More on the Complexity of Quantifier-Free Fixed-Size Bit-Vector Logics with Binary Encoding[⋆]

Andreas Fröhlich, Gergely Kovásznai, and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria

Abstract. Bit-precise reasoning is important for many practical applications of Satisfiability Modulo Theories (SMT). In recent years, efficient approaches for solving fixed-size bit-vector formulas have been developed. From the theoretical point of view, only few results on the complexity of fixed-size bit-vector logics have been published. Most of these results only hold if unary encoding on the bit-width of bit-vectors is used.
In previous work [1], we showed that binary encoding adds more expressiveness to bit-vector logics, e.g. it makes fixed-size bit-vector logic without uninterpreted functions nor quantification NExpTime-complete. In this paper, we look at the quantifier-free case again and propose two new results. While it is enough to consider logics with *bitwise operations*, *equality*, and *shift by constant* to derive NExpTime-completeness, we show that the logic becomes PSpace-complete if, instead of *shift by constant*, only *shift by 1* is permitted, and even NP-complete if *no shifts* are allowed at all.

1 Introduction

Bit-precise reasoning over bit-vector logics is important for many practical applications of Satisfiability Modulo Theories (SMT), particularly for hardware and software verification. Examples of state-of-the-art SMT solvers with support for bit-precise reasoning are Boolector, MathSAT, STP, Z3, and Yices.

Syntax and semantics of *fixed-size bit-vector logics* do not differ much in the literature [2,3,4,5,6]. Concrete formats for specifying bit-vector problems also exist, e.g. the SMT-LIB format [7] or the BTOR format [8].

Working with *non-fixed-size* bit-vectors has been considered for instance in [4,9], and more recently in [10], but is not the focus of this paper. Most industrial applications (and examples in the SMT-LIB) have fixed bit-width.

We investigate the *complexity* of solving *fixed-size bit-vector formulas*. Some papers propose such complexity results, e.g. in [3] the authors consider quantifier-free bit-vector logic and give an argument for the NP-*hardness* of its satisfiability problem. In [5], a sublogic of the previous one is claimed to be NP-*complete*. Interestingly, in [11] there is a claim about the full quantifier-free bit-vector

[⋆] Supported by FWF, NFN Grant S11408-N23 (RiSE).

logic without uninterpreted functions (QF_BV) being NP-complete, however, the proposed decision procedure confirms this claim only if the bit-widths of the bit-vectors in the input formula are written/encoded in *unary* form. In [12,13], the *quantified* case is addressed, and the satisfiability problem of this logic with uninterpreted functions (UFBV) is proved to be NExpTime-*complete*. Again, the proof only holds if we assume unary encoded bit-widths. In practice, a more natural and exponentially more succinct *logarithmic* encoding is used, such as in the SMT-LIB, the BTOR, and the Z3 format.

In previous work [1], we already investigated how complexity varies if we consider either a unary or a logarithmic, actually without loss of generality, *binary encoding*. Apart from this, we are not aware of any work that investigates how the particular encoding of the bit-widths in the input affects complexity (as an exception, see [14, Page 239, Footnote 3]). Tab. 1 summarizes the completeness results we obtained in [1].

		quantifiers			
		no		*yes*	
		uninterpreted functions		uninterpreted functions	
		no	*yes*	*no*	*yes*
encoding	*unary*	NP	NP	PSpace	NExpTime
	binary	NExpTime	NExpTime	?	2-NExpTime

Table 1. Completeness results of [1] for various bit-vector logics and encodings.

In this paper, we revisit QF_BV2, the quantifier-free case with binary encoding and without uninterpreted functions. We then put certain restrictions on the operations we use (in particular on the *shift* operation). As a result, we obtain two new sublogics which we show to be PSpace-complete resp. NP-complete.

2 Motivation

In practice, state-of-the-art bit-vector solvers rely on rewriting and bit-blasting. The latter is defined as the process of translating a bit-vector resp. word-level description into a bit-level circuit, as in hardware synthesis. The result can then be checked by a (propositional) SAT solver. In [1], we gave the following example (in SMT2 syntax) to point out that bit-blasting is not polynomial in general. It checks commutativity of adding two bit-vectors of bit-width 1000000:

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 1000000))
(declare-fun y () (_ BitVec 1000000))
(assert (distinct (bvadd x y) (bvadd y x)))
```

Bit-blasting such formulas generates huge circuits, which shows that checking bit-vector logics through bit-blasting cannot be considered to be a polynomial reduction. This also disqualifies bit-blasting as a sound way to argue that the decision problem for (quantifier-free) bit-vector logics is in NP. We actually proved in [1], that deciding bit-vector logics, even without quantifiers, is much harder. It turned out to be NExpTime-complete in the general case.

However, in [1] we then also defined a class of *bit-width bounded problems* and showed that under certain restrictions on the bit-widths this growth in complexity can be avoided and the problem remains in NP.

In this paper, we give a more detailed classification of quantifier-free fixed-size bit-vector logics by investigating how complexity varies when we restrict the operations that can be used in a bit-vector formula. We establish two new complexity results for restricted bit-vector logics and bring together our previous results in [1] with work on linear arithmetic on non-fixed-size bit-vectors [10,15] and work on the reduction of bit-widths [16,17]. The formula in the given example only contains bitwise operations, equality, and addition. Solving this kind of formulas turns out to be PSpace-complete.

3 Definitions

We assume the usual syntax for (quantifier-free) bit-vector logics, with a restricted set of bit-vector operations: bitwise operations, equality, and (left) shift by constant.

Definition 1 (Term). *A bit-vector term t of bit-width n ($n \in \mathbb{N}$, $n \geqslant 1$) is denoted by $t^{[n]}$. A term is defined inductively as follows:*

	term	condition	bit-width
bit-vector constant:	$c^{[n]}$	$c \in \mathbb{N}$, $0 \leqslant c < 2^n$	n
bit-vector variable:	$x^{[n]}$	*x is an identifier*	n
bitwise negation:	$\sim t^{[n]}$	*$t^{[n]}$ is a term*	n
bitwise and/or/xor: $\bullet \in \{\&, \mid, \oplus\}$	$\left(t_1{}^{[n]} \bullet t_2{}^{[n]}\right)$	*$t_1{}^{[n]}$ and $t_2{}^{[n]}$ are terms*	n
equality:	$\left(t_1{}^{[n]} = t_2{}^{[n]}\right)$	*$t_1{}^{[n]}$ and $t_2{}^{[n]}$ are terms*	1
shift by constant:	$\left(t^{[n]} \ll c^{[n]}\right)$	*$t^{[n]}$ is a term, $c^{[n]}$ is a constant*	n

We also define how to measure the size of bit-vector expressions:

Definition 2 (Size). *The size of a bit-vector term $t^{[n]}$ is denoted by $|t^{[n]}|$ and is defined inductively as follows:*

	term	size
natural number:	$enc(n)$	$\lceil \log_2 (n+1) \rceil + 1$

bit-vector constant:	$\lvert c^{[n]} \rvert$	$enc(c) + enc(n)$
bit-vector variable:	$\lvert x^{[n]} \rvert$	$1 + enc(n)$
bitwise negation:	$\lvert \sim t^{[n]} \rvert$	$1 + \lvert t^{[n]} \rvert$
binary operations: $\bullet \in \{\&, \mid, \oplus, =, \ll\}$	$\lvert \left(t_1{}^{[n]} \bullet t_2{}^{[n]}\right) \rvert$	$1 + \lvert t_1{}^{[n]} \rvert + \lvert t_2{}^{[n]} \rvert$

A bit-vector term $t^{[1]}$ is also called a *bit-vector formula.* We say that a bit-vector formula is in *flat form* if it does not contain nested equalities. It is easy to see that any bit-vector formula can be translated to this form with only linear growth in the number of variables. In the rest of the paper, we may omit parentheses in a formula for the sake of readability.

Let Φ be a bit-vector formula and α an assignment to the variables in Φ. We use the notation $\alpha(\Phi)$ to denote the evaluation of Φ under α, with $\alpha(\Phi) \in \{0, 1\}$. α satisfies Φ if and only if $\alpha(\Phi) = 1$. We define three different bit-vector logics:

- QF_BV2$_{\ll c}$: bitwise operations, equality, and shift by any constant are allowed
- QF_BV2$_{\ll 1}$: bitwise operations, equality, and shift by only $c = 1$ are allowed
- QF_BV2$_{bw}$: only bitwise operations and equality are allowed

Obviously, QF_BV2$_{bw}$ $\subseteq$ QF_BV2$_{\ll 1}$ $\subseteq$ QF_BV2$_{\ll c}$. In Sec. 4, we investigate the complexity of the satisfiability problem for these logics:

- QF_BV2$_{\ll c}$ is NExpTime-complete.
- QF_BV2$_{\ll 1}$ is PSpace-complete.
- QF_BV2$_{bw}$ is NP-complete.

Adding uninterpreted functions does not change expressiveness of these logics, since in the quantifier-free case, uninterpreted functions can always be replaced by new variables. To guarantee functional consistency, Ackermann constraints have to be added to the formula. However, even in the worst case, the number of Ackermann constraints is only quadratic in the number of function instances. Without loss of generality, we therefore do not explicitly deal with uninterpreted functions.

4 Complexity Results

Theorem 1. QF_BV2$_{\ll c}$ *is* NExpTime-*complete.*

Proof. The claim directly follows from our previous work in [1]. We informally defined QF_BV2 as the quantifier-free bit-vector logic that uses the common bit-vector operations as defined for example in SMT-LIB, including bitwise operations, equality, shifts, addition, multiplication, concatenation, slicing, etc., and then showed that QF_BV2 is NExpTime-complete.

Obviously, QF_BV2$_{\ll c}$ $\subseteq$ QF_BV2 and therefore, QF_BV2$_{\ll c}$ $\in$ NExpTime. To show the NExpTime-hardness of QF_BV2, we gave a (polynomial) reduction from DQBF (which is NExpTime-complete [18]) to QF_BV2. Since we only used *bitwise operations*, *equality*, and *shift*[1] *by constant* in our reduction, we also immediately get the NExpTime-hardness of QF_BV2$_{\ll c}$. $\square$

Theorem 2. QF_BV2$_{\ll 1}$ *is* PSpace-*complete.*

Proof. In Lemma 1, we give a (polynomial) reduction from QBF (which is PSpace-complete) to QF_BV2$_{\ll 1}$. This shows the PSpace-hardness of QF_BV2$_{\ll 1}$. In Lemma 2, we then prove that QF_BV2$_{\ll 1}$ $\in$ PSpace by giving a translation from QF_BV2$_{\ll 1}$ to (polynomial sized) Sequential Circuits. As pointed out for example in [19], the symbolic reachability problem is PSpace-complete as well. $\square$

Lemma 1. QBF *can be (polynomially) reduced to* QF_BV2$_{\ll 1}$.

Proof. To show the PSpace-hardness of QF_BV2$_{\ll 1}$, we give a polynomial reduction from QBF similar to the one from DQBF to QF_BV2 that we proposed in [1]. For our reduction, we again use the so-called *binary magic numbers* (or magic masks in [20, p. 141]). Appendix B demonstrates how the reduction works.

Given $m, n \in \mathbb{N}$ with $0 \leqslant m < n$, a binary magic number can be written in the following form:

$$\mathit{binmagic}\,(2^m,\ 2^n) = \overbrace{\underbrace{0\ldots0}_{2^m}\,\underbrace{1\ldots1}_{2^m}\cdots\underbrace{0\ldots0}_{2^m}\,\underbrace{1\ldots1}_{2^m}}^{2^n}$$

Note that in [1], we used *shift by constant* to construct the binary magic numbers, as done in the literature [20]. This is not permitted in QF_BV2$_{\ll 1}$. We therefore give an alternative construction using only *bitwise operations*, *equality*, and *shift by 1*:

Given $n > 0$, for all m, $0 \leqslant m < n$, add the following equation to the formula:

$$b_m'^{[2^n]} = \left(\bigwedge_{0 \leqslant i < m} b_i^{[2^n]}\right) \oplus b_m^{[2^n]}$$

Consider all the bit-vector variables $b_0^{[2^n]}, \ldots, b_{n-1}^{[2^n]}$ as column vectors in a matrix $B^{[2^n \times n]}$ and all the bit-vector variables $b_0'^{[2^n]}, \ldots, b_{n-1}'^{[2^n]}$ as column vectors in a matrix $B'^{[2^n \times n]}$. If each row of B is interpreted as a number $0 \leqslant c < 2^n$ in binary representation, the corresponding row of B' is equal to $c + 1$.

Now, again for all m, $0 \leqslant m < n$, add another constraint:

$$b_m'^{[2^n]} = b_m^{[2^n]} \ll 1^{[2^n]}$$

[1] Note, logical right shifts were used in the proof in [1]. However, by applying negated bit masks throughout the proof, all right shifts can be rewritten as left shifts.

Together with the previous n equations, those n constraints force the rows of B to represent an enumeration of all binary numbers $0 \leqslant c < 2^n$. Therefore, the columns of B, i.e. the individual bit-vectors ${b_0}^{[2^n]}, \ldots, {b_{n-1}}^{[2^n]}$, exactly define the binary magic numbers: $\mathit{binmagic}\,(2^m,\ 2^n) := {b_m}^{[2^n]}$.

Of course, all b'_m, for $0 \leqslant m < n$, can be eliminated and the two sets of constraints can be replaced by a single set of constraints:

$$\left(\bigwedge_{0 \leqslant i < m} {b_i}^{[2^n]}\right) \oplus {b_m}^{[2^n]} = {b_m}^{[2^n]} \ll 1^{[2^n]}$$

Now let $\phi := Q.M$ denote a QBF formula with quantifier prefix Q and matrix M. Since ϕ is a QBF formula (in contrast to DQBF in [1]), we know that Q defines a total order on the universal variables. We now assume the universal variables $u_0, \ldots, u_{n-1}$ of ϕ are ordered according to their appearance in Q, with u_0 (resp. u_{n-1}) being the innermost (resp. outermost) variable.

Translate ϕ to a QF_BV2$_{\ll 1}$ formula Φ by eliminating the quantifier prefix and translating the matrix as follows:

Step 1. Replace Boolean constants 0 and 1 with $0^{[2^n]}$ resp. $\sim 0^{[2^n]}$ and logical connectives with corresponding bitwise bit-vector operations (e.g. $\wedge$ with &). Let Φ' denote the formula generated so far. Extend it to the formula $\left(\Phi' = \sim 0^{\left[2^k\right]}\right)$.

Step 2. For each universal variable $u_m \in \{u_0, \ldots, u_{n-1}\}$,

1. translate (all the occurrences of) u_m to a new bit-vector variable ${U_m}^{[2^n]}$;
2. in order to assign a binary magic number to ${U_m}^{[2^n]}$, add the following equation (i.e., conjunct it with the current formula):

$$ {U_m}^{[2^n]} = \mathit{binmagic}\,(2^m,\ 2^n) $$

Step 3. For an existential variable e depending on $\mathit{Deps}(e) = \{u_m, \ldots, u_{n-1}\}$, with u_m being the innermost universal variable that e depends on,

1. translate (all the occurrences of) e to a new bit-vector variable $E^{[2^n]}$;
2. if $\mathit{Deps}(e) = \varnothing$ add the following equation:

$$ (E \ \& \ \sim 1) = (E \ll 1) \tag{1} $$

otherwise, if $m \neq 0$ add the two equations:

$$ U'_m = \sim\big((U_m \ll 1) \oplus U_m\big) \tag{2} $$
$$ (E \ \& \ U'_m) = \big((E \ll 1) \ \& \ U'_m\big) \tag{3} $$

Note that we omitted the bit-widths in the last equations to improve readability. Each bit position of Φ corresponds to the evaluation of ϕ under a specific assignment to the universal variables $u_0, \ldots, u_{n-1}$, and, by construction of

${U_0}^{[2^n]}, \ldots, {U_{n-1}}^{[2^n]}$, all possible assignments are considered. Eqn. (2) creates a bit-vector ${U'_m}^{[2^n]}$ for which each bit equals to 1 if and only if the corresponding universal variable changes its value from one universal assignment to the next.

Of course, Eqn. (2) does not have to be added multiple times, if several existential variables depend on the same universal variable. Eqn. (3) (resp. Eqn. (1)) ensures that the corresponding bits of $E^{[2^n]}$ satisfy the dependency scheme of ϕ by only allowing the value of e to change if an outer universal variable takes a different value. If $m = 0$, i.e. if e depends on all universal variables, Eqn. (2) evaluates to ${U'_0}^{[2^n]} = 0$, and as a consequence Eqn. (3) simplifies to *true*. Because of this no constraints need to be added for $m = 0$. A similar approach used for translating QBF to Symbolic Model Verification (SMV) can be found in [21]. See also [19] for a translation from QBF to Sequential Circuits. $\square$

Lemma 2. QF_BV2$_{\ll 1}$ *can be (polynomially) reduced to Sequential Circuits.*

Proof. In [10,15], the authors give a translation from quantifier-free Presburger arithmetic with bitwise operations (QFPAbit) to Sequential Circuits. We can adopt their approach in order to construct a translation for QF_BV2$_{\ll 1}$. The main difference between QFPAbit and QF_BV2$_{\ll 1}$ is the fact that bit-vectors of arbitrary, non-fixed, size are allowed in QFPAbit while all bit-vectors contained in QF_BV2$_{\ll 1}$ have a fixed bit-width.

Given $\Phi \in$ QF_BV2$_{\ll 1}$ in flat form. Let $x^{[n]}, y^{[n]}$ denote bit-vector variables, $c^{[n]}$ a bit-vector constant, and ${t_1}^{[n]}, {t_2}^{[n]}$ bit-vector terms only containing bit-vector variables and bitwise operations. Following [10,15] we further assume w.l.o.g that Φ only consists of three types of expressions: ${t_1}^{[n]} = {t_2}^{[n]}$, $x^{[n]} = c^{[n]}$, and $x^{[n]} = y^{[n]} \ll 1^{[n]}$, since any QF_BV2$_{\ll 1}$ formula can be written like this with only a linear growth in the number of original variables.

We encode each equality in Φ separately into an atomic Sequential Circuit. Compared to [10,15], two modifications are needed. First, we need to give a translation for $x = y \ll 1$ to Sequential Circuits. This can be done for example by using the Sequential Circuit for $x = 2 \cdot y$ in QFPAbit. However, a direct translation can also easily be constructed.

The second modification relates to dealing with *fixed-size* bit-vectors. Let n be the bit-width of all bit-vectors in a given equality. We extend each atomic Sequential Circuit to include a counter (circuit). The counter initially is set to 0 and is incremented by 1 in each clock cycle up to a value of n.

When the counter reaches a value of n, it does not change anymore and the output of the atomic Sequential Circuit is set to the same value as the output in the previous cycle. A counter like this can be realized with $\lceil \log_2(n) \rceil$ gates, i.e. polynomially in the size of Φ. In contrast to the implementation described in [15], we assume that the input streams for all variables start with the least significant bit. However, as already pointed out by the authors in [15], their choice was arbitrary and it is no more complicated to construct the circuits the other way round.

Finally, after constructing atomic circuits, their outputs are combined by logical gates following the Boolean structure of Φ, in the same way as for unbounded bit-width in [10,15]. Due to adding counters, we ensure that for every input stream x_i, only the first n_i bits of x_i influence the result of the whole circuit. $\square$

For the proof of Thm. 2, we need the following definition and lemma from [1]:

Definition 3 (Bit-Width Bounded Formula Set [1]). *Given a formula Φ, we denote the* maximal bit-width *in Φ with $max_{bw}(\Phi)$. An infinite set S of bit-vector formulas is* (polynomially) bit-width bounded*, if there exists a polynomial function $p : \mathbb{N} \mapsto \mathbb{N}$ such that $\forall \Phi \in S.\ max_{bw}(\Phi) \leqslant p(|\Phi|)$.*

Lemma 3 ([1]). *$S \in \mathrm{NP}$ for any bit-width bounded formula set $S \subseteq$ QF_BV2.*

Theorem 3. QF_BV2$_{bw}$ *is* NP*-complete.*

Proof. Since *Boolean Formulas* are a subset of QF_BV2$_{bw}$, NP-hardness follows directly. To show that QF_BV2$_{bw} \in$ NP, we give a reduction from QF_BV2$_{bw}$ to a *bit-width bounded* set of formulas. The claim then follows from Lemma 3.

Given a formula $\Phi \in$ QF_BV2$_{bw}$ in flat form. If Φ contains any constants $c^{[n]} \neq 0^{[n]}$, we remove those constants in a (polynomial) pre-processing step. Let ${c_{max}}^{[n]} = b_{k-1} \dots b_1 b_0$ be the largest constant in Φ denoted in binary representation with $b_{k-1} = 1$ and arbitrary bits $b_{k-2}, \dots, b_0$. We now replace each equality ${t_1}^{[m]} = {t_2}^{[m]}$ in Φ with

$$({t_{1,k'-1}}^{[1]} = {t_{2,k'-1}}^{[1]}) \ \& \ \dots \ \& \ ({t_{1,0}}^{[1]} = {t_{2,0}}^{[1]})$$

where $k' = min\{m, k\}$, and, if $m > k$, we additionally add

$$\& \ ({t_{1,hi}}^{[m-k]} = {t_{2,hi}}^{[m-k]})$$

For $0 \leqslant i < k$, we use $({t_{1,i}}^{[1]} = {t_{2,i}}^{[1]})$ to express the ith row of the original equality. All occurrences of a variable $x^{[m]}$ are replaced with a new variable ${x_i}^{[1]}$. All occurrences of a constant $c^{[m]}$ are replaced with $0^{[1]}$ if the ith bit of the constant is 0, and by $\sim 0^{[1]}$ otherwise.

In a similar way, if $m > k$, $({t_{1,hi}}^{[m-k]} = {t_{2,hi}}^{[m-k]})$ represents the remaining $(m-k)$ rows of the original equality corresponding to the most significant bits. All occurrences of a variable $x^{[m]}$ are replaced with a new variable ${x_{hi}}^{[m-k]}$ and all occurrences of a constant $c^{[m]}$ are replaced with $0^{[m-k]}$. Since this pre-processing step is logarithmic in the value of c_{max}, it is polynomial in $|\Phi|$. Without loss of generality, we now assume that Φ does not contain any bit-vector constants different from $0^{[n]}$.

We now construct a formula Φ' by reducing the bit-widths of all bit-vector terms in Φ. Each term $t^{[n]}$ in Φ with bit-width n is replaced with a term $t^{[n']}$, with $n' := min\{n, |\Phi|\}$. Apart from this, Φ' is exactly the same as Φ. As a consequence, $max_{bw}(\Phi') \leqslant |\Phi|$. The set of formulas constructed in this way is bit-width bounded according to Def. 3.

To complete our proof, we now have to show that the proposed reduction is sound, i.e. out of every satisfying assignment to the bit-vector variables ${x_1}^{[n_1]}, \dots, {x_k}^{[n_k]}$ for Φ we can also construct a satisfying assignment to ${x_1}^{[n'_1]}, \dots, {x_k}^{[n'_k]}$ for Φ' and vice versa.

It is easy to see that whenever we have a satisfying assignment α' for Φ', we can construct a satisfying assignment α for Φ. This can be done by simply setting all additional bits of all bit-vector variables to the same value as the most significant bit of the corresponding original vector, i.e. by performing a signed extension. Since all equalities still evaluate to the same value under the extended assignment, $\alpha(F) = \alpha'(F')$ for all equalities F (resp. F') of Φ (resp. Φ'). As a direct consequence, $\alpha(\Phi) = \alpha'(\Phi) = 1$.

The other direction needs slightly more reasoning. Given α, with $\alpha(\Phi) = 1$, we need to construct α', with $\alpha'(\Phi') = 1$. Again, we want to ensure that $\alpha'(F') = \alpha(F)$ for all equalities F (resp. F') in Φ (resp. Φ').

In each variable ${x_i}^{[n_i]}$, $i \in \{1, \dots, k\}$, we are going to select some of the bits. For each equality F with $\alpha(F) = 0$, we select a bit-index as a witness for its evaluation. If $\alpha(F) = 1$, we select an arbitrary bit-index. We then mark the selected bit-index in all bit-vector variables contained in F, as well as in all other bit-vector variables of the same bit-width. Having done this for all equalities, we end up with sets M_i of selected bit-indices, for all $i \in \{1, \dots, k\}$, where

$$\begin{aligned} &|M_i| \leqslant min\{n_i, |\Phi|\} \\ &M_i = M_j \qquad\qquad \forall j \in \{1, \dots, k\} \text{ with } n_i = n_j \end{aligned}$$

The selected indices contain a witness for the evaluation of each equality. We now add arbitrary further bit-indices, again selecting the same indices in bit-vector variables of the same bit-width, until $|M_i| = min\{n_i, |\Phi|\}\ \forall i \in \{1, \dots, k\}$.

Finally, we can directly construct α' using the selected indices and get $\alpha'(\Phi') = \alpha(\Phi) = 1$ because of the fact that we included a witness for every equality in our index-selection process. Note, that we only had to choose a specific witness for the case that $\alpha(F) = 0$. For $\alpha(F) = 1$, we were able to choose an arbitrary bit-index because every satisfied equality will trivially still be satisfied when only a subset of all bit-indices is considered. $\square$

Remark 1. A similar proof can be found in [16,17]. While the focus of [16,17] lies on improving the practical efficiency of SMT-solvers by reducing the bit-width of a given formula before bit-blasting, the author does not investigate its influence on the complexity of a given problem class. In fact, the author claims that bit-vector theories with common operators are NP-complete. As we have already shown in [1], this only holds if unary encoding on the bit-widths is used. However, unary encoding leads to the fact that the given class of formulas remains NP-complete, independent of whether a reduction of the bit-width is possible. While the arguments on bit-width reduction given in [16,17] still hold for binary encoded bit-vector formulas when only bitwise operators are used, our proof considers the complexity of the problem class.

5 Discussion

The complexity results given in Sec. 4 provide some insight in where the expressiveness of bit-vector logics with binary encoding comes from. While we assume bitwise operations and equality naturally being part of a bit-vector logic, if and to what extent we allow shifts directly determines its complexity. Shifts, in a certain way, allow different bits of a bit-vector to interact with each other. Whether we allow no interaction, interaction between neighbouring bits, or interaction between arbitrary bits is crucial to the expressiveness of bit-vector logics and the complexity of their decision problem.

Additionally, we directly get classifications for various other bit-vector operations: for example, we still remain in PSpace if we add *linear modular arithmetic* to QF_BV2$_{\ll 1}$. This can be seen by replacing expressions $x^{[n]} = y^{[n]} + z^{[n]}$ by

$$\left(x^{[n]} = y^{[n]} \oplus z^{[n]} \oplus {c_{in}}^{[n]}\right) \;\&\; \left({c_{in}}^{[n]} = {c_{out}}^{[n]} \ll 1^{[n]}\right) \;\&$$
$$\left({c_{out}}^{[n]} = \left(x^{[n]} \;\&\; y^{[n]}\right) \mid \left({c_{in}}^{[n]} \;\&\; y^{[n]}\right) \mid \left(x^{[n]} \;\&\; {c_{in}}^{[n]}\right)\right)$$

with new variables ${c_{in}}^{[n]}, {c_{out}}^{[n]}$, and by splitting multiplication by constant into several multiplications by 2 (resp. shift by 1), similar to [10,15]. However, this is not surprising since QFPAbit is already known to be PSpace-complete [15].

More interestingly, we can also extend QF_BV2$_{\ll 1}$ (resp. QFPAbit) by *indexing* (denoted by $x^{[n]}[i]$) without growth in complexity. The counter we introduced in our translation from QF_BV2$_{\ll 1}$ to Sequential Circuits can be used to return the value at a specific bit-index of a bit-vector. Extending QF_BV2$_{\ll 1}$ with additional relational operators like e.g. *unsigned less than* (denoted by $x^{[n]} <_{\mathbf{u}} y^{[n]}$) does not increase complexity, either. For instance, the above expression can be replaced by checking whether $x - y < 0$ holds, which can simply be done by constructing an adder for $x^{[n]} + \left(\sim y^{[n]} + 1^{[n]}\right)$, as shown above, and then check whether overflow occurs, i.e., $\left(y^{[n]} \neq 0^{[n]}\right) \;\&\; \left({c_{out}}^{[n]}[n-1] = 0^{[1]}\right)$.

On the other hand, *slicing* (denoted by $x^{[n]}\,[i:j]$) cannot be added without growth in complexity. To prove this, consider

$$\left(x^{[n]}\,[n-1:c] = y^{[n]}\,[n-c-1:0]\right) \;\&\; \left(x^{[n]}\,[c-1:0] = 0^{[c]}\right)$$

which is equivalent to

$$x^{[n]} = (y^{[n]} \ll c^{[n]})$$

and shows that *slicing* can be used to express shift by constant. Therefore, the resulting logic becomes NExpTime-complete. The same result holds for general *multiplication*. We can use

$$x^{[n]} = (y^{[n]} \cdot 2^{c^{[n]}})$$

to replace shift by constant and use exponentiation by squaring to calculate $2^{c^{[n]}}$ with $\lceil \log_2(n) \rceil$ multiplications.

Note that those results only hold for fixed-size bit-vector logics. For example, allowing *multiplication* (in combination with *addition*) makes non-fixed-size bit-vector logics undecidable [22]. We are not aware of any complexity results concerning non-fixed-size bit-vector logics with *slicing* or *shift by constant*.

6 Conclusion

In this paper, we discussed the complexity of fixed-size bit-vector logics with binary encoding on numbers. In contrast to existing literature, except for [1], where usually it is not distinguished between unary or binary encoding, we argued that it is important to make this distinction. Our results apply to the actually much more natural binary encoding as it is also used in standard formats, e.g. in the SMT-LIB format. In previous work [1], we already showed the quantifier-free case of those bit-vector logics to be NExpTime-complete. We now extended our previous work by analyzing the quantifier-free case in more detail and gave two new complexity results.

In particular, we showed that the complexity of deciding quantifier-free bit-vector logics with bitwise operations and equality depends on whether we allow *shift by constant* ($\mathrm{QF\_BV2}_{\ll c}$), *shift by 1* ($\mathrm{QF\_BV2}_{\ll 1}$), or *no shifts at all* ($\mathrm{QF\_BV2}_{bw}$). While deciding $\mathrm{QF\_BV2}_{\ll c}$ remains NExpTime-complete, we proved that $\mathrm{QF\_BV2}_{\ll 1}$ is PSpace-complete, and $\mathrm{QF\_BV2}_{bw}$ even becomes NP-complete.

In addition to the already previously proposed concept of bit-width boundedness, this gives an alternative way to avoid the increase in complexity that comes with binary encoding in the general case. To be more specific for practical logics, we then looked at the effect some other common operations have on this complexity results. We discussed why logics with *addition*, *multiplication by constant*, *indexing*, and *relational operations* still can be decided in PSpace, and showed that allowing *general multiplication* or *slicing* already leads to NExpTime-completeness.

On the one hand, our theoretical results give an argument for using more powerful solving techniques when dealing with bit-vector logics. Currently the most common approach used in state-of-the-art SMT solvers for bit-vectors is based on simple rewriting, bit-blasting, and SAT solving. We have shown this can possibly produce exponentially larger formulas when a logarithmic encoding is used in the input. As already argued in [1], possible candidates for the general case are techniques used in EPR and/or DQBF solvers (see e.g. [23,24]).

On the other hand, we described various logics that remain in lower complexity classes. For $\mathrm{QF\_BV2}_{bw}$ this shows the importance of bit-width reduction as proposed in [16,17] before bit-blasting. For formulas in $\mathrm{QF\_BV2}_{\ll 1}$ or one of the related classes, only using *shift by 1*, *addition*, *multiplication by constant*, and *indexing*, techniques used in state-of-the-art QBF solvers [25] or symbolic model checking on Sequential Circuits [19] might be of interest.

References

1. Kovásznai, G., Fröhlich, A., Biere, A.: On the complexity of fixed-size bit-vector logics with binary encoded bit-width. In: Proc. SMT'12. (2012) 44–55
2. Cyrluk, D., Möller, O., Rue, H.: An efficient decision procedure for a theory of fixed-sized bitvectors with composition and extraction. In: Proc. CAV'97. Volume 1384 of LNCS. (1997) 60–71

3. Barrett, C.W., Dill, D.L., Levitt, J.R.: A decision procedure for bit-vector arithmetic. In: Proc. DAC'98. (1998) 522–527
4. Bjørner, N., Pichora, M.C.: Deciding fixed and non-fixed size bit-vectors. In: Proc. TACAS'98. Volume 1384 of LNCS. (1998) 376–392
5. Bruttomesso, R., Sharygina, N.: A scalable decision procedure for fixed-width bit-vectors. In: Proc. ICCAD'09, IEEE (2009) 13–20
6. Franzén, A.: Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT. PhD thesis, University of Trento (2010)
7. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard: Version 2.0. In: Proc. SMT'10, Edinburgh, UK (2010)
8. Brummayer, R., Biere, A., Lonsing, F.: BTOR: bit-precise modelling of word-level problems for model checking. In: Proc. BPR 2008, New York, ACM (2008) 33–38
9. Ayari, A., Basin, D.A., Klaedtke, F.: Decision procedures for inductive boolean functions based on alternating automata. In: Proc. CAV'00. Volume 1855 of LNCS. (2000) 170–186
10. Spielmann, A., Kuncak, V.: Synthesis for unbounded bit-vector arithmetic. In: Proc. IJCAR'12. Volume 7364 of LNCS. (2012) 499–513
11. Bryant, R.E., Kroening, D., Ouaknine, J., Seshia, S.A., Strichman, O., Brady, B.: Deciding bit-vector arithmetic with abstraction. In: Proc. TACAS'07. Volume 4424 of LNCS. (2007) 358–372
12. Wintersteiger, C.M., Hamadi, Y., de Moura, L.M.: Efficiently solving quantified bit-vector formulas. In: Proc. FMCAD'10, IEEE (2010) 239–246
13. Wintersteiger, C.M.: Termination Analysis for Bit-Vector Programs. PhD thesis, ETH Zurich, Switzerland (2011)
14. Cook, B., Kroening, D., Rümmer, P., Wintersteiger, C.M.: Ranking function synthesis for bit-vector relations. In: Proc. TACAS'10. Volume 6015 of LNCS. (2010) 236–250
15. Spielmann, A., Kuncak, V.: On synthesis for unbounded bit-vector arithmetic. Technical report, EPFL, Lausanne, Switzerland (February 2012)
16. Johannsen, P.: Reducing bitvector satisfiability problems to scale down design sizes for RTL property checking. In: Proc. HLDVT'01. (2001) 123–128
17. Johannsen, P.: Speeding Up Hardware Verification by Automated Data Path Scaling. PhD thesis, CAU Kiel, Germany (2002)
18. Peterson, G.L., Reif, J.H.: Multiple-person alternation. In: Proc. FOCS'79. (1979) 348–363
19. Prasad, M.R., Biere, A., Gupta, A.: A survey of recent advances in SAT-based formal verification. STTT **7**(2) (2005) 156–173
20. Knuth, D.E.: The Art of Computer Programming, Volume 4A: Combinatorial Algorithms. Addison-Wesley (2011)
21. Donini, F.M., Liberatore, P., Massacci, F., Schaerf, M.: Solving QBF with SMV. In: Proc. KR'02. (2002) 578–589
22. Davis, M., Matijasevich, Y., Robinson, J.: Hilbert's tenth problem: Diophantine equations: positive aspects of a negative solution. In: Proc. Sympos. Pure Mathematics. Volume 28. (1976) 323–378
23. Fröhlich, A., Kovásznai, G., Biere, A.: A DPLL algorithm for solving DQBF. In: Proc. POS'12. (2012)
24. Korovin, K.: iProver — an instantiation-based theorem prover for first-order logic (system description). In: Proc. IJCAR'08. Volume 5195 of LNCS. (2008) 292–298
25. Lonsing, F., Biere, A.: Integrating dependency schemes in search-based QBF solvers. In: Proc. SAT'10. Volume 6175 of LNCS. (2010) 158–171

A Table: Comparison of Completeness Results for Fixed-Size and Non-Fixed-Size Bit-Vector Logics

	fixed-size	*non-fixed-size*
QF_BV2:	NExpTime [1]	undecidable [22]
QF_BV2$_{\ll c}$:		?
QF_BV2$_{\ll 1}$:	PSpace [⋆]	PSpace [10,15]
QF_BV2$_{bw}$:	NP [⋆]	NP [⋆] [2]
Presburger arithmetic:	?	NP

(⋆ cites our current paper)

Table 5. Completeness results for various fixed-size and non-fixed-size bit-vector logics.

B Example: A reduction of QBF to QF_BV2$_{\ll 1}$

Consider the following QBF formula:

$$\begin{aligned}\exists x \forall u_2 \exists y \forall u_1 u_0 \exists z \;.\; & (u_2 \vee u_1 \vee \neg z) \wedge \\ & (u_2 \vee \neg x \vee y) \wedge \\ & (u_0 \vee \neg x \vee \neg z) \wedge \\ & (u_1 \vee \neg y \vee z) \wedge \\ & (u_0 \vee \neg u_1 \vee z)\end{aligned}$$

This QBF formula is *satisfiable*, and is translated to the following QF_BV2$_{\ll 1}$ formula:

$$\begin{aligned}&\Big((U_2 \mid U_1 \mid{\sim} Z) \mathbin{\&} (U_2 \mid{\sim} X \mid Y) \mathbin{\&} (U_0 \mid{\sim} X \mid{\sim} Z) \mathbin{\&} \\ &(U_1 \mid{\sim} Y \mid Z) \mathbin{\&} (U_0 \mid{\sim} U_1 \mid Z)\Big) = {\sim} 0^{[8]} \wedge \\ &\bigwedge_{m \in \{0,1,2\}} \left(\left(\bigwedge_{0 \leqslant i < m} U_i\right) \oplus U_m \;=\; U_m \ll 1\right) \wedge \\ &(X \mathbin{\&} {\sim} 1) \;=\; (X \ll 1) \wedge \\ &\big(U_2' \;=\; {\sim}\big((U_2 \ll 1) \oplus U_2\big)\big) \;\wedge\; \big((Y \mathbin{\&} U_2') \;=\; \big((Y \ll 1) \mathbin{\&} U_2'\big)\big)\end{aligned} \tag{4}$$

[2] Although we did not point this out explicitely, it is easy to check that the proof we gave for the specific fixed-sized bit-vector logic in Thm. 2 still holds for the corresponding non-fixed-size one if we set all bit-widths in Φ' to $n' := |\Phi|$.

Let us note that we omit the bit-widths for the sake of readability, and also because all the bit-vector variables are of bit-width 8.

In the following, let us show that this formula is also satisfiable. Note that $U_0 = 01010101_2{}^{[8]}$, $U_1 = 00110011_2{}^{[8]}$, and $U_2 = 00001111_2{}^{[8]}$. The following table gives some insight into the process of generating these binary magic numbers:

$1 \oplus U_0$	$= U_0 \ll 1$	$\rightarrow\ U_0$	$U_0 \oplus U_1$	$= U_1 \ll 1$	$\rightarrow\ U_1$	$(U_0 \wedge U_1) \oplus U_2$	$= U_2 \ll 1$	$\rightarrow\ U_2$
$\neg U_{0,7}$	$U_{0,6}$	0	$U_{1,7}$	$U_{1,6}$	0	$U_{2,7}$	$U_{2,6}$	0
$\neg U_{0,6}$	$U_{0,5}$	1	$\neg U_{1,6}$	$U_{1,5}$	0	$U_{2,6}$	$U_{2,5}$	0
$\neg U_{0,5}$	$U_{0,4}$	0	$U_{1,5}$	$U_{1,4}$	1	$U_{2,5}$	$U_{2,4}$	0
$\neg U_{0,4}$	$U_{0,3}$	1	$\neg U_{1,4}$	$U_{1,3}$	1	$\neg U_{2,4}$	$U_{2,3}$	0
$\neg U_{0,3}$	$U_{0,2}$	0	$U_{1,3}$	$U_{1,2}$	0	$U_{2,3}$	$U_{2,2}$	1
$\neg U_{0,2}$	$U_{0,1}$	1	$\neg U_{1,2}$	$U_{1,1}$	0	$U_{2,2}$	$U_{2,1}$	1
$\neg U_{0,1}$	$U_{0,0}$	0	$U_{1,1}$	$U_{1,0}$	1	$U_{2,1}$	$U_{2,1}$	1
$\neg U_{0,0}$	0	1	$\neg U_{1,0}$	0	1	$\neg U_{2,0}$	0	1

First, we show how the bits of X get restricted by the constraints introduced above. Let us denote the originally unrestricted bits of X with $x_7, x_6, \ldots, x_0$. Since the bit-vectors

$$\begin{array}{lcl} (X\ \&\ \sim 1) & = & \big(x_7, x_6, x_5, x_4, x_3, x_2, x_1, 0\big) \\ (X \ll 1) & = & \big(x_6, x_5, x_4, x_3, x_2, x_1, x_0, 0\big) \end{array}$$

are forced to be equal, all bits of X have to be equal:

$$X \ := \ \big(x_0, x_0, x_0, x_0, x_0, x_0, x_0, x_0\big)$$

Similarly we get the constraints on Y:

$$U_2' \ := \ \sim\big((U_2 \ll 1) \oplus U_2\big) \ = \ 11101110$$

and therefore

$$\begin{array}{lcl} (Y\ \&\ U_2') & = & \big(y_7, y_6, y_5, 0, y_3, y_2, y_1, 0\big) \\ (Y \ll 1)\ \&\ U_2'\big) & = & \big(y_6, y_5, y_4, 0, y_2, y_1, y_0, 0\big) \end{array}$$

which are forced to be equal. Then we put restrictions on individual bits of Y:

$$Y \ := \ \big(y_4, y_4, y_4, y_4, y_0, y_0, y_0, y_0\big)$$

Finally, Z is not restricted in any way since u_0 is the innermost universal variable that z depends on, i.e. z depends on all universal variables.

$$Z \ := \ \big(z_7, z_6, z_5, z_4, z_3, z_2, z_1, z_0\big)$$

In order to show that the formula (4) is satisfiable, let us evaluate the "clauses" in the formula:

$$\begin{aligned}
(U_2 \mid U_1 \mid\!\sim Z) &= \big(\neg z_7,\ \neg z_6,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1\big)\\
(U_2 \mid\!\sim X \mid Y) &= \big(\neg x_0 \vee y_4,\ \neg x_0 \vee y_4,\ \neg x_0 \vee y_4,\ \neg x_0 \vee y_4,\ 1,\ 1,\ 1,\ 1\big)\\
(U_0 \mid\!\sim X \mid\!\sim Z) &= \big(\neg x_0 \vee \neg z_7,\ 1,\ \neg x_0 \vee \neg z_5,\ 1,\ \neg x_0 \vee \neg z_3,\ 1,\ \neg x_0 \vee \neg z_1,\ 1\big)\\
(U_1 \mid\!\sim Y \mid Z) &= \big(\neg y_4 \vee z_7,\ \neg y_4 \vee z_6,\ 1,\ 1,\ \neg y_0 \vee z_4,\ \neg y_0 \vee z_3,\ 1,\ 1\big)\\
(U_0 \mid\!\sim U_1 \mid Z) &= \big(1,\ 1,\ z_5,\ 1,\ 1,\ 1,\ z_1,\ 1\big)
\end{aligned}$$

By applying *bitwise and* to them, we get the following bit-vector:

$$\Phi' = \begin{pmatrix}
\neg z_7 \wedge (\neg x_0 \vee y_4) \wedge (\neg x_0 \vee \neg z_7) \wedge (\neg y_4 \vee z_7)\\
\neg z_6 \wedge (\neg x_0 \vee y_4) \wedge (\neg y_4 \vee z_6)\\
(\neg x_0 \vee y_4) \wedge (\neg x_0 \vee \neg z_5) \wedge z_5\\
\neg x_0 \vee y_4\\
(\neg x_0 \vee \neg z_3) \wedge (\neg y_0 \vee z_4)\\
\neg y_0 \vee z_3\\
(\neg x_0 \vee \neg z_1) \wedge z_1\\
1
\end{pmatrix}$$

In order to check if $\Phi' = \sim 0^{[8]}$ is satisfiable, one can check the satisfiability of the following simplified clause set:

$$\{\ \neg z_7,\ \neg x_0,\ \neg y_4,\ \neg z_6,\ z_5,\ \neg y_0 \vee z_4,\ \neg y_0 \vee z_3,\ z_1\ \}$$

This can be satisfied e.g. by setting

$$\begin{aligned}
&z_7 = x_0 = y_4 = z_6 = y_0 = 0\\
&z_5 = z_1 = 1
\end{aligned}$$

Therefore,

$$\begin{aligned}
U_0 &= 01010101_2{}^{[8]}, & U_1 &= 00110011_2{}^{[8]}, & U_2 &= 00001111_2{}^{[8]},\\
X &= 00000000_2{}^{[8]}, & Y &= 00000000_2{}^{[8]}, & Z &= 00111111_2{}^{[8]}
\end{aligned}$$

is a possible solution of the bit-vector formula (4).