding all axioms from the first run, reaching a fixpoint after two iterations.

5.4 Summary

Summarizing these three case studies, Table 9 shows the percentage of problems that were proved by the various solver in each case study. Numbers in parenthesis denote the total number of SMT-LIB 2 benchmarks used in each case (including all encodings). Rows 1–3 show the percentage of problems proved by each solver, row 4 shows the percentage of problems proved by all three solvers, and row 5 shows the percentage of problems that were proved by at least one solver, which corresponds to the success rate of the virtual best solver.

6 Conclusion and Further Research

We have studied several translations from bit-vector formulas with parametric bit-width to the theories of integer arithmetic and uninterpreted functions. The translations differ in the way that the operator $2^{(\_)}$ and bitwise logical operators are axiomatized, namely, fully (using their recursive definition) or partially (using some of their key properties). Our empirical results show that state-of-the-art SMT solvers are capable of solving the translated formulas for various benchmarks that originate from the verification of invertibility conditions, LLVM optimizations, and rewriting rules for fixed-size bit-vectors.

In future research, we plan to investigate an implementation of our translation in a proof assistant such as Coq, for which a bit-vector library was recently developed [13], and supports the formalization of bit-width independent properties. A verified translation to integers can open a path to automating proofs for such properties in Coq. This will require also supporting proofs in the SMT solver for non-linear arithmetic and quantifiers. We also plan to explore satisfiable benchmarks and to consider lazy variants of the proposed translations. In such a setting, a partitioning of the input problem into sub-problems can be beneficial, as for each part, only the axioms that relate to the operators that occur in it will be considered. Something similar can be done with the bounds on the variables that occur in each part. In such an approach, we can sometimes avoid the introduction of quantifiers (which currently are always introduced, by adding all the axioms, even if the original formula is quantifier-free). This has potential in improving performance, and can also help with Nelson-Oppen style combination with other theories. Finally, we are experimenting with solving fixed-width (i.e., non-parametric) bit-vector formulas using a similar translation to integers. The main difference, though, is that in such a case, quantification is not needed. For exponentiation, the exponent is always a numeral when the original formula has only fixed-width bit-vectors. For the bitwise operators, their corresponding axiom schemas can be instantiated lazily or eagerly. Since the bit-width is known, this can be done in a complete manner.

References

1. Alur, R., Bodík, R., Juniwal, G., Martin, M.M.K., Raghothaman, M., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20–23, 2013, pp. 1–8 (2013)
2. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: CAV, CAV'11, pp. 171–177. Springer (2011). http://dl.acm.org/citation.cfm?id=2032305.2032319. Accessed on 15 Apr 2020
3. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. In: Gupta, A., Kroening, D. (eds.) Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK) (2010)
4. Bjørner, N.S., Pichora, M.C.: Deciding fixed and non-fixed size bit-vectors. In: Steffen, B. (ed.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 376–392. Springer, Berlin (1998)
5. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending sledgehammer with SMT solvers. J. Autom. Reason. **51**(1), 109–128 (2013)
6. Bozzano, M., Bruttomesso, R., Cimatti, A., Franzén, A., Hanna, Z., Khasidashvili, Z., Palti, A., Sebastiani, R.: Encoding rtl constructs for mathsat: a preliminary report. Electron. Notes Theor. Comput. Sci. **144**(2), 3–14 (2006). Proceedings of the Third Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR 2005)
7. Brinkmann, R., Drechsler, R.: Rtl-datapath verification using integer linear programming. In: Proceedings of ASP-DAC/VLSI Design 2002. 7th Asia and South Pacific Design Automation Conference and 15h International Conference on VLSI Design, pp. 741–746 (2002)

8. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Experimenting on solving nonlinear integer arithmetic with incremental linearization. In: SAT, Lecture Notes in Computer Science, vol. 10929, pp. 383–398. Springer (2018)
9. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. ACM Trans. Comput. Logic **19**(3), 19:1–19:52 (2018)
10. Development team, T.C.: The coq proof assistant reference manual version 8.9 (2019). https://coq.inria.fr/distrib/current/refman/. Accessed on 15 Apr 2020
11. de Moura, L.M., Bjørner, N.: Efficient e-matching for SMT solvers. In: Automated Deduction—CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17–20, 2007, Proceedings, pp. 183–198 (2007)
12. De Moura, L., Bjørner, N.: Z3: an efficient smt solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08/ETAPS'08, pp. 337–340. Springer (2008). http://dl.acm.org/citation.cfm?id=1792734.1792766. Accessed on 15 Apr 2020
13. Ekici, B., Mebsout, A., Tinelli, C., Keller, C., Katz, G., Reynolds, A., Barrett, C.: Smtcoq: a plug-in for integrating smt solvers into coq. In: CAV, pp. 126–133. Springer (2017)
14. Ekici, B., Viswanathan, A., Zohar, Y., Barrett, C.W., Tinelli, C.: Verifying bit-vector invertibility conditions in coq (extended abstract). PxTP, EPTCS **301**, 18–26 (2019)
15. Enderton, H., Enderton, H.B.: A Mathematical Introduction to Logic. Elsevier, Amsterdam (2001)
16. Gupta, A., Fisher, A.L.: Parametric circuit representation using inductive boolean functions. In: Courcoubetis, C. (ed.) CAV, pp. 15–28. Springer, Berlin (1993)
17. Gupta, A., Fisher, A.L.: Representation and symbolic manipulation of linearly inductive boolean functions. In: CAV, ICCAD '93, pp. 192–199. IEEE Computer Society Press, Los Alamitos, CA, USA (1993). http://dl.acm.org.stanford.idm.oclc.org/citation.cfm?id=259794.259827. Accessed on 15 Apr 2020
18. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: CAV, Lecture Notes in Computer Science, vol. 8044, pp. 1–35. Springer (2013)
19. Kovásznai, G., Fröhlich, A., Biere, A.: Complexity of fixed-size bit-vector logics. Theory Comput. Syst. **59**(2), 323–376 (2016)
20. Kroening, D., Strichman, O.: Decision Procedures—An Algorithmic Point of View. Texts in Theoretical Computer Science. An EATCS Series, 2nd edn. Springer, Berlin (2016)
21. Lattner, C., Adve, V.S.: LLVM: A compilation framework for lifelong program analysis & transformation. In: 2nd IEEE/ACM International Symposium on Code Generation and Optimization (CGO 2004), 20–24 March 2004, San Jose, CA, USA, pp. 75–88. IEEE Computer Society (2004)
22. Lopes, N.P., Menendez, D., Nagarakatte, S., Regehr, J.: Provably correct peephole optimizations with alive. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '15, pp. 22–32. ACM, New York, NY, USA (2015)
23. Niemetz, A., Preiner, M., Reynolds, A., Barrett, C., Tinelli, C.: Solving quantified bit-vectors using invertibility conditions. In: Chockler, H., Weissenbacher, G. (eds.) CAV, Lecture Notes in Computer Science, vol. 10982, pp. 236–255. Springer (2018)
24. Niemetz, A., Preiner, M., Reynolds, A., Zohar, Y., Barrett, C.W., Tinelli, C.: Towards bit-width-independent proofs in SMT solvers. In: CADE, Lecture Notes in Computer Science, vol. 11716, pp. 366–384. Springer (2019)
25. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-order Logic, vol. 2283. Springer, Berlin (2002)
26. Nötzli, A., Reynolds, A., Barbosa, H., Niemetz, A., Preiner, M., Barrett, C.W., Tinelli, C.: Syntax-guided rewrite rule enumeration for SMT solvers. In: SAT, Lecture Notes in Computer Science, vol. 11628, pp. 279–297. Springer (2019)
27. Pichora, M.C.: Automated reasoning about hardware data types using bit-vectors of symbolic lengths. Ph.D. thesis, Toronto, Ont., Canada, Canada (2003). AAINQ84686
28. Reger, G., Suda, M., Voronkov, A.: Unification with abstraction and theory instantiation in saturation-based reasoning. In: Tools and Algorithms for the Construction and Analysis of Systems—24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14–20, 2018, Proceedings, Part I, pp. 3–22 (2018)
29. Reynolds, A., Barbosa, H., Fontaine, P.: Revisiting enumerative instantiation. In: Tools and Algorithms for the Construction and Analysis of Systems—24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14–20, 2018, Proceedings, Part II, pp. 112–131 (2018)

30. Reynolds, A., Barbosa, H., Nötzli, A., Barrett, C.W., Tinelli, C.: cvc4sy: Smart and fast term enumeration for syntax-guided synthesis. In: Dillig, I., Tasiran, S. (eds.) CAV, Lecture Notes in Computer Science, vol. 11562, pp. 74–83. Springer (2019)
31. Reynolds, A., Tinelli, C., Jovanovic, D., Barrett, C.: Designing theory solvers with extensions. In: Frontiers of Combining Systems—11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27–29, 2017, Proceedings, pp. 22–40 (2017)
32. Reynolds, A., Tinelli, C., de Moura, L.M.: Finding conflicting instances of quantified formulas in SMT. In: Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21–24, 2014, pp. 195–202 (2014)
33. Solidity Language Developers: Solidity (2018). https://solidity.readthedocs.io/en/v0.4.25/. Accessed on 15 Apr 2020
34. Tinelli, C., Zarba, C.G.: Combining decision procedures for sorted theories. In: Alferes, J.J., Leite, J. (eds.) Logics in Artificial Intelligence, pp. 641–653. Springer, Berlin (2004)
35. Voronkov, A.: AVATAR: the architecture for first-order theorem provers. In: CAV, Lecture Notes in Computer Science, vol. 8559, pp. 696–710. Springer (2014)
36. Zeng, Z., Kalla, P., Ciesielski, M.: Lpsat: a unified approach to rtl satisfiability. In: Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001, pp. 398–402 (2001)