On the Complexity of Fixed-Size Bit-Vector Logics with Binary Encoded Bit-Width

Gergely Kovásznai, Andreas Fröhlich, 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. In this paper we show that some of these results only hold if unary encoding on the bit-width of bit-vectors is used. We then consider fixed-size bit-vector logics with binary encoded bit-width and establish new complexity results. Our proofs show that binary encoding adds more expressiveness to bit-vector logics, e.g. it makes fixed-size bit-vector logic even without uninterpreted functions nor quantification NExpTime-complete. We also show that under certain restrictions the increase of complexity when using binary encoding can be avoided.

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. Syntax and semantics of *fixed-size bit-vector logics* do not differ much in the literature [9, 3, 4, 11, 6]. Concrete formats for specifying bit-vector problems also exist, like the SMT-LIB format or the BTOR format [5]. Working with *non-fixed-size* bit-vectors has been considered for instance in [4, 1] and more recently in [20], but will not be further discussed in 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 NP-*hardness* of its satisfiability problem. In [6], a sublogic of the previous one is claimed to be NP-*complete*. In [23, 22], the *quantified* case is addressed, and the satisfiability of this logic with uninterpreted functions is proven to be NExpTime-*complete*. The proof holds only if we assume that the bit-widths of the bit-vectors in the input formula are written/encoded in *unary* form. 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 [8, Page 239, Footnote 3]). 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. We investigate how complexity varies if we consider either a unary or a logarithmic (actually without loss of generality) *binary encoding*.

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. We give an example, why in general bit-blasting is not polynomial. Consider checking commutativity of bit-vector addition for two bit-vectors of size one million. Written to a file this formula in SMT2 syntax can be encoded with 138 bytes:

*This work is partially supported by FWF, NFN Grant S11408-N23 (RiSE).

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

Using Boolector [5] with rewriting optimizations switched off (except for structural hashing), bit-blasting produces a circuit of size 103 MB in AIGER format. Tseitin transformation results in a CNF in DIMACS format of size 1 GB. A bit-width of 10 million can be represented by two more bytes in the SMT2 input, but could not bit-blasted anymore with our tool-flow (due to integer overflow). As this example shows, checking bit-vector logics through bit-blasting can not be considered to be a polynomial reduction, which also disqualifies bit-blasting as a sound way to prove that the decision problem for (quantifier-free) bit-vector logics is in NP. We show that deciding bit-vector logics, even without quantifiers, is much harder: it is NExpTime-complete.

Informally speaking, we show that moving from unary to binary encoding for bit-widths increases complexity *exponentially* and that binary encoding has at least as much expressive power as quantification. However we give a sufficient condition for bit-vector problems to remain in the "lower" complexity class, when moving from unary to binary encoding. We call them *bit-width bounded* problems. For such problems it does not matter, whether bit-width is encoded unary or binary. We also discuss some concrete examples from SMT-LIB.

2 Preliminaries

We assume the common syntax for (fixed-size) *bit-vector formulas*, c.f. SMT-LIB and [9, 3, 4, 11, 6, 5]. Every bit-vector possesses a bit-width n, either explicit or implicit, where n is a natural number, $n \geq 1$. We denote a bit-vector constant with $c^{[n]}$, where c is a natural number, $0 \leq c < 2^n$. A variable is denoted with $x^{[n]}$, where x is an identifier. Let us note that no explicit bit-width belongs to bit-vector operators, and, therefore, the bit-width of a compound term is *implicit*, i.e., can be calculated. Let $t^{[n]}$ denote the fact that the bit-vector term t is of bit-width n. We even omit an explicit bit-width if it can be deduced from the context.

In our proofs we use the following *bit-vector operators*: *indexing* ($t^{[n]}[i]$, $0 \leq i < n$), *bitwise negation* ($\sim t^{[n]}$), *bitwise and* ($t_1^{[n]}$ & $t_2^{[n]}$), *bitwise or* ($t_1^{[n]} \mid t_2^{[n]}$), *shift left* ($t_1^{[n]} \ll t_2^{[n]}$), *logical shift right* ($t_1^{[n]} \gg t_2^{[n]}$), *addition* ($t_1^{[n]} + t_2^{[n]}$), *multiplication* ($t_1^{[n]} \cdot t_2^{[n]}$), *unsigned division* ($t_1^{[n]} \,/\, t_2^{[n]}$), and *equality* ($t_1^{[n]} = t_2^{[n]}$). Including other common operations (e.g., slicing, concatenation, extensions, arithmetic right shift, signed arithmetic and relational operators, rotations etc.) does not destroy the validity of our subsequent propositions, since they all can be bit-blasted polynomially in the bit-width of their operands. *Uninterpreted functions* will also be considered. They have an explicit bit-width for the result type. The application of such a function is written as $f^{[n]}(t_1, \ldots, t_m)$, where f is an identifier, and $t_1^{[n_1]}, \ldots, t_m^{[n_m]}$ are terms.

Let QF_BV1 resp. QF_BV2 denote the logics of quantifier-free bit-vectors with unary resp. binary encoded bit-width (without uninterpreted functions). As mentioned before, we prove that the complexity of deciding QF_BV2 is exponentially higher than deciding QF_BV1. This fact is, of course, due to the more succinct encoding. The logics we get by adding *uninterpreted functions* to these logics are denoted by QF_UFBV1 resp. QF_UFBV2. Uninterpreted functions are powerful tools for abstraction, e.g., they can formalize reads on arrays. When *quantification* is introduced, we get the logics BV1 resp. BV2 when uninterpreted functions are prohibited. When they are allowed, we get UFBV1 resp. UFBV2. These latter logics are expressive enough, for instance, to formalize reads and writes on arrays with quantified indices.[1]

[1]Let us emphasize again that among all these logics the ones with binary encoding correspond to the logics

3 Complexity

In this section we discuss the complexity of deciding the bit-vector logics defined so far. We first summarize our results, and then give more detailed proofs for the new non-trivial ones. The results are also summarized in a tabular form in Appendix A.

First, consider *unary encoding* of bit-widths. Without uninterpreted functions nor quantification, i.e., for QF_BV1, the following complexity result can be proposed (for partial results and related work see also [3] and [6]):

Proposition 1. QF_BV1 *is* NP*-complete*[2]

Proof. By bit-blasting, QF_BV1 can be polynomially reduced to *Boolean formulas*, for which the satisfiability problem (SAT) is NP-complete. The other direction follows from the fact that Boolean formulas are actually QF_BV1 formulas whose all terms are of bit-width 1. □

Adding uninterpreted functions to QF_BV1 does not increase complexity:

Proposition 2. QF_UFBV1 *is* NP*-complete.*

Proof. In a formula, uninterpreted functions can be eliminated by replacing each occurrence with a new bit-vector variable and adding (at most quadratic many) Ackermann constraints, e.g. [16, Chapter 3.3.1]. Therefore, QF_UFBV1 can be polynomially translated to QF_BV1. The other direction directly follows from the fact that QF_BV1 $\subset$ QF_UFBV1. □

Adding quantifiers to QF_BV1 yields the following complexity (see also [8]):

Proposition 3. BV1 *is* PSPACE*-complete.*

Proof. By bit-blasting, BV1 can be polynomially reduced to *Quantified Boolean Formulas* (QBF), which is PSPACE-complete. The other direction directly follows from the fact that QBF $\subset$ BV1 (following the same argument as in Prop. 1). □

Adding quantifiers to QF_UFBV1 increases complexity exponentially:

Proposition 4 (see [22])**.** UFBV1 *is* NEXPTIME*-complete.*

Proof. Effectively Propositional Logic (EPR), being NEXPTIME-complete, can be polynomially reduced to UFBV1 [22, Theorem 7]. For completing the other direction, apply the reduction in [22, Theorem 7] combined with the bit-blasting of the bit-vector operations. □

Our main contribution is to give complexity results for the more common logarithmic (actually without loss of generality) *binary encoding*. Even without uninterpreted functions nor quantification, i.e., for QF_BV2, we obtain the same complexity as for UFBV1.

Proposition 5. QF_BV2 *is* NEXPTIME*-complete.*

Proof. It is obvious that QF_BV2 $\in$ NEXPTIME, since a QF_BV2 formula can be translated exponentially to QF_BV1 $\in$ NP (Prop. 1), by a simple unary re-encoding of all bit-widths. The proof that QF_BV2 is NEXPTIME-hard is more complex and given in Sect. 3.1. □

Adding uninterpreted functions to QF_BV2 does not increase complexity, again using Ackermann constraints, as in the proof for Prop. 2:

QF_BV, QF_UFBV, BV, and UFBV used by the SMT community, e.g., in SMT-LIB.

[2] This kind of result is often called unary NP-completeness [14].

Proposition 6. QF_UFBV2 *is* NExpTime*-complete.*

However, adding quantifiers to QF_UFBV2 increases complexity exponentially:

Proposition 7. UFBV2 *is* 2-NExpTime*-complete.*

Proof. Similarly to the proof of Prop. 5, a UFBV2 formula can be exponentially translated to UFBV1 $\in$ NExpTime (Prop. 4), simply by re-encoding all the bit-widths to unary. It is more difficult to prove that UFBV2 is 2-NExpTime-hard, which we show in Sect. 3.2. □

Notice that deciding QF_BV2 has the same complexity as UFBV1. Thus, starting with QF_BV1, re-encoding bit-widths to binary gives the same expressive power, in a precise complexity theoretical sense, as introducing uninterpreted functions and quantification all together. Thus it is important to differentiate between unary and binary encoding of bit-widths in bit-vector logics. Our results show that binary encoding is at least as expressive as quantification, while only the latter has been considered in [23, 22].

3.1 QF_BV2 is NExpTime-hard

In order to prove that QF_BV2 is NExpTime-hard, we pick a NExpTime-hard problem and, then, we reduce it to QF_BV2. Let us choose the satisfiability problem of *Dependency Quantified Boolean Formulas* (DQBF), which has been shown to be NExpTime-complete [2].

In DQBF, quantifiers are not forced to be totally ordered. Instead a partial order is explicitly expressed in the form $e(u_1, \ldots, u_m)$, stating that an existential variable e depends on the universal variables $u_1, \ldots, u_m$, where $m \geq 0$. Given an existential variable e, we will use $Deps(e)$ to denote the set of universal variables that e depends on. A more formal definition can be found in [2]. Without loss of generality, we can assume that a DQBF formula is in clause normal form.

In the proof, we are going to apply bitmasks of the form

$$\overbrace{\underbrace{0\ldots0}_{2^i}\underbrace{1\ldots1}_{2^i}\ldots\underbrace{0\ldots0}_{2^i}\underbrace{1\ldots1}_{2^i}}^{2^n}$$

Given $n \geq 1$ and i, with $0 \leq i < n$, we denote such a bitmask with M_i^n. Notice that these bitmasks correspond to the *binary magic numbers* [12] (see also Chpt. 7 of [21]), and, can thus arithmetically be calculated in the following way (actually as sum of a geometric series):

$$M_i^n \quad := \quad \frac{2^{(2^n)} - 1}{2^{(2^i)} + 1}$$

In order to reformulate this definition in terms of bit-vectors, the numerator can be written as $\sim 0^{[2^n]}$, and $2^{(2^i)}$ as $1 \ll (1 \ll i)$, which results in the following bit-vector expression:

$$M_i^n \quad := \quad \sim 0^{[2^n]} \;/\; \big((1 \ll (1 \ll i)) + 1\big) \tag{1}$$

Theorem 8. DQBF *can be (polynomially) reduced to* QF_BV2*.*

Proof. The basic idea is to use bit-vector logic to encode function tables in an exponentially more succinct way, which then allows to characterize independence of an existential variable from a particular universal variable polynomially.

More precisely, we will use binary magic numbers, as constructed in Eqn. (1), to create a certain set of fully-specified exponential-size bit-vectors by using a polynomial expression, due to binary encoding. We will then formally point out the well-known fact that those bit-vectors correspond exactly to the set of *all assignments*. We can then use a polynomial-size bit-vector formula for *cofactoring Skolem-functions* in order to express independency constraints.

First, we describe the reduction (c.f. an example in Appendix B), then show that the reduction is polynomial, and, finally, that it is correct.

The reduction. Given a DQBF formula $\phi := Q.m$ consisting of a quantifier prefix Q and a Boolean CNF formula m called the *matrix* of ϕ. Let $u_0, \ldots, u_{k-1}$ denote all the universal variables that occur in ϕ. Translate ϕ to a QF_BV2 formula Φ by eliminating the quantifier prefix and translating the matrix as follows:

Step 1. Replace Boolean constants 0 and 1 with $0^{[2^k]}$ resp. $\sim 0^{[2^k]}$ and logical connectives with corresponding bitwise bit-vector operators ($\vee$, $\wedge$, $\neg$ with $|$, & , $\sim$, resp.).

Let Φ' denote the formula generated so far. Extend it to the formula $\left(\Phi' = \sim 0^{[2^k]}\right)$.

Step 2. For each u_i,

1. translate (all the occurrences of) u_i to a new bit-vector variable $U_i^{[2^k]}$;
2. in order to assign the appropriate bitmask of Eqn. (1) to U_i, add the following equation (i.e., conjunct it with the current formula):

$$U_i \quad = \quad M_i^k \tag{2}$$

For an optimization see Remark 9 further down.

Step 3. For each existential variable e depending on universals $Deps(e) \subseteq \{u_0, \ldots, u_{k-1}\}$,

1. translate (all the occurrences of) e to a new bit-vector variable $E^{[2^k]}$;
2. for each $u_i \notin Deps(e)$, add the following equation:

$$(E \ \& \ U_i) \quad = \quad \Big(\big(E \gg (1 \ll i)\big) \ \& \ U_i\Big) \tag{3}$$

As it is going to be detailed in the rest of the proof, the above equations enforce the corresponding bits of $E^{[2^k]}$ to satisfy the dependency scheme of ϕ. More precisely, Eqn. (3) makes sure that the positive and negative cofactors of the Skolem-function representing e with respect to an independent variable u_i have the same value.

Polynomiality. Let us recall that all the bit-widths are encoded *binary* in the formula Φ, and thus exponential bit-widths (2^k) are encoded into linear many (k) bits. We show now that each reduction step results in polynomial growth of the formula size.

Step 1 may introduce additional bit-vector constants to the formula. Their bit-width is 2^k, therefore, the resulting formula is bounded quadratically in the input size. *Step 2* adds k variables $U_i^{[2^k]}$ for the original universal variables, as well as k equations as restrictions. The bit-widths of added variables and constants is 2^k. Thus the size of the added constraints is bounded quadratically in the input size. *Step 3* adds one bit-vector variable $E^{[2^k]}$ and at most k constraints for each existential variable. Thus the size is bounded cubically in the input size.

Correctness. We show the original ϕ and the result Φ of the translation to be equisatisfiable. Consider one bit-vector variable U_i introduced in Step 2. In the following, we formalize the well-known fact that all the U_is correspond exactly to *all assignments*. By construction, all bits of U_i are fixed to some constant value. Additionally, for every bit-vector index $b_m \in [0, 2^k - 1]$ there exists a bit-vector index $b_n \in [0, 2^k - 1]$ such that

$$U_i[b_m] \neq U_i[b_n] \text{ and} \tag{4a}$$

$$U_j[b_m] = U_j[b_n], \ \forall j \neq i. \tag{4b}$$

Actually, let us define b_n in the following way (considering the 0th bit the least significant):

$$b_n \quad := \quad \begin{cases} b_m - 2^i & \text{if } U_i[b_m] = 0 \\ b_m + 2^i & \text{if } U_i[b_m] = 1 \end{cases}$$

By defining b_n this way, Eqn. (4a) and (4b) both hold, which can be seen as follows. Let $R(c, l)$ be the bit-vector of length l with each bit set to the Boolean constant c. Eqn. (4a) holds, since, due to construction, U_i consists of several (2^{k-1-i}) concatenated bit-vector fragments $0 \ldots 01 \ldots 1 = R(0, 2^i)R(1, 2^i)$ (with both 2^i zeros and 2^i ones). Therefore it is easy to see that $U_i[b_m] \neq U_i[b_m - 2^i]$ (resp. $U_i[b_m] \neq U_i[b_m + 2^i]$) holds if $U_i[b_m] = 0$ (resp. $U_i[b_m] = 1$). With a similar argument, we can show that Eqn. (4b) holds: $U_j[b_m] = U_j[b_m - 2^i]$ (resp. $U_j[b_m] = U_j[b_m + 2^i]$) if $U_j[b_m] = 0$ (resp. $U_j[b_m] = 1$), since $b_m - 2^i$ (resp. $b_m + 2^i$) is located either still in the same half or already in a concatenated copy of a $R(0, 2^j)R(1, 2^j)$ fragment, if $j \neq i$.

Now consider all possible assignments to the universal variables of our original DQBF-formula ϕ. For a given assignment $\alpha \in \{0, 1\}^k$, the existence of such a previously defined b_n for every U_i and b_m allows us to iteratively find a b_α such that $(U_0[b_\alpha], \ldots, U_{k-1}[b_\alpha]) = \alpha$. Thus, we have a *bijective mapping* of every *universal assignment* α in ϕ to a *bit-vector index* b_α in Φ.

In Step 3 we first replace each existential variable e with a new bit-vector variable E, which can take $2^{(2^k)}$ different values. The value of each individual bit $E[b_\alpha]$ corresponds to the value e takes under a given assignment $\alpha \in \{0, 1\}^k$ to the universal variables. Note that without any further restriction, there is no connection between the different bits in E and therefore the vector represents an arbitrary Skolem-function for an existential variable e. It may have different values for all universal assignments and thus would allow e to depend on all universals.

If, however, e does not depend on a universal variable u_i, we add the constraint of Eqn. (3). In DQBF, independence can be formalized in the following way: e does not depend on u_i if e has to take the same value in the case of all pairs of universal assignments $\alpha, \beta \in \{0, 1\}^k$ where $\alpha[j] = \beta[j]$ for all $j \neq i$. Exactly this is enforced by our constraint. We have already shown that for α we have a corresponding bit-vector index b_α, and we have defined how we can construct a bit-vector index b_β for β. Our constraint for independence ensures that $E[b_\alpha] = E[b_\beta]$.

Step 1 ensures that all logical connectives and all Boolean constants are consistent for each bit-vector index, i.e. for each universal assignment, and that the matrix of ϕ evaluates to 1 for each universal assignment.

□

Remark 9. Using Eqn. (1) in Eqn. (2) seems to require the use of *division*, which, however, can easily be eliminated by rewriting Eqn. (2) to

$$\Big(U_i \cdot \big((1 \ll (1 \ll i)) + 1\big)\Big) \quad = \quad \sim 0^{[2^k]}$$

Multiplication in this equation can then be eliminated by rewriting it as follows:

$$\Big((U_i \ll (1 \ll i)) + U_i\Big) \quad = \quad \sim 0^{[2^k]}$$

3.2 UFBV2 is 2-NExpTime-hard

In order to prove that UFBV2 is 2-NExpTime-hard, we pick a 2-NExpTime-hard problem and then, we reduce it to UFBV2. We can find such a problem among the so-called *domino tiling* problems [7]. Let us first define what a domino system is, and then we specify a 2-NExpTime-hard problem on such systems.

Definition 10 (Domino System)**.** A domino system is a tuple $\langle T, H, V, n\rangle$, where

- T is a finite set of *tile types*, in our case, $T = [0, k-1]$, where $k \geq 1$;
- $H, V \subseteq T \times T$ are the horizontal and vertical matching conditions, respectively;
- $n \geq 1$, encoded *unary*.

Let us note that the above definition differs (but not substantially) from the classical one in [7], in the sense that we use sub-sequential natural numbers for identifying tiles, as it is common in recent papers. Similarly to [17] and [18], the size factor n, encoded *unary*, is part of the input. However while a start tile α and a terminal tile ω is used usually, in our case the starting tile is denoted by 0 and the terminal tile by $k-1$, without loss of generality.

There are different domino tiling problems examined in the literature. In [7] a classical tiling problems is introduced, namely the *square tiling problem*, which can be defined as follows.

Definition 11 (Square Tiling)**.** Given a domino system $\langle T, H, V, n\rangle$, an $f(n)$-square tiling is a mapping $\lambda : [0, f(n)-1] \times [0, f(n)-1] \mapsto T$ such that

- the first row starts with the start tile: $\lambda(0,0) = 0$
- the last row ends with the terminal tile: $\lambda(f(n)-1, f(n)-1) = k-1$
- all horizontal matching conditions hold: $\big(\lambda(i,j), \lambda(i,j+1)\big) \in H \ \forall i < f(n), j < f(n)-1$
- all vertical matching conditions hold: $\big(\lambda(i,j), \lambda(i+1,j)\big) \in V \ \forall i < f(n)-1,\ j < f(n)$

In [7], a general theorem on the complexity of domino tiling problems is proved:

Theorem 12 (from [7])**.** *The $f(n)$-square tiling problem is* NTime$\,(f(n))$*-complete.*

Since for completing our proof on UFBV2 we need a 2-NExpTime-hard problem, let us emphasize the following easy corollary:

Corollary 13. *The $2^{(2^n)}$-square tiling problem is* 2-NExpTime*-complete.*

Theorem 14. *The $2^{(2^n)}$-square tiling problem can be (polynomially) reduced to* UFBV2*.*

Proof. Given a domino system $\langle T = [0, k-1], H, V, n\rangle$, let us introduce the following notations which we intend to use in the resulting UFBV2 formula.

- Represent each tile in T with the corresponding bit-vector of bit-width $l := \lceil \log k \rceil$.
- Represent the horizontal and vertical matching conditions with the uninterpreted functions (predicates) $h^{[1]}(t_1^{[l]}, t_2^{[l]})$ and $v^{[1]}(t_1^{[l]}, t_2^{[l]})$, respectively.
- Represent the tiling with an uninterpreted function $\lambda^{[l]}(i^{[2^n]}, j^{[2^n]})$. As it is obvious, λ represents the type of the tile in the cell at the row index i and column index j. Notice that the bit-width of i and j is exponential in the size of the domino system, but due to *binary encoding* it can represented polynomially.

The resulting UFBV2 formula is the following:

$$\lambda(0,0) = 0 \quad \wedge \quad \lambda\Big(2^{(2^n)}-1, 2^{(2^n)}-1\Big) = k-1 \quad \wedge \bigwedge_{(t_1,t_2)\in H} h(t_1,t_2) \quad \wedge \bigwedge_{(t_1,t_2)\in V} v(t_1,t_2)$$

$$\wedge \quad \forall i,j \left(\begin{array}{c} \Big(\; j < 2^{(2^n)}-1 \;\Rightarrow\; h\big(\lambda(i,j), \lambda(i,j+1)\big) \; \Big) \\ \wedge \\ \Big(\; i < 2^{(2^n)}-1 \;\Rightarrow\; v\big(\lambda(i,j), \lambda(i+1,j)\big) \; \Big) \end{array} \right)$$

This formula contains four kinds of constants. Three can be encoded directly ($0^{[2^n]}$, $0^{[l]}$, and $(k-1)^{[l]}$). However, the constant $2^{(2^n)}-1$ has to be treated in a special way, in order to avoid double exponential size, namely in the following form: $\sim 0^{[2^n]}$. The size of the resulting formula, due to *binary encoding* of the bit-width, is polynomial in the size of the domino system. □

4 Problems Bounded in Bit-Width

We are going to introduce a sufficient condition for bit-vector problems to remain in the "lower" complexity class, when re-encoding bit-width from unary to binary. This condition tries to capture the bounded nature of bit-width in certain bit-vector problems.

In any bit-vector formula, there has to be at least one term with explicit specification of its bit-width. In the logics we are dealing with, only a variable, a constant, or an uninterpreted function can have *explicit bit-width*. Given a formula ϕ, let us denote the *maximal explicit bit-width* in ϕ with $max_{bw}(\phi)$. Furthermore, let $size_{bw}(\phi)$ denote *the number of terms with explicit bit-width* in ϕ.

Definition 15 (Bit-Width Bounded Formula Set)**.** 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 $\forall \phi \in S.\ max_{bw}(\phi) \leq p(size_{bw}(\phi))$.

Proposition 16. *Given a bit-width bounded set S of formulas with binary encoded bit-width, any $\phi \in S$ grows polynomially when re-encoding the bit-widths to unary.*

Proof. Let ϕ' denote the formula obtained through re-encoding bit-widths in ϕ to *unary*. For the size of ϕ' the following upper bound can be shown: $|\phi'| \leq size_{bw}(\phi) \cdot max_{bw}(\phi) + c$. Notice that $size_{bw}(\phi) \cdot max_{bw}(\phi)$ is an upper bound on the sum over the sizes of all the terms with *explicit bit-width* in ϕ'. The constant c represents the size of the rest of the formula. Since S is *bit-width bounded*, it holds that

$$|\phi'| \;\leq\; size_{bw}(\phi) \cdot max_{bw}(\phi) + c \;\leq\; size_{bw}(\phi) \cdot p(size_{bw}(\phi)) + c \;\leq\; |\phi| \cdot p(|\phi|) + c$$

where p is a polynomial function. Therefore, the size of ϕ' is *polynomial* in the size of ϕ. □

By applying this proposition to the logics of Sect. 2 we get:

Corollary 17. *Let us assume a bit-width bounded set S of bit-vector formulas. If $S \subseteq$* QF_UFBV2 *(and even if $S \subseteq$* QF_BV2*), then $S \in$* NP*. If $S \subseteq$* BV2*, then $S \in$* PSpace*. If $S \subseteq$* UFBV2*, then $S \in$* NExpTime*.*

4.1 Benchmark Problems

In this section we discuss concrete SMT-LIB benchmark problems, and whether they are bit-width bounded. Since in SMT-LIB bit-widths are encoded logarithmically and quantification on bit-vectors is not (yet) addressed, we have picked benchmarks from `QF_BV`, which can be considered as QF_BV2 formulas.

First consider the benchmark family `QF_BV/brummayerbiere2/umulov2bw`b, which represent instances of an *unsigned multiplication overflow detection* equivalence checking problem, and is parameterized by the bit-width of unsigned multiplicands (b). We show that the set of these benchmarks, with $b \in \mathbb{N}$, is *bit-width bounded*, and therefore is in NP. This problem checks that a certain (unsigned) overflow detection unit, defined in [19], gives the same result as the following condition: if the $b/2$ most significant bits of the multiplicands are zero, then no overflow occurs. It requires $2 \cdot (b-2)$ variables and a fixed number of constants to formalize the overflow detection unit, as detailed in [19]. The rest of the formula contains only a fixed number of variables and constants. The maximal bit-width in the formula is b. Therefore, the (maximal explicit) bit-width is linearly bounded in the number of variables and constants.

The benchmark family `QF_BV/brummayerbiere3/mulhs`b represents instances of computing the *high-order half of product* problem, parameterized by the bit-width of unsigned multiplicands (b). In this problem the high-order $b/2$ bits of the product are computed, following an algorithm detailed in [21, Page 132]. The maximal bit-width is b and the number of variables and constants to formalize this problem is fixed, i.e., independent of b. Therefore, the (maximal explicit) bit-width is *not bounded* in the number of variables and constants.

The family `QF_BV/bruttomesso/lfsr/lfsr`t_b_n formalizes the behaviour of a *linear feedback shift register* [6]. Since, by construction, the bit-width (b) and the number (n) of registers do not correlate, and only n variables are used, this benchmark problem is *not bit-width bounded*.

5 Conclusion

We discussed complexity of deciding various quantified and quantifier-free fixed-size bit-vector logics. In contrast to existing literature, where usually it is not distinguished between unary or binary encoding of the bit-width, we argued that it is important to make this distinction. Our new results apply to the actual much more natural binary encoding as it is also used in standard formats, e.g. in the SMT-LIB format.

We proved that deciding QF_BV2 is NExpTime-complete, which is the same complexity as for deciding UFBV1. This shows that binary encoding for bit-widths has at least as much expressive power as quantification does. We also proved that UFBV2 is 2-NExpTime-complete. The complexity of deciding BV2 remains unclear. While it is easy to show ExpSpace-inclusion for BV2 by bit-blasting to an exponential-size QBF, and NExpTime-hardness follows directly from QF_BV2 $\subset$ BV2, it is not clear whether QF_BV2 is complete for any of these classes.

We also showed that under certain conditions on bit-width the increase of complexity that comes with a binary encoding can be avoided. Finally, we gave examples of benchmark problems that do or do not fulfill this condition. As future work it might be interesting to consider our results in the context of parametrized complexity [10].

Our theoretical results give an argument for using more powerful solving techniques. 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 as an input. Possible candidates are techniques used in EPR and/or (D)QBF solvers (see e.g. [13, 15]).

References

[1] Abdelwaheb Ayari, David A. Basin, and Felix Klaedtke. Decision procedures for inductive boolean functions based on alternating automata. In *CAV*, volume 1855 of *LNCS*. Springer, 2000.

[2] Salman Azhar, Gary Peterson, and John Reif. Lower bounds for multiplayer non-cooperative games of incomplete information. *Computers & Mathematics with Applications*, 41:957–992, 2001.

[3] Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. A decision procedure for bit-vector arithmetic. In *Proceedings of the 35th Design Automation Conference*, pages 522–527, 1998.

[4] Nikolaj Bjørner and Mark C. Pichora. Deciding fixed and non-fixed size bit-vectors. In *TACAS*, volume 1384 of *LNCS*, pages 376–392. Springer, 1998.

[5] Robert Brummayer, Armin Biere, and Florian Lonsing. BTOR: bit-precise modelling of word-level problems for model checking. In *Proc. 1st International Workshop on Bit-Precise Reasoning*, pages 33–38, New York, NY, USA, 2008. ACM.

[6] Roberto Bruttomesso and Natasha Sharygina. A scalable decision procedure for fixed-width bit-vectors. In *ICCAD*, pages 13–20. IEEE, 2009.

[7] Bogdan S. Chlebus. From domino tilings to a new model of computation. In *Symposium on Computation Theory*, volume 208 of *LNCS*. Springer, 1984.

[8] Byron Cook, Daniel Kroening, Philipp Rümmer, and Christoph M. Wintersteiger. Ranking function synthesis for bit-vector relations. In *TACAS*, volume 6015 of *LNCS*. Springer, 2010.

[9] David Cyrluk, Oliver Möller, and Harald Rueß. An efficient decision procedure for a theory of fixed-sized bitvectors with composition and extraction. In *Computer-Aided Verification (CAV '97)*, pages 60–71. Springer, 1997.

[10] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. 530 pp.

[11] Anders Franzén. *Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT*. PhD thesis, University of Trento, 2010.

[12] Edwin E. Freed. Binary magic numbers – some applications and algorithms. *Dr. Dobb's Journal of Software Tools*, 8(4):24–37, 1983.

[13] Andreas Fröhlich, Gergely Kovásznai, and Armin Biere. A DPLL algorithm for solving DQBF. In *Pragmatics of SAT 2012*, 2012. to appear.

[14] Michael R. Garey and David S. Johnson. "Strong" NP-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, July 1978.

[15] Konstantin Korovin. iProver — an instantiation-based theorem prover for first-order logic (system description). In *Proc. IJCAR'08*, IJCAR '08. Springer, 2008.

[16] Daniel Kroening and Ofer Strichman. *Decision Procedures: An Algorithmic Point of View*. Texts in Theoretical Computer Science. Springer, 2008.

[17] Maarten Marx. Complexity of modal logic. In *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 139–179. Elsevier, 2007.

[18] Matthias Niewerth and Thomas Schwentick. Two-variable logic and key constraints on data words. In *ICDT*, pages 138–149, 2011.

[19] Michael J. Schulte, Mustafa Gok, Pablo I. Balzola, and Robert W. Brocato. Combined unsigned and two's complement saturating multipliers. In *Proceedings of SPIE : Advanced Signal Processing Algorithms, Architectures, and Implementations*, pages 185–196, July 2000.

[20] Andrej Spielmann and Viktor Kuncak. On synthesis for unbounded bit-vector arithmetic. Technical report, EPFL, Lausanne, Switzerland, February 2012.

[21] Henry S. Warren. *Hacker's Delight*. Addison-Wesley Longman, 2002.

[22] Christoph M. Wintersteiger. *Termination Analysis for Bit-Vector Programs*. PhD thesis, ETH Zurich, Switzerland, 2011.

[23] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. Efficiently solving quantified bit-vector formulas. In *Proc. FMCAD*, pages 239–246. IEEE, 2010.

A Table: Completeness results for bit-vector logics

		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 for various bit-vector logics considering different encodings

B Example: A reduction of DQBF to QF_BV2

Consider the following DQBF formula:

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

This DQBF formula is *unsatisfiable*. Let us note that by adding one more dependency for y, or even by making x and y dependent on all u_is, the resulting QBF formula becomes satisfiable.

Using the reduction in Sect. 3.1, this formula is translated to the following QF_BV2 formula:

$$\begin{aligned}&\Big((X \mid Y \mid\!\sim U_0 \mid\!\sim U_1)\ \&\ (X \mid\!\sim Y \mid U_0 \mid\!\sim U_1 \mid\!\sim U_2)\ \&\ (X \mid\!\sim Y \mid\!\sim U_0 \mid\!\sim U_1 \mid U_2)\ \& \\ &(\sim X \mid Y \mid\!\sim U_0 \mid\!\sim U_2)\ \&\ (\sim X \mid\!\sim Y \mid U_0 \mid U_1 \mid\!\sim U_2)\Big) = \sim 0^{[8]} \wedge \\ &\bigwedge_{i \in \{0,1,2\}} \Big(\big((U_i \ll (1 \ll i)) + U_i\big)\ = \sim 0^{[8]}\Big) \wedge \\ &(X\ \&\ U_1)\ =\ \big((X \gg (1 \ll 1))\ \&\ U_1\big) \wedge \\ &(X\ \&\ U_2)\ =\ \big((X \gg (1 \ll 2))\ \&\ U_2\big) \wedge \\ &(Y\ \&\ U_0)\ =\ \big((Y \gg (1 \ll 0))\ \&\ U_0\big)\end{aligned} \tag{5}$$

In the following, let us show that this formula is also unsatisfiable. Note that $M_0^3 = 55_{16}^{[8]} = 01010101_2^{[8]}$, $M_1^3 = 33_{16}^{[8]} = 00110011_2^{[8]}$, and $M_2^3 = 0\mathrm{F}_{16}^{[8]} = 00001111_2^{[8]}$, where "$\cdot_{16}$" resp. "$\cdot_2$" denotes hexadecimal resp. binary encoding of the binary magic numbers.

In the following, let us show that the formula (5) is also unsatisfiable. 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

$$(X\ \&\ U_1)\ =\ \big(0, 0, X[5], X[4], 0, 0, X[1], X[0]\big)$$

and

$$\big((X \gg (1 \ll 1))\ \&\ U_1\big)\ =\ \big(0, 0, X[7], X[6], 0, 0, X[3], X[2]\big)$$

are forced to be equal, some bits of X should coincide, as follows:

$$X \;:=\; \big(x_5, x_4, x_5, x_4, x_1, x_0, x_1, x_0\big)$$

Furthermore, considering also the equation of

$$(X \;\&\; U_2) \;=\; \big(0, 0, 0, 0, X[3], X[2], X[1], X[0]\big)$$

and

$$\big((X \gg (1 \ll 2)) \;\&\; U_2\big) \;=\; \big(0, 0, 0, 0, X[7], X[6], X[5], X[4]\big)$$

results in

$$X \;:=\; \big(x_1, x_0, x_1, x_0, x_1, x_0, x_1, x_0\big)$$

In a similar fashion, the bits of Y are constrained as follows:

$$Y \;:=\; \big(y_6, y_6, y_4, y_4, y_2, y_2, y_0, y_0\big)$$

In order to show that the formula (5) is unsatisfiable, let us evaluate the "clauses" in the formula:

$$\begin{aligned}
(X \mid Y \mid\!\sim U_0 \mid\!\sim U_1) &= \big(\ 1\ ,\ 1\ ,\ 1\ ,\ x_0 \vee y_4\ ,\ 1\ ,\ 1\ ,\ 1\ ,\ x_0 \vee y_0\ \big)\\
(X \mid\!\sim Y \mid U_0 \mid\!\sim U_1 \mid\!\sim U_2) &= \big(\ 1\ ,\ 1\ ,\ 1\ ,\ 1\ ,\ 1\ ,\ 1\ ,\ x_1 \vee \neg y_0\ ,\ 1\ \big)\\
(X \mid\!\sim Y \mid\!\sim U_0 \mid\!\sim U_1 \mid U_2) &= \big(\ 1\ ,\ 1\ ,\ 1\ ,\ x_0 \vee \neg y_4\ ,\ 1\ ,\ 1\ ,\ 1\ ,\ 1\ \big)\\
(\sim X \mid Y \mid\!\sim U_0 \mid\!\sim U_2) &= \big(\ 1\ ,\ 1\ ,\ 1\ ,\ 1\ ,\ 1\ ,\ \neg x_0 \vee y_2\ ,\ 1\ ,\ \neg x_0 \vee y_0\ \big)\\
(\sim X \mid\!\sim Y \mid U_0 \mid U_1 \mid\!\sim U_2) &= \big(\ 1\ ,\ 1\ ,\ 1\ ,\ 1\ ,\ \neg x_1 \vee \neg y_2\ ,\ 1\ ,\ 1\ ,\ 1\ \big)
\end{aligned}$$

By applying *bitwise and* to them, we get the bit-vector represented by the formula (5):

$$\begin{pmatrix} 1\\ 1\\ 1\\ (x_0 \vee \neg y_4) \wedge (x_0 \vee y_4)\\ \neg x_1 \vee \neg y_2\\ \neg x_0 \vee y_2\\ x_1 \vee \neg y_0\\ (x_0 \vee y_0) \wedge (\neg x_0 \vee y_0) \end{pmatrix} = \begin{pmatrix} 1\\ 1\\ 1\\ x_0\\ \neg x_1 \vee \neg y_2\\ \neg x_0 \vee y_2\\ x_1 \vee \neg y_0\\ y_0 \end{pmatrix}$$

In order to check if every bits of this bit-vector can evaluate to 1, it is sufficient to try to satisfy the set of the above (propositional) clauses. It is easy to see that this clause set is unsatisfiable, since by unit propagation x_1 and y_2 must be 1, which contradicts with the clause $\neg x_1 \vee \neg y_2$.