EPiC Series in Computing

Volume 57, 2018, Pages 488–497

LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Is Satisfiability of Quantified Bit-Vector Formulas Stable Under Bit-Width Changes?*

Martin Jonáš and Jan Strejček

Masaryk University, Brno, Czech Republic
{xjonas, strejcek}@fi.muni.cz

Abstract

In general, deciding satisfiability of quantified bit-vector formulas becomes harder with increasing maximal allowed bit-width of variables and constants. However, this does not have to be the case for formulas that come from practical applications. For example, software bugs often do not depend on the specific bit-width of the program variables and would manifest themselves also with much lower bit-widths. We experimentally evaluate this thesis and show that satisfiability of the vast majority of quantified bit-vector formulas from the SMT-LIB repository remains the same even after reducing bit-widths of variables to a very small number. This observation may serve as a starting-point for development of heuristics or other techniques that can improve performance of SMT solvers for quantified bit-vector formulas.

1 Introduction

In the modern world, as the computer software becomes still more ubiquitous and complex, there is an increasing need to test it and formally verify its correctness. Several approaches to software verification, such as symbolic execution or bounded model checking, rely on the ability to decide whether a given first-order formula in a suitable logical theory is satisfiable. To this end, many of the verifiers use Satisfiability Modulo Theories (SMT) solvers, which can solve precisely the task of checking satisfiability of a given first-order formula in a given logical theory. For describing software, the natural choice of a logical theory is the theory of *fixed-size bit-vectors* in which the objects are vectors of bits and the operations on them precisely reflect operations performed by computers. Moreover, in applications such as synthesis of invariants, ranking functions, or loop summaries, the formulas in question also naturally contain quantifiers [10, 21, 6, 16, 17].

For most of the decision procedures for satisfiability of quantifier-free bit-vector formulas, their time and space complexities grow with the increasing bit-widths of used variables. This is caused by the conversion of the input formula to the equisatisfiable propositional formula (*bit-blasting*), which is a fall-back strategy for most of the decision procedures. The problem with growing bit-width in turn applies to decision procedures for quantified bit-vectors based on the quantifier-instantiation used by Boolector [18], CVC4 [19], and Z3 [22], which employ a

*The authors have been supported by the Czech Science Foundation grant GBP202/12/G061.

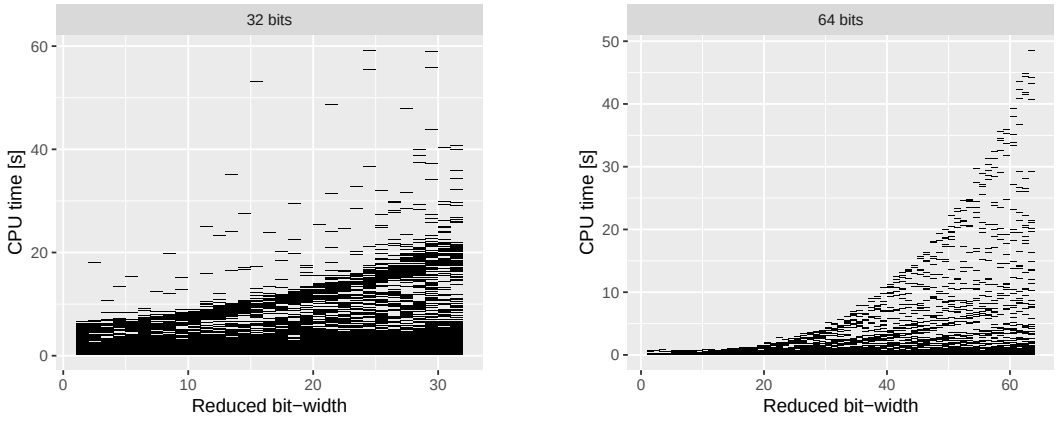

Figure 1: The scatter plots show CPU times of the solver CVC4 on a subset of 32bit and 64bit formulas from the SMT-LIB repository after changing their bit-width to the bit-width specified on the x-axis. The formulas are divided according to the original bit-width.

solver for quantifier-free formulas as a black box. The approach to quantified bit-vector formulas based on binary decision diagrams used by the solver Q3B [14] is sensitive to bit-width as well: as the bit-width increases, so do the sizes of the produced binary decision diagrams. These claims are supported by plots in Figure 1, which show solving times of the solver CVC4 on almost all 32bit and 64bit quantified bit-vector formulas from the SMT-LIB repository [2] after reducing their bit-widths to all values between 1 and the original bit-width. Details concerning the selection of benchmarks and reduction of their bit-widths are described in Section 3.

Under reasonable complexity-theory assumptions, the effect of increasing bit-width is also observable from the complexity-theory point of view. For both quantifier-free and quantified formulas, the respective complexity classes of deciding satisfiability of formulas with bit-widths encoded in unary and in binary differ: **NP** vs. **NEXPTIME** for quantifier-free formulas and **PSPACE** vs. **AEXP**(poly) for quantified formulas [15, 13]. Therefore, the decision problem has to become harder with the increasing bit-width, otherwise the complexity classes for unary and binary encoding would coincide.

For quantifier-free bit-vector formulas, this problem has been tackled several times. Bryant et al. have proposed an abstraction-based procedure that tries to solve the underapproximations of the input formula with reduced bit-widths of variables, whose satisfiability would prove the input formula satisfiable, and selectively increases their bit-widths if the underapproximation is unsatisfiable [5]. Fröhlich et al. have proposed stochastic local search approach that looks for models of quantifier-free formulas and thus avoids bit-blasting with its inherent space-complexity dependence on the bit-widths of the input formula [9]. This approach was improved by propagation rules by Niemetz et al. [18]. Zeljić et al. have developed the solver mcBV [24] that implements the model-constructing satisfiability calculus [8] that tries to construct a model directly and thus also avoids bit-blasting. Johannsen has shown how to compute a bit-width to which a formula can be reduced while preserving its satisfiability for a restricted class of formulas that represent *bitwise functions* [11, 12]. Kovásznai et al. used this observation regarding such formulas to show that the complexity of their satisfiability is the same for both unary and binary encoding of the bit-widths [15]. Considering a related theory of *floating-point arithmetic*, Zeljić et al. have introduced an approximation framework that produces mixed-approximations of the

original formula by reducing the bit-widths used for floating-point variables. After solving the mixed-approximation, the solver checks its result against the original formula and if it fails, it refines the approximation [25, 23]. In general, this approach works for arbitrary quantifier-free theory, but as far as we know, it has been implemented only for floating-point arithmetic.

For quantified bit-vector formulas, only one existing SMT solver tries to reduce bit-widths of some variables of the input formula: the solver Q3B uses approximations inspired by the abstractions by Bryant et al. and computes underapproximations and overapproximations by reducing bit-widths of existentially and universally quantified variables, respectively [14].

In this paper, we experimentally evaluate the thesis that the satisfiability of only a small number of quantified bit-vector formulas changes as the bit-widths of their variables decrease. We show that the satisfiability of the vast majority of quantified bit-vector formulas from the SMT-LIB repository remains the same even after reducing bit-widths of their variables to a very small number of bits. Therefore, the results of this experimental paper suggest that extending the techniques described for quantifier-free formulas to quantified formulas or designing novel techniques for reducing bit-widths of quantified bit-vector formulas could be worthwhile. These techniques could allow SMT solvers both solving more quantified bit-vector formulas and to solving them more quickly.

The paper is structured as follows: Section 2 provides the necessary background and notation for the bit-vector theory, Section 3 describes how we obtain formulas with reduced bit-widths, and Section 4 presents our experiments with the reduced formulas and results of these experiments. The following Section 5 discusses challenges arising from these results.

2 Preliminaries

This section briefly recalls the *theory of fixed-size bit-vectors* (*BV* or *bit-vector theory* for short). In the description, we assume familiarity with standard notions of many-sorted logic.

The bit-vector theory is a many-sorted first-order theory with infinitely many sorts corresponding to bit-vectors of various lengths, which are called their *bit-widths*. We denote a variable x of a sort corresponding to the bit-width $n \geq 1$ as $x^{[n]}$. The BV theory uses only three predicates, 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 functions including *addition* ($+$), *multiplication* ($\cdot$), *unsigned division* ($\div_u$), *signed division* ($\div_s$), *unsigned remainder* ($\%$), bit-wise *and* ($\&$), bit-wise *or* ($|$), bit-wise *exclusive or* ($\hat{}$), *left-shift* ($\ll$), *right-shift* ($\gg$), *concatenation* (`concat`), *zero extension* extending the argument with n most-significant zero bits ($\texttt{zeroExt}_n$), *sign extension* extending the argument with n copies of the sign bit ($\texttt{signExt}_n$), and *extraction* of n bits starting from position p ($\texttt{extract}_p^n$). 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 has an associated bit-width, which is denoted as $\mathrm{bw}(t)$.

For a valuation μ that assigns to each variable a value in its domain, $[\![\_]\!]_\mu$ denotes the evaluation function, which assigns to each term t the bit-vector obtained by substituting variables in t by their values given by μ and evaluating all functions. Similarly, the function $[\![\_]\!]_\mu$ assigns to each formula φ the value obtained by substituting free variables in φ with values given by μ and evaluating all functions, predicates, logic operators etc. A formula φ is *satisfiable* if $[\![\varphi]\!]_\mu = 1$ for some valuation μ; it is *unsatisfiable* otherwise.

The precise definition of many-sorted logic can be found for example in Barrett et al. [3]. The precise description of bit-vector theory and its operations can be found for example in the paper describing complexity of quantified bit-vector theory by Kovásznai et al. [15].

3 Obtaining Formulas with Reduced Bit-Widths

This section describes how we obtain formulas with the reduced bit-width from the original benchmarks. Given a formula φ and a desired bit-width $bw \geq 1$, the following procedure produces the formula $\mathit{reduceF}(\varphi, bw)$ in which all subterms have bit-width at most bw. In particular, $\mathit{reduceF}(\varphi, bw)$ is obtained from φ by:

- decreasing bit-width of all variables with bit-width more than bw to bw,
- replacing all numerals with bit-width more than bw by their bw least-significant bits,
- decreasing the numbers of added bits by all `zeroExt` and `signExt` functions so that the bit-width of the result is at most bw.

Formally, we introduce the following recursive function $\mathit{reduceT}$ that performs the above-described reduction of maximal bit-width on terms:

$$\begin{aligned}
\mathit{reduceT}(m^{[n]}, bw) &= (m \bmod 2^{\min(n,bw)})^{[\min(n,bw)]},\\
\mathit{reduceT}(x^{[n]}, bw) &= x^{[\min(n,bw)]},\\
\mathit{reduceT}(t_1 \diamond t_2, bw) &= \mathit{reduceT}(t_1, bw) \diamond \mathit{reduceT}(t_2, bw) \text{ for } \diamond \in \{+, \cdot, \div_s, \div_u, \%, \&, |, \hat{}, \ll, \gg\},\\
\mathit{reduceT}(\mathit{ext}_n(t), bw) &= \begin{cases} \mathit{reduceT}(t, bw) & \text{if } \mathrm{bw}(t) \geq bw,\\ \mathit{ext}_{\min(n, bw-\mathrm{bw}(t))}(t) & \text{if } \mathrm{bw}(t) < bw, \end{cases} \text{ for } \mathit{ext} \in \{\texttt{zeroExt}, \texttt{signExt}\}.
\end{aligned}$$

By using the function $\mathit{reduceT}$ on arguments of relation symbols in the formula, we obtain the function $\mathit{reduceF}$ that reduces maximal bit-widths in arbitrary formulas:

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

Note that the function $\mathit{reduceT}$ is undefined on terms that contain extraction or concatenation. We have decided to exclude formulas that contain these operations for the following reasons. For extraction, there are multiple arbitrary choices of bits to extract: for example, the extraction of the middle (i.e. the third) bit from a 5bit variable reduced to 3 bits could extract the middle (i.e. the second) bit or the third bit. Reduction of a formula containing concatenation may require reducing a single variable to multiple different bit-widths – although this is possible to achieve by adding extractions, this would change the semantics of the formula beyond merely changing the bit-widths of variables in which we are interested. For example, after reducing the formula $\texttt{concat}(x^{[4]}, y^{[4]}) = \texttt{concat}(y^{[4]}, x^{[4]})$ to 6 bits, the variable x would get reduced to 2 bits on the left-hand side, but would stay 4bit on the right-hand side.

We have written a simple tool that for an input formula φ in the SMT-LIB [1] format generates formulas $\mathit{reduceF}(\varphi, i)$ for all i between 1 and the maximal bit-width (included) of a subterm of the formula φ. The tool uses API of the SMT solver Z3 [7] and it is available at `https://gitlab.fi.muni.cz/xjonas/FormulaReducer`. Using this tool, we have generated reduced versions of all quantified bit-vector formulas from the SMT-LIB repository except for

- all 400 4bit and 32bit formulas from the *2018-Preiner-cav18* benchmark family. This does not lead to loss of information because all these formulas are generated from the remaining 64bit formulas by the function $\mathit{reduceF}$ with parameter 4 and 32, respectively;

Table 1: The table shows the formulas excluded from our evaluation according to their families. The column *total* shows a total number of formulas in each family (except for *2018-Preiner-cav18*, where all 4bit and 32bit benchmarks have been excluded). Next three columns show numbers of formulas excluded because of too large bit-width, use of operations `concat` and `extract`, and timeout of the solver for any of the reduced versions of the formula, respectively. The last column shows the number of remaining formulas on which the evaluation of effects of bit-width reduction on satisfiability was performed.

benchmark family	total	bw > 100	`concat` `extract`	T/O	remaining
2017-Preiner-keymaera	4035	0	0	65	3970
2017-Preiner-psyco	194	0	0	5	189
2017-Preiner-scholl-smt08	374	0	0	153	221
2017-Preiner-tptp	73	0	0	0	73
2017-Preiner-UltimateAutomizer	153	0	0	2	151
2017-Heizmann-UltimateAutomizer	131	0	4	7	120
2018-Preiner-cav18	200	0	40	30	130
wintersteiger	191	21	67	52	51
total	5351	21	111	314	4905

- 21 formulas that contain subterms of bit-width larger than 100 to keep the solving time reasonable;
- 111 formulas that use operations `concat` or `extract`.

Table 1 shows the numbers of such excluded benchmarks according to their families.

From the set of 5219 non-excluded original formulas, we have generated in total 173 105 corresponding formulas with the reduced bit-widths. An archive containing all these generated formulas can be found at `http://fi.muni.cz/~xstrejc/lpar2018/ReducedBW.tar.gz`.

4 Experimental Evaluation

We have evaluated satisfiability of all the resulting 173 105 formulas. For the evaluation, we have used the SMT solver CVC4 [19], as it is the winner of the SMT Competition 2018 in the category of quantified bit-vector formulas. The solver was run with 1 minute CPU time limit and 8 GiB RAM limit. For this, we employed BenchExec [4], a tool that allocates resources for a program execution and precisely measures their use. All experiments were performed on a Debian machine with two six-core Intel Xeon E5-2620 2.00GHz processors and 128 GB of RAM.

From the original 5219 formulas, 4905 were decided by CVC4 for all bit-widths. On the other hand, CVC4 exceeded the time limit on at least one bit-width on the remaining 314 formulas. The distribution of these non-decided formulas among benchmark families can be found in Table 1. We excluded these 314 formulas from the evaluation and performed the evaluation only on 4905 formulas with known status for all bit-widths. When grouped by their maximal bit-width, the set of evaluated formulas contains 10 formulas of bit-width 1, 10 of bit-width 8, 20 of bit-width 20, 4727 of bit-width 32, 1 of bit-width 33, 134 of bit-width 64, and 3 of bit-width 65.

Table 2: The table shows the numbers of benchmarks in the individual families whose satisfiability status is different for the original formula and for any reduced formula with bit-width at least 1, 2, 4, and 8 bits, respectively.

benchmark family	benchmarks	$\geq$ 1b	$\geq$ 2b	$\geq$ 4b	$\geq$ 8b
2017-Preiner-keymaera	3970	64	19	12	4
2017-Preiner-psyco	189	49	5	0	0
2017-Preiner-scholl-smt08	221	1	0	0	0
2017-Preiner-tptp	73	20	10	3	0
2017-Preiner-UltimateAutomizer	151	41	32	3	0
2017-Heizmann-UltimateAutomizer	120	29	20	6	5
2018-Preiner-cav18	130	4	3	3	3
wintersteiger	51	8	6	5	2
total	4905	216	95	32	14
%		4.4	1.9	0.65	0.29

4.1 Satisfiability of Formulas with Reduced Bit-Widths

From the 4905 formulas, only 4.4% have a different satisfiability status for any of their reduced version. Moreover, only 1.9% have a different satisfiability status after reducing to 2 bits or more, 0.65% have a different satisfiability status after reducing to 4 bits or more, and only 0.29% have a different satisfiability status after reducing to 8 bits or more. Table 2 shows the numbers of such benchmarks precisely after grouping the formulas to their respective families. For example, the table shows that all decided formulas from the family *2017-Preiner-scholl-smt08*, which contains the largest number of undecided benchmarks, have the same satisfiability status for all bit-widths from 2 to the original bit-width and all decided formulas from the family *2017-Preiner-psyco* have the same satisfiability status for all bit-widths from 4 to the original bit-width. Note that the original bit-width of all benchmarks from these families is 32 bits.

Figure 2 presents these results graphically for all bit-widths between 1 and the original bit-width. The figure shows the results only for formulas with the original bit-width of 32 or 64 bits as the number of benchmarks of other bit-widths is negligible. It can be seen that only under 0.25% of 32bit formulas change their satisfiability status after reducing their bit-width to 6 bits or more.

Figure 3 presents the satisfiability status of each reduction for all 32 formulas that changed the status after reducing the bit-width to 4 bits or more. Although the plot does not show names of the respective formulas due to the available space, the names can be found at the accompanying web page. Note that most of these formulas are unsatisfiable; this is caused by the fact that most of the formulas in the whole benchmark set are unsatisfiable. The plot contains four outstanding groups of formulas:

- Formulas 7–10 (*intersection-example-onelane.proof-node*{*19355,20770,46589,54847*} from *2017-Preiner-keymaera*). These formulas are unsatisfiable for even bit-widths and satisfiable for odd bit-widths because they contain subformulas equivalent to

  $$(c^{[n]} = (x^{[n]} \cdot x^{[n]}) \div_s (2^{[n]} \cdot y^{[n]})) \ \wedge \ (0^{[n]} \leq_s y^{[n]}) \ \wedge \ (y^{[n]} >_s y^{[n]} + c^{[n]}) \ \wedge \ (y^{[n]} \leq_s c^{[n]}).$$

  For $n \geq 2$, this subformula entails the formula

  $$(x^{[n]} \cdot x^{[n]}) \div_s (2^{[n]} \cdot y^{[n]}) \geq_s (2^{n-2})^{[n]},$$

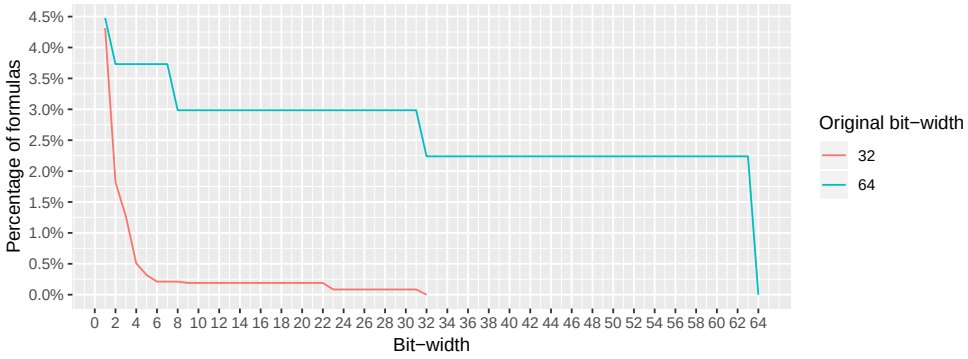

Figure 2: The plot shows the percentage of 32bit and 64bit benchmarks (y-axis) whose satisfiability status is different for the original formula and for any reduced formula with a given (x-axis) or a larger bit-width.

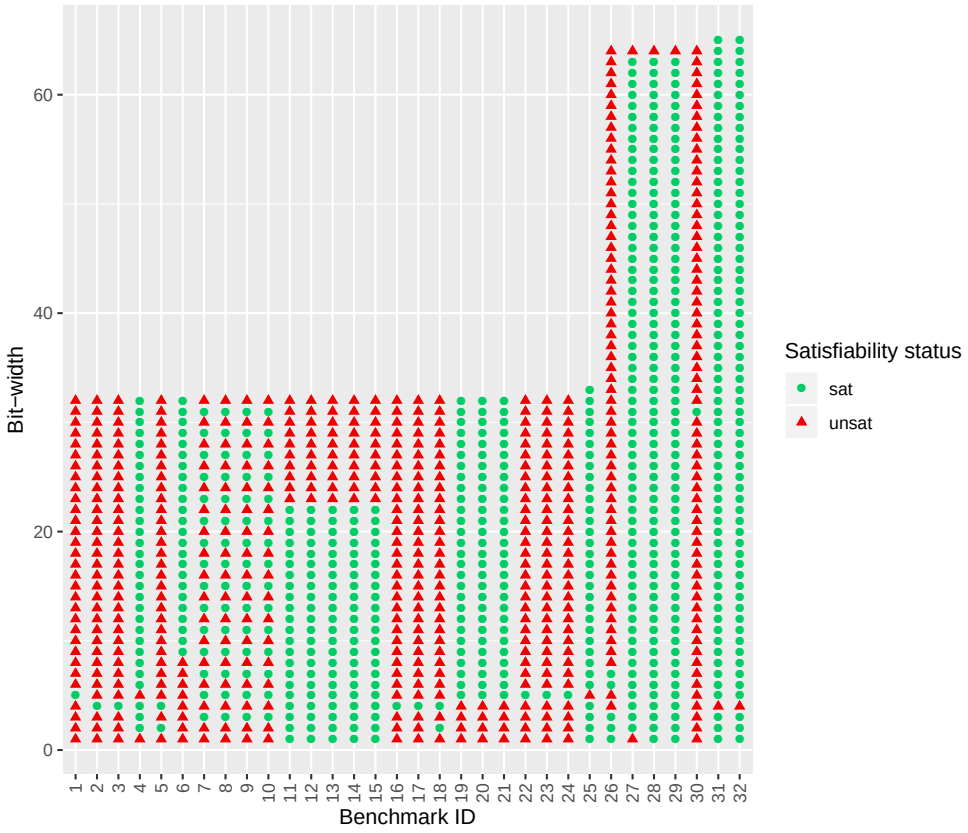

Figure 3: For each of the 32 benchmarks that have a different satisfiability result when reducing their bit-width to 4 bits or more, the plot shows satisfiability statuses for all their reduced versions.

which is unsatisfiable for even bit-widths, but satisfiable for odd bit-widths by setting $x^{[n]} \mapsto (2^{(n-1)/2})^{[n]}$ and $y^{[n]} \mapsto (2^{(n-1)} - 1)^{[n]}$.

- Formulas 11–15 (*jain_7_true_unreach-call_true-no-overflow_i_{215,242,245,262,475}* from *2017-Heizmann-UltimateAutomizer*). Satisfiability of these formulas differs for bit-widths less than 23 and at least 23, because these formulas contain the numeral (presented here in binary)

$$1111\,1111\,1100\,0000\,0000\,0000\,0000\,0000,$$

which gets reduced to 0 after reduction to less then 23 bits.

- Formulas 27–29 (*check_eq_bvashr0_64bit* and *check_ne_{bvlshr0,bvshl0}_64bit* from *2018-Preiner-cav18*). Satisfiability of these formulas is different for the original bit-width of 64 bits and for almost all smaller bit-widths because the formulas contain a subformula similar to $(x^{[64]} <_u 64^{[64]}) \to \psi$, where ψ contains a subterm of the form $t \ll x^{[64]}$.

- Formula 30 (*mmedia_gsm610_gsm6102.c* from *wintersteiger*). Satisfiability of this formula is different for reduction to 31 bits, as it contains a subformula of the form

$$x^{[32]} \leq_s 0100\,0000\,0000\,0000\,0000\,0000\,0000\,0000,$$

in which the second argument has a negative sign-bit precisely for the reduction to 31 bits.

Detailed results together with the raw data files and scripts we used to produce them can be found at: http://fi.muni.cz/~xstrejc/lpar2018/

5 Discussion

The experimental evaluation in the previous section shows that the satisfiability of the vast majority of quantified bit-vector formulas remains the same even after reducing their maximal bit-widths to a very small number of bits. In our opinion, this observation can be helpful in several ways:

- Similarly to the case of the approximation framework of Zeljić et al. for quantifier-free formulas, the performance of SMT solvers for quantified bit-vectors could be improved by first solving a reduced version of the input formula and then checking the result against the original formula. For example, the solver Boolector computes symbolically expressed Skolem functions, which certify satisfiability, and Herbrand functions, which certify unsatisfiability [20]. These functions can be computed from a reduced formula and their validity can be checked against the original formula. More generally, in an SMT solver based on quantifier instantiation such as Boolector, CVC4 [19], or Z3 [22], the set of quantifier instances that are sufficient to decide satisfiability of the reduced formula can be checked against the original formula.

- Because satisfiability of some formulas can be decided even without using the original bit-width, more fine-grained computational complexity of deciding their satisfiability could be identified. Currently, the known results of computational complexity are in term of the size of the input formula, from which the bit-widths are inseparable. In contrast, *parameterized computational complexity* could be examined to show how the complexity depends on various parameters such as the bit-width, the size of the largest constant, the number of used function symbols, number of quantifier alternations etc.

- As a practical use of the previous point, it could be possible for some formulas to compute a bit-width for which the reduced formula is equisatisfiable to the input one. Such a bit-width could be used to decide the satisfiability without using the original bit-width.

6 Conclusions

We have experimentally evaluated the effect of reducing bit-width of variables in quantified bit-vector formulas from SMT-LIB repository and have shown that satisfiability of the vast majority of these formulas remains the same even after reducing bit-widths of variables to a very small number of bits.

References

[1] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `www.SMT-LIB.org`.

[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] Randal E. Bryant, Daniel Kroening, Joël Ouaknine, Sanjit A. Seshia, Ofer Strichman, and Bryan A. Brady. An abstraction-based decision procedure for bit-vector arithmetic. *STTT*, 11(2):95–104, 2009.

[6] Byron Cook, Daniel Kroening, Philipp Rümmer, and Christoph M. Wintersteiger. Ranking function synthesis for bit-vector relations. *Formal Methods in System Design*, 43(1):93–120, 2013.

[7] 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.

[8] Leonardo Mendonça de Moura and Dejan Jovanovic. A model-constructing satisfiability calculus. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, pages 1–12, 2013.

[9] Andreas Fröhlich, Armin Biere, Christoph M. Wintersteiger, and Youssef Hamadi. Stochastic Local Search for Satisfiability Modulo Theories. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1136–1143, 2015.

[10] Sumit Gulwani, Saurabh Srivastava, and Ramarathnam Venkatesan. Constraint-based invariant inference over predicate abstraction. In *Verification, Model Checking, and Abstract Interpretation, 10th International Conference, VMCAI 2009, Savannah, GA, USA, January 18-20, 2009. Proceedings*, pages 120–135, 2009.

[11] Peer Johannsen. Reducing bitvector satisfiability problems to scale down design sizes for RTL property checking. In *Proceedings of the Sixth IEEE International High-Level Design Validation and Test Workshop 2001, Monterey, California, USA, November 7-9, 2001*, pages 123–128, 2001.

[12] Peer Johannsen. *Speeding up hardware verification by automated data path scaling.* PhD thesis, University of Kiel, Germany, 2002.

[13] Martin Jonáš and Jan Strejček. On the Complexity of the Quantified Bit-Vector Arithmetic with Binary Encoding. *Inf. Process. Lett.*, 135:57–61, 2018.

[14] Martin Jonáš and Jan Strejček. Solving Quantified Bit-Vector Formulas Using Binary Decision Diagrams. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 267–283. Springer, 2016.

[15] Gergely Kovásznai, Andreas Fröhlich, and Armin Biere. Complexity of fixed-size bit-vector logics. *Theory Comput. Syst.*, 59(2):323–376, 2016.

[16] Daniel Kroening, Matt Lewis, and Georg Weissenbacher. Under-approximating loops in C programs for fast counterexample detection. In *Computer Aided Verification - 25th International Conference, CAV 2013*, volume 8044 of *LNCS*, pages 381–396. Springer, 2013.

[17] Jan Mrázek, Petr Bauch, Henrich Lauko, and Jiří Barnat. SymDIVINE: Tool for control-explicit data-symbolic state space exploration. In *Model Checking Software - 23rd International Symposium, SPIN 2016, Co-located with ETAPS 2016, Eindhoven, The Netherlands, April 7-8, 2016, Proceedings*, pages 208–213, 2016.

[18] Aina Niemetz, Mathias Preiner, and Armin Biere. Propagation based local search for bit-precise reasoning. *Formal Methods in System Design*, 51(3):608–636, 2017.

[19] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark Barrett, and Cesare Tinelli. Solving quantified bit-vectors using invertibility conditions. 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 II*, pages 236–255, 2018.

[20] Mathias Preiner, Aina Niemetz, and Armin Biere. Counterexample-Guided Model Synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 264–280. Springer, 2017.

[21] Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster. From program verification to program synthesis. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, pages 313–326, 2010.

[22] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. Efficiently solving quantified bit-vector formulas. *Formal Methods in System Design*, 42(1):3–23, 2013.

[23] Aleksandar Zeljic, Peter Backeman, Christoph M. Wintersteiger, and Philipp Rümmer. Exploring Approximations for Floating-Point Arithmetic Using UppSAT. In *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, pages 246–262, 2018.

[24] Aleksandar Zeljic, Christoph M. Wintersteiger, and Philipp Rümmer. Deciding Bit-Vector Formulas with mcSAT. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 249–266, 2016.

[25] Aleksandar Zeljic, Christoph M. Wintersteiger, and Philipp Rümmer. An Approximation Framework for Solvers and Decision Procedures. *J. Autom. Reasoning*, 58(1):127–147, 2017.