# Speeding Up Quantified Bit-Vector SMT Solvers by Bit-Width Reductions and Extensions

Martin Jonáš[1] and Jan Strejček[2][0000−0001−5873−403X]⋆

[1] Fondazione Bruno Kessler, Trento, Italy
mjonas@fbk.eu
[2] Masaryk University, Brno, Czech Republic
strejcek@fi.muni.cz

**Abstract.** Recent experiments have shown that satisfiability of a quantified bit-vector formula coming from practical applications almost never changes after reducing all bit-widths in the formula to a small number of bits. This paper proposes a novel technique based on this observation. Roughly speaking, a given quantified bit-vector formula is reduced and sent to a solver, an obtained model is then extended to the original bit-widths and verified against the original formula. We also present an experimental evaluation demonstrating that this technique can significantly improve the performance of state-of-the-art SMT solvers Boolector, CVC4, and Q3B on quantified bit-vector formulas from the SMT-LIB repository.

## 1 Introduction

We have recently studied the influence of bit-width changes on satisfiability of quantified bit-vector formulas from the SMT-LIB repository. The experiments showed that satisfiability is surprisingly stable under these changes [7]. For example, more than 95% of the considered formulas keep the same satisfiability status after changing the bit-widths of all variables and constants to an arbitrary value between 1 and the original bit-width. Indeed, all these stable formulas have the same satisfiability status even if we replace every bit-vector constant and variable by its least-significant bit and thus transform the formula into a quantified Boolean formula. Moreover, the percentage of stable formulas increased well over 99% if all bit-widths are reduced to any value between 4 and the original bit-width.

The experiments also confirm natural expectation that a formula with smaller bit-widths can be often solved considerably faster than the formula with the same structure but with larger bit-widths. For example, solving the formula

$$\varphi \quad \equiv \quad \forall x \forall y \exists z \ (x \cdot (y + z) = 0)$$

takes Boolector [11] several minutes on a standard desktop machine when the variables $x, y, z$ and the constant 0 have the bit-width 32, but it is solved instantly when the bit-width is 2.

⋆ J. Strejček has been supported by the Czech Science Foundation grant GA18-02177S.

This paper presents a new technique for deciding satisfiability of quantified bit-vector formulas that builds on the two mentioned observations: satisfiability status of these formulas is usually stable under bit-width reduction and formulas with reduced bit-widths can be often solved faster. Intuitively, the technique consists of the following steps:

1. Reduce bit-widths of variables and constants in the input formula to a smaller bit-width.
2. Decide satisfiability of the reduced formula using a standard decision procedure. If the reduced formula is satisfiable, obtain its model. If it is unsatisfiable, obtain its countermodel (i.e., a reason for unsatisfiability).
3. Extend the model or the countermodel to the original bit-widths. For example, a model of the formula $\varphi$ defined above reduced to bit-widths 2 has the form $z^{[2]} = -y^{[2]}$, where the superscripts denote bit-widths of the corresponding variables. After extension to the original bit-width, we get $z^{[32]} = -y^{[32]}$.
4. Check whether the extended (counter)model is also a (counter)model of the original formula. If the extended model is a model of the original formula, then the formula is satisfiable. If the extended countermodel is a countermodel of the original formula, then the formula unsatisfiable. In the remaining cases, we increase the bit-widths in the reduced formula and repeat the process.

The technique has some similarities with the approximation framework of Zeljić et al. [12], which also reduces the precision of a given formula, computes a model of the reduced formula, and checks if it is a model of the original formula. However, the framework considers only quantifier-free formulas (and hence models are just elements of the considered domains, while they are interpretations of Skolem functions in our setting) and it does not work with countermodels (it processes unsat cores of reduced formulas instead).

The detailed description of formula reduction and (counter)model extension is given in Section 3, preceded by Section 2 that recalls all necessary background and notation. The algorithm is precisely formulated in Section 4. Section 5 presents our proof-of-concept implementation and it discusses many practical aspects: how to get a counterexample, what to do with an incomplete model etc. Experimental evaluation of the technique can be found in Section 6. It clearly shows that the presented technique can improve performance of considered state-of-the-art solvers for quantified bit-vector formulas, namely Boolector [11], CVC4 [1], and Q3B [9], on both satisfiable and unsatisfiable formulas from various subcategories of the relevant SMT-LIB category BV.

## 2 Preliminaries

This section briefly recalls the used logical notions and the *theory of fixed sized bit-vectors* (*BV* or *bit-vector theory* for short). In the description, we assume familiarity with standard definitions of a many-sorted logic, well-sorted terms, atomic formulas, and formulas. To simplify the presentation, we suppose that all formulas are in the *negation normal form*. That is, all formulas use only logical connectives disjunction, conjunction, and negation and the arguments of all negations are atomic formulas. We suppose that sets of all free and all bound variables in the formula are disjoint and that each variable is quantified at most once.

The bit-vector theory is a many-sorted first-order theory with infinitely many sorts denoted $[n]$, where $n$ is a positive integer. Each sort $[n]$ is interpreted as the set of all bit-vectors of length $n$, which is also called their *bit-width*. We denote the set of all bit-vectors of bit-width $n$ as $\mathcal{BV}_n$ and variables of sort $[n]$ as $x^{[n]}$, $y^{[n]}$, etc. The BV theory uses only three predicate symbols, namely *equality* ($=$), *unsigned inequality* of binary-encoded natural numbers ($\leq_u$), and *signed inequality* of integers in two's complement representation ($\leq_s$). The theory also contains various interpreted function symbols. Many of them represent binary operations that produce a bit-vector of the same bit-width as its two arguments. This is the case of *addition* ($+$), *multiplication* ($\cdot$), bit-wise *and* ($\&$), bit-wise *or* ($|$), bit-wise *exclusive or* ($\oplus$), *left-shift* ($\ll$), and *right-shift* ($\gg$). The theory further contains function symbols for *two's complement negation* ($-$), *concatenation* ($\mathsf{concat}$), *zero extension* extending the argument with $n$ most-significant zero bits ($\mathsf{zeroExt}_n$), *sign extension* extending the argument with $n$ copies of the sign bit ($\mathsf{signExt}_n$), and *extraction* of bits delimited by positions $u$ and $l$ (including bits at these positions) from the argument, where position 0 refers to the least-significant bit and $u \geq l$ ($\mathsf{extract}_l^u$). The signature of BV theory also contains numerals for constants $m^{[n]}$ for each bit-width $n > 0$ and each number $0 \leq m \leq 2^n - 1$. Each term $t$ has an associated bit-width, which is denoted as $\mathrm{bw}(t)$. The precise definition of the many-sorted logic can be found for example in Barrett et al. [3]. The precise description of bit-vector theory can be found for example in the paper describing complexity of quantified bit-vector theory by Kovásznai et al. [10].

A signature $\Sigma$ is a set of *uninterpreted function symbols*, which is disjoint with the set of all interpreted bit-vector function and predicate symbols. Each function symbol $f \in \Sigma$ has an associated arity $k \in \mathbb{N}_0$ and a sort $(n_1, n_2, \ldots, n_k, n) \in \mathbb{N}^{k+1}$, where the numbers $n_i$ represent bit-widths of the arguments of $f$ and $n$ represents the bit-width of its result. A $\Sigma$-structure $\mathcal{M}$ maps each uninterpreted function $f$ of the sort $(n_1, n_2, \ldots, n_k, n)$ to a function of the type $\mathcal{BV}_{n_1} \times \mathcal{BV}_{n_2} \times \ldots \times \mathcal{BV}_{n_k} \to \mathcal{BV}_n$, and each variable $x^{[n]}$ to a bit-vector value in $\mathcal{BV}_n$.

For a $\Sigma$-structure $\mathcal{M}$, we define the evaluation function $[\![\_]\!]_{\mathcal{M}}$, which assigns to each term $t$ the bit-vector $[\![t]\!]_{\mathcal{M}}$ obtained by (i) substituting each variable $x$ in $t$ by its value $\mathcal{M}(x)$ given by $\mathcal{M}$ and (ii) evaluating all interpreted functions and predicates using their given semantics and all uninterpreted functions using their interpretations given by $\mathcal{M}(f)$. Similarly, the function $[\![\_]\!]_{\mathcal{M}}$ assigns to each formula $\varphi$ the Boolean value $[\![\varphi]\!]_{\mathcal{M}}$ obtained by substituting free variables in $\varphi$ by values given by $\mathcal{M}$ and evaluating all functions, predicates, logical operators etc. according to $\mathcal{M}$ and the standard semantics. A formula $\varphi$ is *satisfiable* if

$[\![\varphi]\!]_{\mathcal{M}} = \top$ for some $\Sigma$-structure $\mathcal{M}$; it is *unsatisfiable* otherwise. A $\Sigma$-structure $\mathcal{M}$ is called a *model* of $\varphi$ whenever $[\![\varphi]\!]_{\mathcal{M}} = \top$.

The *Skolemization* of a formula $\varphi$, denoted *skolemize*$(\varphi)$, is a formula that is obtained from $\varphi$ by replacing each existentially quantified variable $x^{[n]}$ in $\varphi$ by a fresh uninterpreted function symbol $f_{x^{[n]}}$ that has as arguments all variables that are universally quantified above $x^{[n]}$ in the syntactic tree of the formula $\varphi$. Skolemization preserves the satisfiability of the input formula $\varphi$ [6].

For a satisfiable formula $\varphi$ without uninterpreted functions, a model $\mathcal{M}$ of *skolemize*$(\varphi)$ assigns a bit-vector $\mathcal{M}(y^{[m]}) \in \mathcal{BV}_m$ to each free variable $y^{[m]}$ in $\varphi$, and a function $\mathcal{M}(f_{x^{[n]}})$ to each Skolem function $f_{x^{[n]}}$, which corresponds to an existentially quantified variable $x^{[n]}$ in the formula $\varphi$. The functions $\mathcal{M}(f_{x^{[n]}})$ may be arbitrary functions (of the corresponding type) in the mathematical sense. To be able to work with the model, we use the notion of a *symbolic model*, in which the functions $\mathcal{M}(f_{x^{[n]}})$ are represented symbolically by terms. Namely, $\mathcal{M}(f_{x^{[n]}})$ is a bit-vector term of bit-width $n$ whose free variables may be only the variables that are universally quantified above $x^{[n]}$ in the original formula $\varphi$. In the further text, we treat the symbolic models as if they assign a term not to the corresponding Skolem function $f_{x^{[n]}}$, but directly to the existentially quantified variable $x^{[n]}$. For example, the formula

$$\forall x^{[32]} \forall y^{[32]}\ \exists z^{[32]}\ \left(x^{[32]} \cdot (y^{[32]} + z^{[32]}) = 0^{[32]}\right)$$

from the introduction has a symbolic model $\{z^{[32]} \mapsto -y^{[32]}\}$.

For a sentence $\varphi$, the dual notion to the symbolic model is a *symbolic countermodel*. The symbolic countermodel of a sentence $\varphi$ is a symbolic model of the negation normal form of $\neg\varphi$, i.e., a $\Sigma$-structure $\mathcal{M}$ that assigns to each *universally* quantified variable $x^{[n]}$ in $\varphi$ a term of bit-width $n$ whose free variables may be only the *existentially* quantified variables that are quantified above $x^{[n]}$ in the original formula $\varphi$.

We can define substitution of a symbolic (counter)model into a given formula. We define this notion more generally to allow substitution of an arbitrary assignment that assigns terms to variables of the formula. For each such assignment $\mathcal{A}$ and a formula $\varphi$, we denote as $\mathcal{A}(\varphi)$ the result of simultaneous substitution of the term $\mathcal{A}(x^{[n]})$ for each variable $x^{[n]}$ in the domain of $\mathcal{A}$ and removing all quantifications of the substituted variables. For example, the value of

$$\mathcal{A}\left(\forall x^{[32]} \forall y^{[32]}\ \exists z^{[32]}\ (x^{[32]} \cdot (y^{[32]} + z^{[32]}) = 0^{[32]})\right)$$

for $\mathcal{A} = \{z^{[32]} \mapsto -y^{[32]}\}$ is $\forall x^{[32]} \forall y^{[32]}\ (x^{[32]} \cdot (y^{[32]} + (-y^{[32]})) = 0^{[32]})$.

## 3 Formula Reduction and Model Extension

This section describes the basic building blocks of our new technique, namely reduction of bit-widths in a given formula and extension of bit-widths in a given model or countermodel.

## 3.1 Reduction of Bit-Widths in Formulas

The goal of the reduction procedure is to reduce the bit-widths of all variables and constants in a given formula so that they do not exceed a given bit-width. In fact, we reduce bit-widths of all terms in the formula in order to keep the formula type consistent. A similar reduction is defined in our previous paper [7], but only for a simpler fragment of the considered logic.

As the first step, we inductively define a function $rt$ that takes a term and a bit-width $bw \in \mathbb{N}$ and reduces all subterms of the term. The function always cuts off all most-significant bits above the given bit-width $bw$. As the base case, we define the reduction on constants and variables.

$$\begin{aligned} rt(m^{[n]}, bw) &= (m \bmod 2^{\min(n,bw)})^{[\min(n,bw)]} \\ rt(x^{[n]}, bw) &= x^{[\min(n,bw)]} \end{aligned}$$

Further, let $\circ$ range over the set $\{+, \cdot, \&, |, \oplus, \ll, \gg\}$ of binary functions that produce results of the same bit-width as the bit-width of their arguments. To reduce a term $t_1 \circ t_2$, we just need to reduce the arguments.

$$\begin{aligned} rt(t_1 \circ t_2, bw) &= rt(t_1, bw) \circ rt(t_2, bw) \\ rt(-t_1, bw) &= -rt(t_1, bw) \end{aligned}$$

The most interesting cases are the functions that change bit-widths. As the first case, let $ext_n$ be a function that extends its argument with $n$ most-significant zero bits ($\mathtt{zeroExt}_n$) or with $n$ copies of the sign bit ($\mathtt{signExt}_n$). A term $ext_n(t)$ where $t$ has $bw$ or more bits is reduced just to $rt(t, bw)$. Indeed, the function $ext_n$ is completely removed as the bits it would add exceed the maximal bit-width. When $t$ has less than $bw$ bits, we apply the extension function but we decrement its parameter if the bit-width of the resulting term should exceed $bw$. Moreover, we also apply the reduction function to $t$ to guarantee that bit-widths of its subterms do not exceed $bw$.

$$rt(ext_n(t), bw) = \begin{cases} rt(t, bw) & \text{if } \mathrm{bw}(t) \geq bw \\ ext_{\min(n, bw - \mathrm{bw}(t))}(rt(t, bw)) & \text{if } \mathrm{bw}(t) < bw \end{cases}$$

As the second case, consider a term $\mathtt{extract}_l^u(t)$ that represents bits of $t$ between positions $u$ and $l$ (including these positions). The reduction is defined by one of the following three subcases according to the relation of $bw$ and positions $u$ and $l$. Recall that $u \geq l$, the bit-width of the original term is $u - l + 1$, and it has to be reduced to $m = \min(u - l + 1, bw)$.

- If both $u$ and $l$ point to some of the $bw$ least-significant bits of $t$ (i.e., $bw > u$), the positions $u$ and $l$ of $rt(t, bw)$ are defined, and so we just reduce the argument $t$ and do not change the parameters of `extract`.
- If $l$ points to some of the $bw$ least-significant bits of $t$ but $u$ does not (i.e., $u \geq bw > l$), we reduce the argument $t$, extract its most-significant bits up to the position $l$, and extend the result with most-significant zero bits such

that the bit-width of the result is $m$. These additional zero bits correspond to the positions that should be extracted, but are not present in the term $rt(t, bw)$.
- If both positions $u$ and $l$ point outside the $bw$ least-significant bits of $t$ (i.e., $l \geq bw$), we replace the term with the bit-vector of zeroes of the length $m$.

In the following formal definition, we denote by $o$ the bit-width of term $t$ after reduction, i.e., $o = \mathrm{bw}(rt(t, bw)) = \min(\mathrm{bw}(t), bw)$.

$$rt(\mathsf{extract}_l^u(t), bw) = \begin{cases} \mathsf{extract}_l^u(rt(t, bw)) & \text{if } bw > u \\ ext_{m-(o-l)}(\mathsf{extract}_l^{o-1}(rt(t, bw))) & \text{if } u \geq bw > l \\ 0^{[m]} & \text{if } l \geq bw \end{cases}$$

where $ext \in \{\mathsf{signExt}, \mathsf{zeroExt}\}$ can be chosen during the implementation.

Finally, reduction of a term $\mathsf{concat}(t_1, t_2)$ representing concatenation of $t_1$ and $t_2$ is given by one of the following two cases. Note that the reduced term should have the size $m = \min(\mathrm{bw}(t_1) + \mathrm{bw}(t_2), bw)$. If $\mathrm{bw}(t_2) \geq bw$, the term is reduced to $rt(t_2, bw)$ as the bits of $t_1$ exceed the desired maximal bit-width. In the opposite case, we reduce both $t_1$ and $t_2$ and create the term containing all the bits of the reduced term $t_2$ preceded by $m - \mathrm{bw}(t_2)$ least-significant bits of the reduced term $t_1$.

$$rt(\mathsf{concat}(t_1, t_2), bw) =$$
$$= \begin{cases} rt(t_2, bw) & \text{if } \mathrm{bw}(t_2) \geq bw \\ \mathsf{concat}\left(\mathsf{extract}_0^{m-\mathrm{bw}(t_2)-1}\big(rt(t_1, bw)\big), rt(t_2, bw)\right) & \text{if } \mathrm{bw}(t_2) < bw \end{cases}$$

Now we define a function $rf$ that reduces the maximal bit-widths of all terms in a given formula to a given value $bw$. The function is again defined inductively using the function $rt$ in the base case to reduce arguments of predicate symbols. The rest of the definition is straightforward.

$$\begin{aligned} rf(t_1 \bowtie t_2, bw) &= rt(t_1, bw) \bowtie rt(t_2, bw) && \text{for } \bowtie \in \{=, \leq_u, \leq_s\} \\ rf(\neg\varphi, bw) &= \neg rf(\varphi, bw) \\ rf(\varphi_1 \diamond \varphi_2, bw) &= rf(\varphi_1, bw) \circ rf(\varphi_2, bw) && \text{for } \diamond \in \{\wedge, \vee\} \\ rf(Qx^{[n]}.\varphi, bw) &= Qx^{[\min(n, bw)]}.rf(\varphi, bw) && \text{for } Q \in \{\forall, \exists\} \end{aligned}$$

## 3.2 Extending Bit-Widths of Symbolic Models

If a reduced formula is satisfiable and its symbolic model $\mathcal{M}$ is obtained, it cannot be directly substituted into the original formula. It first needs to be *extended* to the original bit-widths. Intuitively, for each result $\mathcal{M}(x) = t$, where the original bit-width of the variable $x$ is $n$, we

1. increase bit-widths of all variables in $t$ to match the bit-widths in the original formula $\varphi$,

2. for each operation whose arguments need to have the same bit-width, we increase bit-width of the argument with the smaller bit-width to match the bit-width of the other argument,
3. change the bit-width of the resulting term to match the bit-width of the original variable $x^{[n]}$.

In the formalization, we need to know bit-widths of the variables in the original formula. Therefore, for a formula $\varphi$, we introduce the function $\mathrm{bws}_\varphi$ that maps each variable name $x$ in $\varphi$ to its original bit-width in $\varphi$. For example, $\mathrm{bws}_{x^{[32]}+y^{[32]}=0^{[32]}}(x) = 32$. Further, we use the function *adjust*, which adjusts the bit-width of the given term $t$ to the given bit-width $bw$.

$$adjust(t, bw) = \begin{cases} t & \text{if } \mathrm{bw}(t) = bw \\ ext_{bw-\mathrm{bw}(t)}(t) & \text{if } \mathrm{bw}(t) < bw \\ \mathtt{extract}_0^{bw-1}(t) & \text{if } \mathrm{bw}(t) > bw \end{cases}$$

where $ext \in \{\mathtt{signExt}, \mathtt{zeroExt}\}$ can be chosen during the implementation.

For each term $t$ of the reduced model, we now recursively construct a term $\overline{t}$, which uses only the variables of the original formula and is well-sorted. In other words, this construction implements the first two steps of the symbolic model extension described above.

As the base cases, we keep the bit-width of all constants and extend the bit-width of all variables to their original bit-widths in $\varphi$.

$$\overline{m^{[n]}} = m^{[n]}$$
$$\overline{x^{[n]}} = x^{[\mathrm{bws}_\varphi(x)]}$$

For any operation $\circ \in \{+, \cdot, \&, |, \oplus, \ll, \gg\}$ that requires arguments of the same bit-widths, we may need to extend the shorter of these arguments.

$$\overline{t_1 \circ t_2} = adjust\big(\overline{t_1}, \max(\mathrm{bw}(\overline{t_1}), \mathrm{bw}(\overline{t_2}))\big) \circ adjust\big(\overline{t_2}, \max(\mathrm{bw}(\overline{t_1}), \mathrm{bw}(\overline{t_2}))\big)$$

For the remaining operations, the construction is straightforward.

$$\overline{-t_1} = -\,\overline{t_1}$$
$$\overline{ext_n(t_1)} = ext_n(\overline{t_1}) \quad \text{for } ext \in \{\mathtt{zeroExt}, \mathtt{signExt}\}$$
$$\overline{\mathtt{extract}_j^i(t_1)} = \mathtt{extract}_j^i(\overline{t_1})$$
$$\overline{\mathtt{concat}(t_1, t_2)} = \mathtt{concat}(\overline{t_1}, \overline{t_2})$$

Now we complete the symbolic model extension with its third step. Formally, for a symbolic model $\mathcal{M}$ we define a model extension $extendM(\mathcal{M})$ that assigns to each variable $x$ in the domain of $\mathcal{M}$ the term $\overline{\mathcal{M}(x)}$ adjusted to the original bit-width of $x$.

$$extendM(\mathcal{M})(x) = adjust(\overline{\mathcal{M}(x)}, \mathrm{bws}_\varphi(x)).$$

*Example 1.* Consider a formula $\varphi$ that contains variables $x^{[8]}$, $y^{[8]}$, $z^{[4]}$, $v^{[8]}$, $w^{[4]}$. Suppose that we have the model $\mathcal{M}$ of $rf(\varphi, 4)$ given below. With the parameter *ext* of *adjust* set to `signExt`, the assignment $extendM(\mathcal{M})$ is defined as follows.

$$\begin{aligned} \mathcal{M} = \{x^{[4]} &\mapsto v^{[4]} + 3^{[4]}, & extendM(\mathcal{M}) = \{x^{[8]} &\mapsto v^{[8]} + 3^{[8]}, \\ y^{[4]} &\mapsto w^{[4]}, & y^{[8]} &\mapsto \mathtt{signExt}_4(w^{[4]}), \\ z^{[4]} &\mapsto v^{[4]}, & z^{[4]} &\mapsto \mathtt{extract}_0^3(v^{[8]})\} \end{aligned}$$

Note that the numeral $3^{[8]}$ in $extendM(\mathcal{M})$ arises by evaluation of the ground term $\mathtt{signExt}_4(3^{[4]})$.

*Note on additional* SMT-LIB *operations.* The syntax of the BV theory given in SMT-LIB actually contains more predicates and functions than we have defined. The constructions presented in Subsections 3.1 and 3.2 can be extended to cover these additional predicates and functions mostly very easily. One interesting case is the *if-then-else* operator $\mathtt{ite}(\varphi, t_1, t_2)$ where the first argument is a formula instead of a term. To accommodate this operator, the reduction functions *rt* and *rf* are defined as mutually recursive, and the symbolic model extension has to be enriched to handle not only terms, but also formulas. All these extensions can be found in the dissertation of M. Jonáš [8]. Note that `ite` indeed appears in symbolic models in practice.

## 4 Algorithm

In this section, we propose an algorithm that employs bit-width reductions and extensions to decide satisfiability of an input formula. In the first subsection, we describe a simpler approach that can only decide that a formula is satisfiable. The following subsection dualizes this approach to unsatisfiable formulas. We then show how to combine these two approaches in a single algorithm, which is able to decide both satisfiability and unsatisfiability of a formula.

### 4.1 Checking Satisfiability Using Reductions and Extensions

Having defined the functions *rf* (see Subsection 3.1), which reduces bit-widths in a formula, and *extendM* (see Subsection 3.2), which extends bit-widths in a symbolic model of the reduced formula, it is fairly straightforward to formulate an algorithm that can decide satisfiability of a formula using reduced bit-widths.

This algorithm first reduces the bit-widths in the input formula $\varphi$, thus obtains a reduced formula $\varphi_{red}$, and checks its satisfiability. If the formula is not satisfiable, the algorithm computes a new reduced formula $\varphi_{red}$ with an increased bit-width and repeats the process. If, on the other hand, the reduced formula $\varphi_{red}$ is satisfiable, the algorithm obtains its symbolic model $\mathcal{M}$, which assigns a term to each existentially quantified and free variable of the formula $\varphi_{red}$.

The model is then extended to the original bit-widths of the variables in the formula $\varphi$ and the extended model is substituted into the original formula $\varphi$, yielding a formula $\varphi_{sub}$. The formula $\varphi_{sub}$ may not be quantifier-free, but it contains only universally quantified variables and no free variables. The formula $\varphi_{sub}$ may therefore be checked for satisfiability by a solver for quantifier-free bit-vector formulas: the solver can be called on the formula $\varphi_{sub}^{\neg}$ that results from removing all quantifiers from the formula $\neg\varphi_{sub}$ transformed to the negation normal form. Since the formula $\varphi_{sub}$ is closed, the satisfiability of $\varphi_{sub}^{\neg}$ implies unsatisfiability of $\varphi_{sub}$ and vice versa. Finally, if the formula $\varphi_{sub}$ is satisfiable, so is the original formula. If the formula $\varphi_{sub}$ is not satisfiable, the process is repeated with an increased bit-width.

*Example 2.* Consider the formula $\varphi \equiv \forall x^{[32]} \exists y^{[32]} \, (x^{[32]} + y^{[32]} = 0^{[32]})$. Reduction to 2 bits yields the formula $rf(\varphi, 2) \equiv \forall x^{[2]} \exists y^{[2]} \, (x^{[2]} + y^{[2]} = 0^{[2]})$. An SMT solver can decide that this formula is satisfiable and its symbolic model is $\{y^{[2]} \mapsto -x^{[2]}\}$. An extended candidate model is then $\{y^{[32]} \mapsto -x^{[32]}\}$. After substituting this candidate model into the formula, one gets the formula $\varphi_{sub} \equiv \forall x^{[32]} \, (x^{[32]} + (-x^{[32]}) = 0^{[32]})$. Negating the formula $\varphi_{sub}$ and removing all the quantifiers yields the quantifier-free formula $(x^{[32]} + (-x^{[32]}) \neq 0^{[32]})$, which is unsatisfiable. Therefore, the formula $\varphi_{sub}$ is satisfiable and, in turn, the original formula $\varphi$ is satisfiable as well.

The correctness of the approach is guaranteed by the following theorem.

**Theorem 1 ([8, Theorem 11.1]).** *Let $\varphi$ be a formula in the negation normal form and $\mathcal{A}$ a mapping that assigns terms only to free and existentially quantified variables of $\varphi$. If each term $\mathcal{A}(x)$ contains only universal variables that are quantified in $\varphi$ before the variable $x$, satisfiability of $\mathcal{A}(\varphi)$ implies satisfiability of $\varphi$.*

## 4.2 Dual Algorithm

The algorithm of the previous subsection can improve performance of an SMT solver only for satisfiable formulas. However, its dual version can be used to improve performance on unsatisfiable formulas. In the dual algorithm, one can decide unsatisfiability of a formula by computing a countermodel of a reduced formula and verifying it against the original formula. More precisely, if the solver decides that the reduced formula $\varphi_{red}$ is unsatisfiable, one can extend its countermodel $\mathcal{M}$, substitute the extended countermodel into the original formula, obtaining a formula $\varphi_{sub}$ which contains only existentially quantified variables. Satisfiability of $\varphi_{sub}$ can be again checked by a solver for quantifier-free formulas applied to $\varphi_{sub}$ after removing all its existential quantifiers. If the formula $\varphi_{sub}$ is unsatisfiable, the original formula $\varphi$ must have been unsatisfiable. If the formula $\varphi_{sub}$ is satisfiable, the process is repeated with an increased bit-width.

*Example 3.* Consider the formula $\varphi = \forall y^{[32]} \, (x^{[32]} + y^{[32]} = 0^{[32]})$. Reduction to one bit yields the formula $rf(\varphi, 1) = \forall y^{[1]} \, (x^{[1]} + y^{[1]} = 0^{[1]})$. This formula can be decided as unsatisfiable by an SMT solver and its countermodel is $\{y^{[1]} \mapsto -x^{[1]} + 1^{[1]}\}$. The extension of this countermodel to the original bit-widths is then $\{y^{[32]} \mapsto -x^{[32]} + 1^{[32]}\}$. After substituting this candidate countermodel to the original formula, one obtains the quantifier-free formula $\varphi_{sub} = (x^{[32]} + (-x^{[32]} + 1^{[32]}) = 0^{[32]})$, which is unsatisfiable. The original formula $\varphi$ is thus unsatisfiable.

The correctness of the dual algorithm is guaranteed by the following theorem.

**Theorem 2 ([8, Theorem 11.2]).** *Let $\varphi$ be a formula in the negation normal form and $\mathcal{A}$ a mapping that assigns terms only to universally quantified variables of $\varphi$. If each term $\mathcal{A}(x)$ contains only free and existential variables that are quantified in $\varphi$ before the variable $x$, unsatisfiability of $\mathcal{A}(\varphi)$ implies unsatisfiability of $\varphi$.*

### 4.3 Combined Algorithm

Now we combine the two algorithms into one. In the rest of this section, we suppose that there exists a *model-generating solver* that produces symbolic models for satisfiable quantified bit-vector formulas and countermodels for unsatisfiable ones. Formally, let `solve`($\varphi$) be the function that returns (`sat`, *model*) if $\varphi$ is satisfiable and (`unsat`, *countermodel*) in the opposite case.

Further, we use SMT queries to check the satisfiability of $\varphi_{sub}$. Generally, these queries can be answered by a different SMT solver than the model-generating one. We call it *model-validating solver* and suppose that it has the function `verify`($\psi$) which returns either `sat` or `unsat` reflecting the satisfiability of $\psi$.

Using these two solvers, the algorithm presented in Listing 1.1 combines the techniques of the two preceding subsections. This algorithm first reduces the bit-widths in the input formula to 1 and checks satisfiability of the reduced formula $\varphi_{red}$ by the model-generating solver. According to the result, we try to validate either the extended symbolic model or the extended symbolic countermodel with the model-validating solver. If the validation succeeds, the satisfiability of the original formula is decided. Otherwise, we repeat the process but this time we reduce the bit-widths in the input formula to twice the value used in the previous iteration. The algorithm terminates at the latest in the iteration when the value of `bw` is so high that the formula $\varphi_{red}$ is identical to the input formula $\varphi$. In this case, the model-generating solver provides a model or a countermodel $\mathcal{M}$ of $\varphi$. As $\mathcal{M}$ contains the unchanged variables of $\varphi$, its extension *extendM*($\mathcal{M}$) is identical to $\mathcal{M}$ and the model-validating solver has to confirm the result.

## 5 Implementation

We have implemented the proposed algorithm in a proof-of-concept tool. However, our implementation differs in several aspects from the described algorithm. This section explains all these differences and provides more details about the implementation.

Listing 1.1: The combined algorithm for checking satisfiability of $\varphi$ using bit-width reductions and extensions.

```
bw ← 1
while (true) {
    $\varphi_{red} \leftarrow rf(\varphi, \mathsf{bw})$
    (result, $\mathcal{M}$) ← solve($\varphi_{red}$)
    $\mathcal{A} \leftarrow extendM(\mathcal{M})$
    $\varphi_{sub} \leftarrow \mathcal{A}(\varphi)$
    if (result == sat) {
        $\varphi_{sub}^{\neg}$ ← removeQuantifiers($\neg\varphi_{sub}$)
        verificationResult ← verify($\varphi_{sub}^{\neg}$)
        if (verificationResult == unsat) return SAT
    }
    if (result == unsat) {
        $\varphi_{sub}$ ← removeQuantifiers($\varphi_{sub}$)
        verificationResult ← verify($\varphi_{sub}$)
        if (verificationResult == unsat) return UNSAT
    }
    bw ← increaseBW(bw)
}
```

## 5.1 Model-Generating Solver

As the model-generating solver, we use Boolector 3.2.0 as it can return symbolically expressed Skolem functions as models of satisfiable quantified formulas, which is crucial for our approach. Unfortunately, Boolector does not satisfy some requirements that we imposed on the model-generating solver.

First, the symbolic model $\mathcal{M}$ returned by Boolector may not contain terms for all existentially quantified variables of the input formula $\varphi$. Therefore, the formula $\varphi_{sub}$ may still contain both existentially and universally quantified variables and we cannot employ an SMT solver for quantifier-free formulas as the model-validation solver. Our implementation thus uses a model-validating solver that supports quantified formulas. An alternative solution is to extend $\mathcal{M}$ to all existentially quantified variables, for example by assigning $0^{[n]}$ to each existentially quantified variable $x^{[n]}$ that is not assigned by $\mathcal{M}$. This allows using a solver for quantifier-free formulas as the model-validating solver. However, our preliminary experiments indicate that this alternative solution does not bring any significant benefit. Moreover, the best performing SMT solvers for the quantifier-free bit-vector formulas can also handle quantified formulas.

Second, Boolector returns symbolic models only for satisfiable formulas and cannot return symbolic countermodels. We alleviate this problem by running two parallel instances of Boolector: one on the original formula $\varphi$ and one on the formula $\neg\varphi'$, where $\varphi'$ arises from $\varphi$ by existential quantification of all free variables. We then use only the result of the solver that decides that the formula is satisfiable; if $\varphi$ is satisfiable, we get its symbolic model, if $\neg\varphi'$ is satisfiable, we get its symbolic model, which is a symbolic countermodel of $\varphi$. Effectively, this is equivalent to running the algorithm of Listing 1.1 without the lines 12–16 in two parallel instances: one on $\varphi$ and the other on $\neg\varphi'$. This is what our implementation actually does.

### 5.2 Portfolio Solver

The aim of our research is to improve the performance of an SMT solver for the BV theory using the bit-width reductions and extensions. The solver is used as the model-validating solver. We investigate two implementations:

– To see real-world benefits, we run the original solver in parallel with the two processes that use bit-width reductions. The result of the first process that decides the satisfiability of the input formula is returned. The schematic overview of our portfolio solver is presented in Figure 1. In this variant, if the reducing solvers reach the original bit-width of the formula, they return `unknown`.
– To see the negative overhead of reductions, we also consider a variant of the above-mentioned approach, but without the middle thread with original solver. In this variant, the reducing solvers are additionally executed for the original bit-width in their last iteration.

Our experimental implementation is written in C++ and Python. It utilizes the C++ API of Z3 [5] to parse the input formula in the SMT-LIB format. The Z3 API is also used in the implementation of formula reductions and some simplifications (conversion to the negation normal form and renaming bound variables to have unique names). The only part written in Python is a simple wrapper that executes the three parallel threads and collects their results. As the parameters, we use $ext$ = `zeroExt` in $rt$, $ext$ = `signExt` in $adjust$, and `increaseBW`$(x) = 2 * x$. These parameters had the best performance during our preliminary evaluation, but can be changed. The implementation is available at: https://github.com/martinjonas/bw-reducing-solver

## 6 Experimental Evaluation

We have evaluated the impact of our technique on the performance of three leading SMT solvers for the BV theory: Boolector 3.2.0 [11], CVC4 1.6 [1], and Q3B 1.0 [9]. Each of these solvers has been employed as the model-validating solver, while the model-generating solver remains the same, namely Boolector. For the evaluation, we have used all 5741 quantified bit-vector formulas from the SMT-LIB benchmark repository [2]. The formulas are divided into 8 benchmark families coming from different sources.

All experiments were performed on a Debian machine with two six-core Intel Xeon E5-2620 2.00GHz processors and 128 GB of RAM. Each benchmark run was limited to use 16 GB of RAM and 5 minutes of wall time. All measured times are wall times. For reliable benchmarking we employed BenchExec [4].

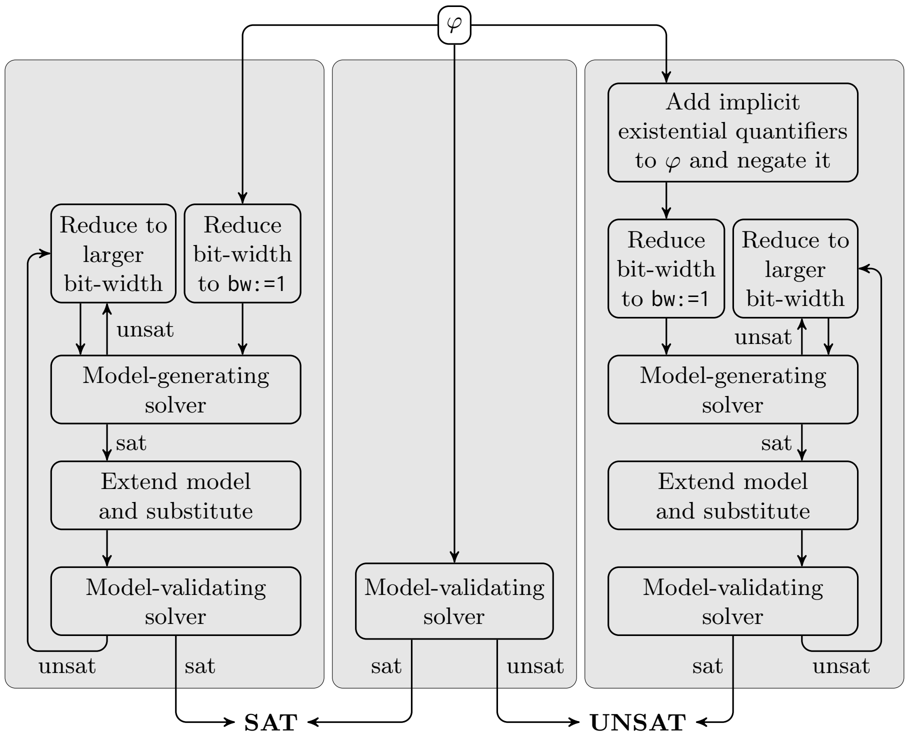
$\varphi$
Reduce to larger bit-width
Reduce bit-width to bw:=1
unsat
Model-generating solver
sat
Extend model and substitute
Model-validating solver
unsat
sat
Model-validating solver
sat
unsat
Add implicit existential quantifiers to $\varphi$ and negate it
Reduce bit-width to bw:=1
Reduce to larger bit-width
unsat
Model-generating solver
sat
Extend model and substitute
Model-validating solver
sat
unsat
SAT
UNSAT

Fig. 1: High-level overview of the portfolio solver. The three shaded areas are executed in parallel and the first result is returned.

## 6.1 Boolector

First, we have evaluated the impact of our technique on the performance of Boolector 3.2.0. We have compared the vanilla Boolector (referred to as `btor`), our portfolio solver running Boolector as both model-generating and model-validating solver (`btor-r`), and the portfolio variant without the original solver (`btor-r-no`). The numbers of formulas of individual benchmark families solved by the three solvers can be found in the corresponding columns of Table 1. While `btor-r-no` is not very competitive, the full portfolio solver was able to solve 22 formulas more than Boolector itself. Note that this amounts to 8.6% of the formulas unsolved by Boolector. The scatter plots in Figure 2 shows the performance of the solvers. With the full portfolio approach, our technique can also significantly reduce the running time of Boolector on a non-trivial number of both satisfiable and unsatisfiable formulas from various benchmark families.

We have also investigated the reduction bit-width that was necessary to improve the performance of Boolector. Among all executions of the full portfolio solver, 475 benchmarks were actually decided by one of the two parallel threads that perform bit-width reductions. From these 475 benchmarks, 193 were de-

Table 1: The table shows for each benchmark family and each solver the number of benchmarks that were solver by the solver within a given timeout.

| Family | Total | `btor` | `btor-r` | `btor-r-no` | `btor|cvc4` | `btor|cvc4-r` | `btor|q3b` | `btor|q3b-r` |
|---|---|---|---|---|---|---|---|---|
| 2017-Preiner-keymaera | 4035 | 4019 | 4022 | 4020 | 4025 | 4027 | 4025 | 4028 |
| 2017-Preiner-psyco | 194 | 193 | 193 | 129 | 193 | 193 | 193 | 193 |
| 2017-Preiner-scholl-smt08 | 374 | 299 | 304 | 224 | 306 | 306 | 327 | 328 |
| 2017-Preiner-tptp | 73 | 70 | 73 | 69 | 73 | 73 | 73 | 73 |
| 2017-Preiner-ua | 153 | 153 | 153 | 23 | 153 | 153 | 153 | 153 |
| 20170501-Heizmann-ua | 131 | 28 | 30 | 25 | 130 | 130 | 128 | 129 |
| 2018-Preiner-cav18 | 600 | 549 | 554 | 477 | 577 | 577 | 590 | 590 |
| wintersteiger | 181 | 152 | 156 | 125 | 167 | 169 | 172 | 174 |
| Total | 5741 | 5463 | 5485 | 5092 | 5624 | 5628 | 5661 | 5668 |

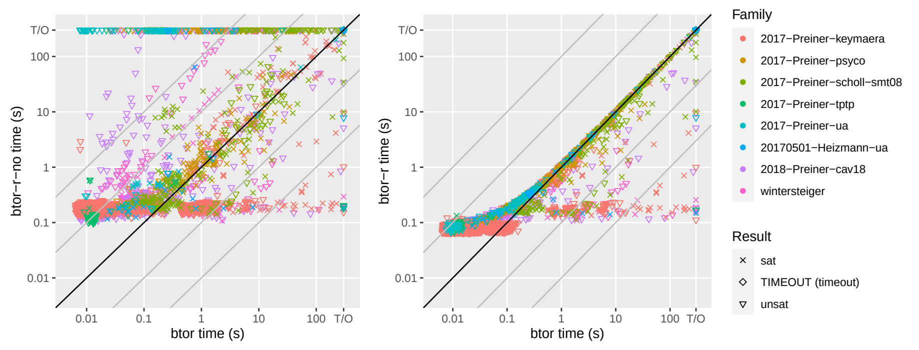

Fig. 2: Scatter plots of wall times of the solver `btor` vs the solvers `btor-r` and `btor-r-no`. Each point represents one benchmark, its color shows the benchmark family, and its shape shows its satisfiability.

cided using the bit-width of 1 bit, 141 using 2 bits, 111 using 4 bits, 23 using 8 bits, and 7 using 16 bits.

## 6.2 CVC4 and Q3B

We have also performed evaluations with CVC4 and Q3B as model-validating solvers. This yields the following four solvers: `cvc4`, `q3b` are the vanilla CVC4 and Q3B, respectively; `cvc4-r`, `q3b-r` are the portfolio solvers using CVC4 and Q3B, respectively, as the model-validating solver.

Whenever the model-generating solver differs from the model-validating solver, the comparison is more involved. For example, the direct comparison of `cvc4` and

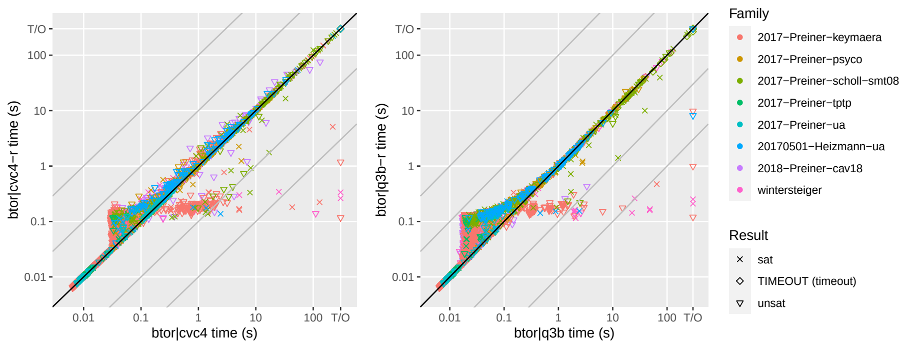

Fig. 3: Scatter plots of wall times of the virtual best solvers `btor|cvc4` vs. `btor|cvc4-r` (left) and `btor|q3b` vs. `btor|q3b-r` (right).

`cvc4-r` would be unfair and could be biased towards `cvc4-r`. This happens because models are provided by Boolector as the model-generating solver and the model validation may become trivial for CVC4, even if it could not solve the reduced formula alone. To eliminate this bias, we do not compare `cvc4` against `cvc4-r`, but the virtual-best solver from `btor` and `cvc4`, denoted as `btor|cvc4`, against the virtual-best solver from `btor` and `cvc4-r`, denoted as `btor|cvc4-r`. We thus investigate only the effect of reductions and not the performance of the model-generating solver on the input formula. Similarly, we compare the virtual-best solver `btor|q3b` against the virtual-best solver `btor|q3b-r`.

Table 1 shows the number of benchmarks solved by the compared solvers. In particular, reductions helped the virtual-best solver `btor|cvc4-r` to solve 4 more benchmarks than the solver `btor|cvc4`. This amounts to 3.4% of the benchmarks unsolved by `btor|cvc4`. For `btor|q3b-r`, the reductions help to solve 7 new benchmarks, i.e., 8.8% of unsolved benchmarks.

Similarly to the case of Boolector, reductions also help `btor|cvc4-r` to decide several benchmarks faster than the solver `btor|cvc4` without reductions. This can be seen on the first scatter plot in Figure 3. As the second scatter plot in this figure shows, reductions also help Q3B to solve some benchmarks faster.

All experimental data, together with additional results and all scripts that were used during the evaluation are available at: https://fi.muni.cz/~xstrejc/sat2020/

## 7 Conclusions

We have described an algorithm that improves performance of SMT solvers for quantified bit-vector formulas by reducing bit-widths in the input formula. We have shown that if used in a portfolio approach, our proof-of-concept implementation of this algorithm improves performance of state-of-the art SMT solvers Boolector, CVC4, and Q3B.

## References

1. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 171–177, 2011.
2. Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.
3. Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, pages 825–885. IOS Press, 2009.
4. Dirk Beyer, Stefan Löwe, and Philipp Wendler. Benchmarking and Resource Measurement. In *Model Checking Software - 22nd International Symposium, SPIN 2015, Proceedings*, volume 9232 of *Lecture Notes in Computer Science*, pages 160–178. Springer, 2015.
5. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.
6. John Harrison. *Handbook of Practical Logic and Automated Reasoning.* Cambridge University Press, 2009.
7. Martin Jonáš and Jan Strejček. Is satisfiability of quantified bit-vector formulas stable under bit-width changes? In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 488–497. EasyChair, 2018.
8. Martin Jonáš. *Satisfiability of Quantified Bit-Vector Formulas: Theory & Practice.* PhD thesis, Masaryk University, 2019.
9. Martin Jonáš and Jan Strejček. Q3B: an efficient BDD-based SMT solver for quantified bit-vectors. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, 2019, Proceedings, Part II*, pages 64–73, 2019.
10. Gergely Kovásznai, Andreas Fröhlich, and Armin Biere. Complexity of fixed-size bit-vector logics. *Theory Comput. Syst.*, 59(2):323–376, 2016.
11. Aina Niemetz, Mathias Preiner, Clifford Wolf, and Armin Biere. Btor2, BtorMC and Boolector 3.0. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 587–595, 2018.
12. Aleksandar Zeljić, Christoph M. Wintersteiger, and Philipp Rümmer. An Approximation Framework for Solvers and Decision Procedures. *J. Autom. Reasoning*, 58(1):127–147, 2017.