On the Complexity of Symbolic Verification and Decision Problems in Bit-Vector Logic*

Gergely Kovásznai[1], Helmut Veith[1], Andreas Fröhlich[2], and Armin Biere[2]

[1] Vienna University of Technology, Wien, Austria
Formal Methods in Systems Engineering Group
[2] Johannes Kepler University, Linz, Austria
Institute for Formal Models and Verification

Abstract. We study the complexity of decision problems encoded in bit-vector logic. This class of problems includes word-level model checking, i.e., the reachability problem for transition systems encoded by bit-vector formulas. Our main result is a generic theorem which determines the complexity of a bit-vector encoded problem from the complexity of the problem in explicit encoding. In particular, NL-completeness of graph reachability directly implies PSpace-completeness and ExpSpace-completeness for word-level model checking with unary and binary arity encoding, respectively. In general, problems complete for a complexity class C are shown to be complete for an exponentially harder complexity class than C when represented by bit-vector formulas with unary encoded scalars, and further complete for a double exponentially harder complexity class than C with binary encoded scalars. We also show that multi-logarithmic succinct encodings of the scalars result in completeness for multi-exponentially harder complexity classes. Technically, our results are based on concepts from descriptive complexity theory and related techniques for OBDDs and Boolean encodings.

1 Introduction

Symbolic encodings of decision problems by Boolean formalisms are well-known to increase the problem complexity [1,2,3,4,5,6,7,8,9,10,11,12]. In particular, the literature has studied graph problems and other relational problems whose adjacency relation is given by a Boolean formula, circuit or BDD. As Tab. 1 shows, the complexity of these problems typically rises by an exponential, e.g., from NL to PSpace, from NP to NExpTime, etc. In this paper, we show that symbolic encodings by quantifier-free bit-vector logic (QF_BV) will in general also lead to a complexity increase which ranges from exponential to multi-exponential. Interestingly, the increase depends on a *single* factor, namely how the bit-width of bit-vectors is encoded. For unary encoding, bit-vector logic shows the same complexity behavior as Boolean logic, and for binary encoding, the complexity

* Supported by the NFN grant S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the grant ICT10-050 (PROSEED) of the Vienna Science and Technology Fund (WWTF).

increase is double exponential. We can generalize the latter encoding, and call it "ν-logarithmic": encode the bit-width $2^{2^{\cdots^{2^c}}}$ as c in binary form, where the degree of exponentiation is $\nu - 2$. We achieve a ν-exponential increase in this case. Importantly, hardness already holds for bit-vector logics with the simple operators $\wedge, \vee, \sim, =$, and the increment operator $+_1$. Membership holds for *all* bit-vector operators which allow log-space computable bit-blasting. Note that $\wedge, \vee, \sim, =, +_1$ defines a very weak logic: $\wedge, \vee, \sim, =$ are contained in all reasonable logics, and the increment operator $+_1$ can be defined from other operators easily [13]. Therefore, our results determine the complexity of decision problems for a large class of bit-vector logics.

Encoding $\rightarrow$ $\downarrow$ *Problem*	explicit	Boolean circ./formula, BDD	unary QF_BV	binary QF_BV	ν-logarithmic QF_BV
Word-Level MC, Reachability	NL	PSPACE	**PSPACE**	**EXPSPACE**	$(\nu-1)$-**EXPSPACE**
Circuit Value, Alternating Reachability	P	EXPTIME	**EXPTIME**	**2-EXPTIME**	ν-**EXPTIME**
Clique, 3-SAT, SAT, Knapsack	NP	NEXPTIME	**NEXPTIME**	**2-NEXPTIME**	ν-**NEXPTIME**
k-QBF	Σ_k^P	$\mathrm{NE}^{\Sigma_k^P}$	$\mathbf{NE}^{\Sigma_k^P}$	$\mathbf{2\text{-}NE}^{\mathbf{\Sigma_k^P}}$	ν-$\mathbf{NE}^{\Sigma_k^P}$

Table 1. Examples of complexity increase by symbolic encoding. New results are indicated in boldface. All membership results hold for logics whose operators allow log-space computable bit-blasting. Hardness requires the operators $\wedge, \vee, \sim, =, +_1$. The column with ν holds for all $\nu > 1$.

Bit-Vector Logic. The theory of *fixed-width bit-vector logics* (i.e., logics where each bit-vector has a given fixed bit-width) is investigated in several scientific works [14,15,16,17,18], and even concrete formats for specifying such bit-vector problems exist, e.g., the SMT-LIB format [19] or the BTOR format [20]. In this paper, we restrict ourselves to *quantifier-free bit-vector* (QF_BV [19]) logics.

As discussed below, bit-vector logics have attracted significant interest in computer-aided verification and SMT solvers. From a theory perspective, bit-vector logics are very succinct logics to express Boolean functions. In contrast to Boolean logic, BDDs, and QBF, they are based on variables for *bit-vectors* rather than variables for individual bits. Thus, for instance $x^{[32]} = y^{[32]}$ expresses that two bit-vectors x and y of bit-width 32 are equal. Bit-vector operators are therefore defined for arbitrary bit-width n, for instance bitwise and/or/xor, shift operators, etc. This has important consequences: (1) a bit-vector logic is given by a list of operators, (2) there is an infinite number of bit-vector logics, and (3) there is no finite functionally complete set of operators from which all other operators can be defined. Moreover, it is evident that the encoding of scalars

such as the number 32 in the above simple example is related to the complexity of bit-vector logic.

In previous work by some of the authors [21,13], we investigated the complexity of satisfiability checking of bit-vector formulas. For instance, we showed in [21] that satisfiability checking of QF_BV is NP-complete resp. NExpTime-complete if unary resp. binary encoding of scalars is used and any standard operator of the SMT-LIB [19] is allowed. (All these operators allow log-space computable bit-blasting.) In the binary case, we further analyzed what happened if we restricted the operator set; e.g., if only *bitwise operators*, *equality*, and *left shift by one* are allowed, then the complexity turns out to be PSpace-complete [13]. In fact, it is easy to see that also the logic of the operators $\wedge, \vee, \sim, =, +_1$ has a satisfiability problem in PSpace.

Word-Level Model Checking and Decision Problems. In hardware and software verification, bit-vector logics are a natural framework for word-level system descriptions; e.g., registers in digital circuits and variables in software can be represented by bit-vectors, and word-level operators, such as bitwise ones and arithmetic ones, can be applied to them. The main practical motivation for our work is *word-level model checking*, a bit-vector encoded problem that is of importance in practice. With word-level model checking, we refer to the problem of reachability in a transition system where a state is given by a valuation of one or more bit-vectors, and the transition relation over the states is expressed as a bit-vector formula. Such a representation provides a natural encoding for design information captured at a higher level than that of individual wires and primitive gates. In the past, there has been lots of research on bit-level model checking [22] as well as bit-vector formula decision procedures [23,24]. Comparatively few work has yet been published on word-level model checking. However, with increasing performance of state-of-the-art model checkers [25] and SMT solvers [26,27], also the interest in word-level model checking is growing [28,20,29]. While there are some practical approaches to attack word-level model checking [28,20,29], we are not aware of any work that is dealing with the complexity of the underlying decision problem. Row 1 of Tab. 1 shows that we determine the complexity of word-level model checking for a large class of operators and scalar encodings.

Beyond word-level model checking, we also address the complexity of other decision problems. Rows 2-4 of Tab. 1 give examples of the complexity results for well-known decision problems in bit-vector encoding.

Technical Contribution. Instead of individual complexity results, the paper presents a generic technique to *lift* known complexity results for explicit encodings to the case of bit-vector encodings. Similar techniques were previously developed for symbolic encodings by circuits [7,8,9], Boolean formulas [10], and OBDDs [30]. Lifting membership for a complexity class is the easier part, for which we give a general result in Thm. 1. Lifting hardness requires more effort. Similarly as in [10,30], our method assumes that the problems in explicit encoding are hard under *quantifier-free reductions*, a notion of reduction introduced in descriptive complexity theory [31]. Note that the problems in Tab. 1 fulfill this

requirement. The key theorem is Thm. 2, from which a general hardness result is implied in Corr. 2.

Discussion. The results of this paper show that the complexity of bit-vector encoded problems depends crucially on the formalism to represent the bit-width of the bit-vectors. At first sight, these results may seem unexpected, e.g., a small part of the formalism clearly dominates the complexity. From an algorithmic perspective, however, this is not surprising: executing a for-loop from `0` to `INT_MAX` on architectures with bit-width 16, 2^{16} or $2^{2^{16}}$ will result in drastically different runtimes!

It may also be surprising that QF_BV fragments with PSPACE satisfiability and fragments with NEXPTIME satisfiability have the same complexity, e.g., for word-level model checking. This is however a common phenomenon: Boolean logic has an NP satisfiability problem, while satisfiability of BDDs is constant time. Nevertheless, the model checking problem for both of them is PSPACE-complete [10,30].

Using unary and binary encodings for scalars draws a connection to previous work [21]. Intuitively, results for the unary case measure complexity in terms of bit-widths, and those for the binary case measure complexity in the classical sense, i.e., in terms of formula size. The ν-logarithmic encoding also manifests itself in practice, such as the one in the SMB-LIB to declare arrays by writing `(Array idx elem)`, where *idx* is the sort for array indexes, and *elem* is the sort for array elements. If *idx* is a bit-vector sort `(_ BitVec n)`, where n is encoded w.l.o.g. in binary form, the size of the array is double exponential in the length of the binary encoding of n.

We finally note that hardness for the unary case can also be concluded from an analysis of the proofs in [10] using the definitions of symbolic encodings in [30]. The current paper gives a direct proof for the unary case which is independent of the predecessor papers.

2 Preliminaries

Let $\mathbb{N}$ be the set of natural numbers $\{0, 1, 2, \dots\}$, while $\mathbb{N}^+$ denotes $\mathbb{N}\backslash\{0\}$. $\mathbb{B} = \{0, 1\}$ is the Boolean domain. Given $i \in \mathbb{N}$, let us define the repeated exponentiation function $exp_i : \mathbb{N} \mapsto \mathbb{N}$ as follows: $exp_0(n) = n$ and $exp_{i+1}(n) = 2^{exp_i(n)}$. Given a logical formula ϕ (in either bit-vector, first-order, or Boolean logic), if $x_1, \dots, x_k$ are all the free variables that occur in ϕ, we indicate this by writing $\phi(x_1, \dots, x_k)$.

Complexity Classes. We assume that the reader is familiar with standard complexity classes such as NL, P, EXPTIME, etc., as listed in Tab 1. For simplicity, we will refer to these complexity classes as "standard complexity classes". For a standard complexity class, it is natural to define the exponentially harder complexity class: $\text{EXP}_1(\text{L}) = \text{EXP}_1(\text{NL}) = \text{PSPACE}$, $\text{EXP}_2(\text{NL}) = \text{EXP}_1(\text{PSPACE}) = \text{EXPSPACE}$, etc. Similarly, $\text{EXP}_1(\text{P}) = \text{EXPTIME}$, $\text{EXP}_2(\text{P}) = \text{EXP}_1(\text{EXPTIME}) =$

2-ExpTime, etc., and analogously for other standard complexity classes. For a formal definition of this concept (which is beyond the scope and goal of this paper) one can use the concept of leaf languages [9,2].

Computational Problems in Descriptive Complexity Theory. A *relational signature* is a tuple $\tau = (P_1^{a_1}, \ldots, P_k^{a_k})$ of relation symbols of arity $a_1, \ldots, a_k$, respectively. A finite *structure* over τ is a tuple $\mathcal{A} = (U, \widehat{P}_1^{a_1}, \ldots, \widehat{P}_k^{a_k})$ where U is a nonempty finite set (called the universe of $\mathcal{A}$) and each $\widehat{P}_i^{a_i} \subseteq U^{a_i}$ is a relation over U. The class of all finite structures over τ is denoted by $\mathit{Struct}(\tau)$. A *computational problem* over τ is a class $A \subseteq \mathit{Struct}(\tau)$, such that A is closed under isomorphism. In this paper, we assume *convex* problems, as introduced in [30], and similarly in [32]. A problem is convex if adding isolated elements to the universe of a structure does not change membership in the problem. In Sec. 4 we will show that the model checking problem is naturally presented in this framework. For background on descriptive complexity see [33].

3 Bit-Vector Logic

A *bit-vector*, or word, is a sequence of bits (i.e. 0 or 1). In this paper, we consider bit-vectors of a fixed size $n \in \mathbb{N}^+$, where n is called the *bit-width* of the bit-vector. We assume the usual syntax and semantics for *quantifier-free bit-vector logic* (QF_BV), cf. the SMT-LIB format [19] and the literature [14,15,16,17,18]. Basically, a bit-vector formula contains bit-vector variables and bit-vector constants, each of which is of a certain bit-width specified next to the variable resp. constant, and uses certain bit-vector operators whose semantics is a priori defined. For example, $x^{[16]} \neq y^{[16]} \ \wedge \ \left(u^{[32]} + v^{[32]} \ = \ (x^{[16]} \circ y^{[16]}) \ll 1^{[32]}\right)$ is a bit-vector formula with variables x and y of bit-width 16, u and v of bit-width 32, and operators for addition, shifting, concatenation, and comparison.

Note that, in bit-vector formulas, there exist such components which themselves do *not* represent bit-vectors, but rather carry additional *numerical* information to the bit-vectors. We call them *scalars*. Bit-width is a scalar, and there might be also other types of scalars in a formula[3]. This paper demonstrates the effect of encoding the scalars in different ways. For instance, scalars could be encoded as unary numbers or w.l.o.g. binary numbers, or we could choose even more succinct encodings, such as the binary encoding of the logarithm of the scalar. Formally, we represent those encodings by an integer $\nu \in \mathbb{N}^+$, i.e., ν denotes how $n \in \mathbb{N}$ is obtained from a scalar s: (1) if $\nu = 1$, then s is a *unary* number encoding of n; (2) if $\nu > 1$, then s is a *binary* number encoding of a number $d \in \mathbb{N}$ such that $n = \mathit{exp}_{\nu-2}(d)$. Let $\mathit{encode}_\nu(n)$ denote the scalar that ν-encodes the number n, and let $\mathit{decode}_\nu(s)$ denote the number that is ν-encoded by the appropriate scalar s.

Now we give a formal definition of bit-vector formulas with the operators we use throughout in the rest of the paper. Let us suppose that an encoding ν is

[3] For example, the common operators *extraction* and *zero/sign extensions* use scalar arguments as well, cf. [19,14,15,16,17,18].

fixed. A *bit-vector term* t of bit-width n is denoted by $t^{[s]}$ where $s = encode_\nu(n)$, and defined inductively as follows:

	term	condition	bit-width
constant:	$c^{[s]}$	$c \in \mathbb{N}$, $0 \leq c < 2^n$	n
variable:	$x^{[s]}$	x is an identifier	n
bitwise negation:	$\sim t^{[s]}$	$t^{[s]}$ is a term	n
bitwise and/or/xor, addition: $\bullet \in \{\&, \mid, \oplus, +\}$	$({t_1}^{[s]} \bullet {t_2}^{[s]})$	${t_1}^{[s]}$, ${t_2}^{[s]}$ are terms	n
equality, unsigned less than: $\bullet \in \{=, <_{\mathbf{u}}\}$	$({t_1}^{[s]} \bullet {t_2}^{[s]})$	${t_1}^{[s]}$, ${t_2}^{[s]}$ are terms	1

Note that the value c of a bit-vector constant is *not* a scalar, therefore it is always encoded as a binary number, regardless of ν. By a *bit-vector formula* we mean a term of bit-width 1, since this case can be considered as the Boolean case. For better readability, we write $\neg, \wedge, \vee$ instead of $\sim, \&, \mid$ for bit-width 1, respectively. Given a bit-vector operator set Ω, let $\mathcal{BV}_\nu^\Omega$ denote the fragment of QF_BV that applies the encoding ν to scalars and only uses operators from Ω. *Bit-blasting*, or flattening [34], interprets bit-vector variables as strings of Boolean variables and translates bit-vector operations into Boolean formulas. By denoting the Boolean logic as $\mathcal{BO}$, we give a formal definition.

Definition 1 (Bit-blasting). *Given an operator set Ω, a bit-blasting function $bblast_\nu^\Omega : \mathcal{BV}_\nu^\Omega \mapsto \mathcal{BO}$ is defined as follows:*

$$bblast_\nu^\Omega\big(\psi({x_1}^{[s_1]}, \ldots, {x_k}^{[s_k]})\big) = \phi(y_1^1, \ldots, y_1^{n_1}, \ldots, y_k^1, \ldots, y_k^{n_k}, z_1, \ldots, z_l)$$

where $n_i = decode_\nu(s_i)$, such that $\forall d_1 \in \mathbb{B}^{n_1}, \ldots, d_k \in \mathbb{B}^{n_k}$

$$\psi(d_1, \ldots, d_k) = true \text{ iff}$$
$$\exists! e_1, \ldots, e_l \in \mathbb{B} \,.\, \phi(d_1^1, \ldots, d_1^{n_1}, \ldots, d_k^1, \ldots, d_k^{n_k}, e_1, \ldots, e_l) = true$$

where d_i^j denotes the jth bit of d_i.

Note that the additional Boolean values $e_1, \ldots, e_l$ are uniquely existentially quantified. Therefore, in fact, each Boolean variable z_i can rather be considered as a bit-vector function $f_i({x_1}^{[s_1]}, \ldots, {x_k}^{[s_k]}) : \mathbb{B}^{n_1} \times \cdots \times \mathbb{B}^{n_k} \mapsto \mathbb{B}$. Thus, ψ and ϕ encode the same $\big(\sum_{i=1}^k n_i\big)$-ary relation over $\mathbb{B}$.

We say that $bblast_\nu^\Omega$ is *log-space computable in bit-width* if it is log-space computable in $\sum_{i=1}^k n_i$. Let Π denote the set of all the bit-vector operators such that $bblast_\nu^\Pi$ is log-space computable in bit-width, for all $\nu \in \mathbb{N}^+$. Note that all the common bit-vector operators [19] fall into Π.

4 Motivating Example: Word-Level Model Checking

We now demonstrate that our generic main results can be applied to the important example of reachability analysis in model checking, as to establish the complexity of reachability in word-level model checking.

The model checking problem has a natural representation with a relational signature $\tau = (I^1, T^2, P^1)$. In model checking terminology, I represents the set of initial states, T the transition relation, and P the condition to check, i.e., the set of states whose reachability we want to verify. Thus, a structure $\mathcal{A} = (U, \widehat{I}^1, \widehat{T}^2, \widehat{P}^1)$ is essentially a Kripke structure. *Reachability analysis* in $\mathcal{A}$ means to check if there exists a reachable $\widehat{P}$-state in the defined transition system, i.e., if $\exists s_0, s_1, \ldots, s_k \in U$ such that (1) $s_0 \in \widehat{I}$, (2) $\forall i \in [1, k]$. $(s_{i-1}, s_i) \in \widehat{T}$, and (3) $s_k \in \widehat{P}$. We call $MC = \{\mathcal{A} \in \mathit{Struct}(\tau) \mid \exists \text{ a reachable } \widehat{P}\text{-state in } \mathcal{A}\}$ the (explicit) *model checking* problem. Since MC is a simple variant of graph reachability, we know from [31] that MC is NL-complete under quantifier-free reductions.

The *word-level* encoding of MC means to encode the states by tuples of bit-vectors, and to define the relations $\widehat{I}, \widehat{T}, \widehat{P}$ by bit-vector formulas. The corresponding decision problem is called $bv_\nu^\Omega(MC)$, where ν specifies the scalar encoding and Ω is a set of bit-vector operators that are allowed in the formulas. We will formally define this problem in Sec. 5.

Our results require the following assumptions on Ω: (1) Ω contains only such operators for which bit-blasting is log-space computable in bit-width and (2) Ω contains all the simple operators $\wedge, \vee, \sim, =, +_1$. In particular, Ω may contain all common bit-vector operators [19] that are used in practice.

Then we obtain the following results as a direct consequence of Thm. 1, Cor. 2, and the NL-completeness of MC:

Corollary 1. *Let $\Omega \subseteq \Pi$. The decision problem $bv_\nu^\Omega(MC)$ is*

1. PSPACE*-complete, if $\nu = 1$ and $\Omega \supseteq \{\wedge, \vee, \neg\}$,*
2. *$(\nu - 1)$-*EXPSPACE*-complete, if $\nu > 1$ and $\Omega \supseteq \{\wedge, \vee, \sim, =, +_1\}$,*

under log-space reductions.

In practice, the term *word-level model checking* usually refers to the problem $bv_2^\Omega(MC)$, i.e., all scalars in the formulas are encoded as w.l.o.g. binary numbers. Thus, our results show that word-level model checking is EXPSPACE-complete.

5 Bit-Vector Representation of Problems

Our intention is to represent instances of computational problems as bit-vector formulas. More precisely, given a relational signature $\tau = (P_1^{a_1}, \ldots, P_k^{a_k})$, we define what the bit-vector definition of a corresponding relation $\widehat{P}_i^{\,a_i}$ looks like and what structure these definitions generate.

In order to simplify the presentation, we introduce the concept of *term vectors*. A term vector is a sequence ${t_1}^{[s_1]}, \ldots, {t_l}^{[s_l]}$ of bit-vector terms. We write term vectors in boldface, i.e., $\mathbf{t} = {t_1}^{[s_1]}, \ldots, {t_l}^{[s_l]}$, and say that $\mathbf{t}$ has the bit-width signature $s_1, \ldots, s_l$. We distinguish the special case when terms are variables, by denoting *variable vectors* as $\mathbf{x}, \mathbf{y}, \mathbf{z}$.

Word-level model checking can again serve as motivation here, since it represents the states of a transition system by the same set of bit-vector variables

$x_1{}^{[s_1]}, \ldots, x_l{}^{[s_l]}$. I.e., a state is in fact can be represented as the valuation of terms $t_1{}^{[s_1]}, \ldots, t_l{}^{[s_l]}$ assigned to those variables. Therefore, it is important that each state must have the same bit-width signature $s_1, \ldots, s_l$.

Definition 2. *Let $\mathbf{x}_1, \ldots, \mathbf{x}_a$ be variable vectors each of which has the bit-width signature $s_1, \ldots, s_l$. Let ν be a scalar encoding, and let $n_i = \mathit{decode}_\nu(s_i)$ denote the actual bit-widths. A bit-vector formula $\psi(\mathbf{x}_1, \ldots, \mathbf{x}_a)$ defines the a-ary relation*

$$\mathit{gen}^a_\nu(\psi) = \{(d_1, \ldots, d_a) \in (\mathbb{B}^{n_1} \times \cdots \times \mathbb{B}^{n_l})^a \mid \psi(d_1, \ldots, d_a) = \mathit{true}\}.$$

Let $\tau = (P_1^{a_1}, \ldots, P_k^{a_k})$ be a relational signature. The tuple of definitions

$$\begin{aligned}\Psi = \Big(& P_1(\mathbf{x}^1_1, \ldots, \mathbf{x}^1_{a_1}) := \psi_1(\mathbf{x}^1_1, \ldots, \mathbf{x}^1_{a_1}),\\ & \ldots,\\ & P_k(\mathbf{x}^k_1, \ldots, \mathbf{x}^k_{a_k}) := \psi_k(\mathbf{x}^k_1, \ldots, \mathbf{x}^k_{a_k})\Big)\end{aligned}$$

where each ψ_i is a bit-vector formula and each $\mathbf{x}^i_j$ is a variable vector that has the bit-width signature $s_1, \ldots, s_l$, defines the τ-structure

$$\mathit{gen}^\tau_\nu(\Psi) = \big(\mathbb{B}^{n_1} \times \cdots \times \mathbb{B}^{n_l}, \mathit{gen}^{a_1}_\nu(\psi_1), \ldots, \mathit{gen}^{a_k}_\nu(\psi_k)\big).$$

The bit-vector representation of a computational problem consists of all the bit-vector representations of all the structures in the problem. Besides the definitions Ψ of relations, it is also necessary to include the scalar encoding ν to use, as follows.

Definition 3. *Let $A \subseteq \mathit{Struct}(\tau)$ be a problem, ν a scalar encoding, and Ω a set of bit-vector operators. Then we define*

$$\mathit{bv}^\Omega_\nu(A) = \big\{(\Psi, \nu) \;\big|\; \mathit{gen}^\tau_\nu(\Psi) \in A, \text{ and } \Psi \text{ contains only } \mathcal{BV}^\Omega_\nu \text{ formulas}\big\}.$$

In order to show how *membership* for a standard complexity class C can be automatically lifted when bit-vector representation is used, we give a necessary, although not very strong, criterion on the operator set. This criterion is based on bit-blasting, and requires to use operators from Π, i.e., those which allow log-space computable bit-blasting in bit-width.

Theorem 1. *Given a problem A, a standard complexity class C, and an operator set $\Omega \subseteq \Pi$, if $A \in C$, then $\mathit{bv}^\Omega_\nu(A) \in \textsc{Exp}_\nu(C)$.*

6 Lifting Hardness

The main contribution of this paper is to show how hardness for a standard complexity class C can also be automatically lifted. Our most important theorem, Thm. 2 gives a rather general hardness result, from which we derive Cor. 2 to show hardness of bv^Ω_ν for $\textsc{Exp}_\nu(C)$, where $\Omega \supseteq \{\wedge, \vee, \sim, =, +_1\}$.

Our proofs employ the framework of *descriptive complexity theory* [31]. In particular, we use the standard assumption that all structures are equipped with a binary successor relation. Thus, the universe of a structure can be naturally seen as an initial segment of the natural numbers. Our complexity results for bit-vector encoded problems assume that the problems in explicit encoding are hard under *quantifier-free reductions*, i.e., quantifier-free interpretations with equality and the successor relation. Examples of such problems including those in Tab. 1 can be found in [35,36,37,38,39]. For natural problems, it is usually not difficult to rephrase an existing reduction as a quantifier-free reduction. Let $A \leqslant_{\mathrm{qf}} B$ resp. $A \leqslant_{\mathrm{L}} B$ denote that the problem A has a *quantifier-free* resp. *log-space* reduction to the problem B. Note that quantifier-free reductions are weaker than log-space reductions, i.e., $A \leqslant_{\mathrm{qf}} B$ implies $A \leqslant_{\mathrm{L}} B$. For exact background material and definitions, see [31].

The key steps for Thm. 2 are two lemmas. Lemma 1 ("Conversion Lemma") shows that a quantifier-free reduction between A and B can be lifted to a log-space reduction between $bv_\nu(A)$ and $bv_\nu(B)$. Lemma 2 shows that A is log-space reducible to $bv_\nu(long_\nu(A))$ where $long_\nu(\cdot)$ is an operator which *decreases* the complexity ν-exponentially. From these two lemmas, Thm. 2 follows easily. The methodology of this paper is closest to [30], which contains a more thorough discussion of related work, descriptive complexity, and complexity theoretic background.

Lemma 1 (Conversion Lemma). *Let $\Omega \supseteq \{\wedge, \vee, \sim, =, +_1\}$. Given two problems $A \subseteq Struct(\sigma)$ and $B \subseteq Struct(\tau)$, if $A \leqslant_{\mathrm{qf}} B$, then $bv_\nu^\Omega(A) \leqslant_{\mathrm{L}} bv_\nu^\Omega(B)$, for any ν.*

The role of the following definition is to obtain from a problem A another problem $long_\nu(A)$ of ν-exponentially lower complexity. In order to construct this latter problem, we are going to "blow up" the size of a structure in a potentially ν-exponential way. To this end, we view a structure $\mathcal{A}$ as a bit string, and interpret the bit string as a binary number $char(\mathcal{A})$. The bit string is obtained from the characteristic sequences of the relations in $\mathcal{A}$, i.e., for each tuple in lexicographic order, a single bit indicates whether the tuple is in the relation. Due to the presence of the successor relation, this notion is well defined.

Definition 4. *Given a structure $\mathcal{A} = (U, \widehat{P}_1, \ldots, \widehat{P}_k)$, let $\mathrm{char}(\widehat{P}_i)$ denote the characteristic sequence of the tuples in $\widehat{P}_i$ in lexicographical order. Let $\mathrm{char}(\mathcal{A})$ denote the binary number obtained by concatenating a leading 1 with the concatenation of $\mathrm{char}(\widehat{P}_1), \ldots, \mathrm{char}(\widehat{P}_k)$.*

We define $long_\nu(\mathcal{A}) = \big\{(V, \widehat{R}^1) \;\big|\; |V| = exp_{\nu-1}(char(\mathcal{A}))$ and $\big|\widehat{R}\big| = |V|\big\}$. For a problem A, let $long_\nu(A) = \bigcup_{\mathcal{A}\in A} long_\nu(\mathcal{A})$. For a complexity class C, let $long_\nu(C) = \bigcup_{A\in C} long_\nu(A)$.

The next lemma shows that encoding the problem $long_\nu(A)$ as bit-vector formulas applying ν-encoding to scalars gives a ν-exponentially more succinct representation, to which, consequently, the original problem A can be reduced.

Lemma 2. *Given a problem A, $A \leqslant_{\mathrm{L}} bv_\nu^\Omega(long_\nu(A))$ if one of the following conditions holds:*

1. $\nu = 1$ *and* $\Omega \supseteq \{<_{\mathbf{u}}\}$
2. $\nu > 1$ *and* $\Omega \supseteq \{=\}$

Theorem 2 (Upgrading Theorem). *Let C_1 and C_2 be complexity classes such that $long_\nu(C_1) \subseteq C_2$. If a problem A is C_2-hard under quantifier-free reductions, then $bv_\nu^\Omega(A)$ is C_1-hard under log-space reductions if one of the following conditions holds:*

1. $\nu = 1$ *and* $\Omega \supseteq \{\wedge, \vee, \sim, =, +_1, <_{\mathbf{u}}\}$
2. $\nu > 1$ *and* $\Omega \supseteq \{\wedge, \vee, \sim, =, +_1\}$

Proof. For any $B \in C_1$, by assumption $long_\nu(B) \in C_2$, and hence $long_\nu(B) \leqslant_{\mathrm{qf}} A$. By Lemma 1, it follows that $bv_\nu^\Omega(long_\nu(B)) \leqslant_{\mathrm{L}} bv_\nu^\Omega(A)$, regardless of the additional operator $<_{\mathbf{u}}$ in the unary case. Furthermore, by Lemma 2, it holds that $B \leqslant_{\mathrm{L}} bv_\nu^\Omega(long_\nu(B))$. To put them together, $B \leqslant_{\mathrm{L}} bv_\nu^\Omega(long_\nu(B)) \leqslant_{\mathrm{L}} bv_\nu^\Omega(A)$ and, therefore, $bv_\nu^\Omega(A)$ is C_1-hard.

As we discussed before, the case of $\nu = 1$ shows the same complexity behavior as Boolean logic. Of course, this is no wonder, since all the operators in $\Omega = \{\wedge, \vee, \sim, =, +_1, <_{\mathbf{u}}\}$, or more precisely, $\mathcal{BV}_1^\Omega$ allows *log-space computable* bit-blasting in bit-width, and also *in formula size*, since bit-widths are now encoded in unary form. Thus, $\mathcal{BV}_1^\Omega$ is log-space reducible to $\mathcal{BV}_1^{\{\wedge,\vee,\neg\}}$, since $\{\wedge, \vee, \neg\}$ is a functionally complete set of Boolean operators. As a consequence, one can strengthen the first statement of Thm. 2 further as follows: $bv_1^{\Omega'}(A)$ is C_1-hard for any $\Omega' \supseteq \{\wedge, \vee, \neg\}$. Note that this is consistent with corresponding results in [10,30]. As a direct consequence, we can give the following corollary.

Corollary 2. *Given a standard complexity class C and a problem A, if A is C-hard under quantifier-free reductions, then $bv_\nu^\Omega(A)$ is $\mathrm{EXP}_\nu(C)$-hard under log-space reductions if one of the following conditions holds:*

1. $\nu = 1$ *and* $\Omega \supseteq \{\wedge, \vee, \neg\}$
2. $\nu > 1$ *and* $\Omega \supseteq \{\wedge, \vee, \sim, =, +_1\}$

7 Conclusion

This paper gives a generic method for asserting the complexity of bit-vector logic encoded problems. As corollary we obtain a new complexity result for word-level model checking, an important practical problem. Since all complexity classes with complete problems have problems complete under quantifier-free reductions [11], we obtain a comprehensive picture of the worst case complexity of problems in bit-vector encoding. Note that our results do not apply to *satisfiability* of bit-vector logic, because "existence of a solution" is not hard for a complexity class, and thus the assumption of the Conversion Lemma is not satisfied. Nevertheless, we expect that the complexity of satisfiability for multi-logarithmic encodings shows a similar behavior as the problems studied here. We leave an analysis of this question to future work.

References

1. Balcázar, J.L., Lozano, A., Torán, J.: The complexity of algorithmic problems on succinct instances. Computer Science (1992) 351–377
2. Borchert, B., Lozano, A.: Succinct circuit representations and leaf language classes are basically the same concept. Inf. Process. Lett. **59**(4) (1996) 211–215
3. Das, B., Scharpfenecker, P., Torán, J.: Succinct encodings of graph isomorphism. In: LATA, Springer (2014) to appear.
4. Feigenbaum, J., Kannan, S., Vardi, M.Y., Viswanathan, M.: Complexity of problems on graphs represented as OBDDs. Chicago Journal of Theoretical Computer Science **5**(5) (1999)
5. Galperin, H., Wigderson, A.: Succinct representations of graphs. Information and Control **56**(3) (1983) 183–198
6. Gottlob, G., Leone, N., Veith, H.: Succinctness as a source of complexity in logical formalisms. Annals of Pure and Applied Logic **97**(1) (1999) 231–260
7. Lozano, A., Balcázar, J.L.: The complexity of graph problems for succinctly represented graphs. In: Graph-Theoretic Concepts in Computer Science, Springer (1990) 277–286
8. Papadimitriou, C.H., Yannakakis, M.: A note on succinct representations of graphs. Information and Control **71**(3) (1986) 181–185
9. Veith, H.: Succinct representation, leaf languages, and projection reductions. In: IEEE Conference on Computational Complexity. (1996) 118–126
10. Veith, H.: Languages represented by boolean formulas. Inf. Process. Lett. **63**(5) (1997) 251–256
11. Veith, H.: Succinct representation, leaf languages, and projection reductions. Information and Computation **142**(2) (1998) 207–236
12. Wagner, K.W.: The complexity of combinatorial problems with succinct input representation. Acta Informatica **23**(3) (1986) 325–356
13. Fröhlich, A., Kovásznai, G., Biere, A.: More on the complexity of quantifier-free fixed-size bit-vector logics with binary encoding. In: CSR'13. Volume 7913 of LNCS., Springer (2013) 378–390
14. Barrett, C.W., Dill, D.L., Levitt, J.R.: A decision procedure for bit-vector arithmetic. In: Proc. DAC'98. (1998) 522–527
15. Bjørner, N., Pichora, M.C.: Deciding fixed and non-fixed size bit-vectors. In: TACAS. Volume 1384 of LNCS., Springer (1998) 376–392
16. Bruttomesso, R., Sharygina, N.: A scalable decision procedure for fixed-width bit-vectors. In: ICCAD, IEEE (2009) 13–20
17. Cyrluk, D., Mller, O., Rueß, H.: An efficient decision procedure for a theory of fixed-sized bitvectors with composition and extraction. In: CAV'97. Volume 1254 of LNCS., Springer (1997) 60–71
18. Franzén, A.: Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT. PhD thesis, University of Trento (2010)
19. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard: Version 2.0. In: Proc. SMT'10. (2010)
20. Brummayer, R., Biere, A., Lonsing, F.: BTOR: bit-precise modelling of word-level problems for model checking. In: Proc. 1st International Workshop on Bit-Precise Reasoning, New York, NY, USA, ACM (2008) 33–38
21. 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

22. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA, USA (1999)
23. Bryant, R.E., Lahiri, S.K., Seshia, S.A.: Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In: CAV'02. Volume 2404 of LNCS., Springer (2002) 78–92
24. Manolios, P., Srinivasan, S.K., Vroon, D.: Bat: The bit-level analysis tool. In Damm, W., Hermanns, H., eds.: CAV'07. Volume 4590 of LNCS., Springer (2007) 303–306
25. Bradley, A.R.: Understanding ic3. In: SAT'12. Volume 7317 of LNCS., Springer (2012) 1–14
26. Brummayer, R., Biere, A.: Boolector: An efficient SMT solver for bit-vectors and arrays. In: TACAS'09. Volume 5505 of LNCS., Springer (2009) 174–177
27. De Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Proc. TACAS'08, Springer-Verlag (2008) 337–340
28. Bjesse, P.: A practical approach to word level model checking of industrial netlists. In: CAV'08. Volume 5123 of LNCS., Springer (2008) 446–458
29. Bjorner, N., McMillan, K., Rybalchenko, A.: Program verification as satisfiability modulo theories. In: Proc. SMT'12. (2013) 3–11
30. Veith, H.: How to encode a logical structure by an OBDD. In: Proc. 13th Annual IEEE Conference on Computational Complexity, IEEE (1998) 122–131
31. Immerman, N.: Languages that capture complexity classes. SIAM Journal on Computing **16**(4) (1987) 760–778
32. Schwentick, T.: Padding and the expressive power of existential second-order logics. In: Computer Science Logic, Springer (1998) 461–477
33. Immerman, N.: Descriptive complexity. Springer (1999)
34. Kroening, D., Strichman, O.: Decision Procedures: An Algorithmic Point of View. Texts in Theoretical Computer Science. Springer (2008)
35. Stewart, I.A.: Complete problems involving boolean labelled structures and projection transactions. Journal of Logic and Computation **1**(6) (1991) 861–882
36. Stewart, I.A.: On completeness for NP via projection translations. In: Computer Science Logic, Springer (1992) 353–366
37. Stewart, I.A.: Using the Hamiltonian path operator to capture NP. Journal of Computer and System Sciences **45**(1) (1992) 127–151
38. Stewart, I.A.: On completeness for NP via projection translations. Mathematical systems theory **27**(2) (1994) 125–157
39. Stewart, I.A.: Complete problems for monotone NP. Theoretical computer science **145**(1) (1995) 147–157