Is Bit-Vector Reasoning as Hard as NExpTime in Practice?

(Extended Abstract)

Nachum Dershowitz[1] and Alexander Nadel[2]

[1] School of Computer Science, Tel Aviv University, Ramat Aviv, Israel
nachumd@cs.tau.ac.il
[2] Intel Corporation, P.O. Box 1659, Haifa 31015, Israel
alexander.nadel@intel.com

Abstract

It has been shown that (quantifier-free) bit-vector logic (BV) is NExpTime-complete, on account of the fact that the number of propositional variables in the SAT encoding of a BV formula grows exponentially with the length of the declarations of the bit-vector variables in the input formula. This level of complexity is surprising, given that in practice BV is being used successfully in a wide range of applications. This work points out that the high complexity does not necessary hold in practical applications of BV. We give two examples of easier families of BV problems. First, we demonstrarate that in a BV subset, originating in a recent critical Intel application of BV from clock routing, the number of propositional variables is polynomial in the size of the BV instance – since the maximal value of the instance's bit-vector variables is written out explicitly as a bit-vector constant, thus establishing that clock routing uses an NP-complete subset of BV. Second, we show that for another application, namely, microcode validation, the (logarithm of the) register width should be viewed as a fixed parameter, and thus microcode validation uses a para-NP-complete parameterized version of BV. Similar arguments are likely to apply to a variety of other BV applications to demonstrate that they too use NP-complete subclasses of BV or its para-NP-complete parametrized version.

1 Introduction

Quantifier-free bit-vector (BV) reasoning is being applied in a variety of domains, including software validation [8, 5, 16], hardware validation [10, 19], answer-set programming [15], clock routing [4], automated configuration [14], and others [13]. On the theoretical side, it has been shown that BV is as hard as NExpTime [11] – a somewhat surprising result, given the apparent practical usefulness of bit-vector reasoning.

The following SMT2 [1] formula is used in [11] to illustrate why BV is NExpTime-hard:

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

Written to a file, this formula can be encoded with 138 bytes. However, the number of propositional variables and clauses required to translate this formula to SAT is huge: one needs $2 \cdot 10^6$ variables merely to represent the input variable `x` or `y`, let alone the auxiliary variables created in the process of translating the formula to CNF. It was reported in [11] that Tseitin-encoding of this problem into CNF resulted in a 1GB DIMACS-format formula file, while the same problem using a bit-width of 10 million could no longer be bit-blasted due to integer overflow.

As this example illustrates, the reason for the high complexity of BV reasoning is the fact that the number of propositional variables grows exponentially with the length of the declarations of bit-vector variable widths introduced when declaring the input bit-vector variables. Our main observation is that, nevertheless, various practical applications of BV use an NP-complete subset of BV. We point to two such applications at Intel.

Section 2 shows that a recent critical application of BV in clock routing [4] uses a subclass of BV, where the number of propositional variables is polynomially bounded by the size of the corresponding BV instance, since the maximal value of the instance's bit-vector variables of the only non-Boolean width appearing in the instance is written out explicitly as a bit-vector constant (and, in addition, the overall number of literals in the clauses is polynomial in the number of variables). Hence, that subclass belongs to NP.

Section 3 argues that there is a fixed small parameter for microcode validation problems [8, 7], corresponding to the (logarithm of the) register width in the underlying architecture. Intuitively, this observation means that BV is NP-complete for the practical needs of microcode validation. The observation can be easily formalized using parameterized complexity theory [2, 3, 17]. Specifically, BV, parameterized by the (logarithm of the) variable width, is para-NP-complete. Similar arguments can be applied to other applications of BV in software and hardware validation.

Section 4 reviews related work, while Sect. 5 summarizes our conclusions.

2 Clock Routing

Matching-constrained routing in automatic clock routing is a recent application of BV [4], one that is of critical practical importance.

Automating the routing process is essential for the semiconductor industry to reduce time-to-market and increase productivity. The problem of matching-constrained routing can be formulated as follows: given a set of nets, each net consisting of a driver and a receiver, connect each driver to its receiver, where the delay should be almost the same across nets. This problem can be reduced to bounded-path, the NP-hard problem of finding a simple path from source to target (in a positively weighted undirected graph) whose cost is within a given range $[c_{min} \mathinner{..} c_{max}]$. Furthermore, the bounded-path problem can be naturally encoded in BV. See [4].

While [4] concentrates on developing efficient graph-aware bounded-path algorithms to make the solution scalable, our focus here is on showing that the initial reduction to BV uses an NP-complete subset of the language.

We call an edge or vertex *active* if it appears on the path from source s to target t. It is shown in [4] how one can formulate connectivity constraints guaranteeing that a valid path of an arbitrary cost from s to t is constructed by the solver. The connectivity constraints comprise a purely Boolean expression over Boolean variables representing the activity status of the vertices and the edges. Clearly, translating a Boolean expression to SAT results in a polynomial number of SAT variables and clauses. For the rest of the discussion, we assume that a valid path of an arbitrary cost is guaranteed to be constructed by the solver.

Figure 1 shows the constants, variables and constraints required for ensuring that the cost is bounded and relevant for complexity considerations.

First, consider the constants. A crucial observation for our purposes is that the constant representing the upper bound c_{max} on the cost of the path must be part of the BV encoding of any bounded-path instance. The constant is written out explicitly along with its width $w = \lceil \log_{10} c_{max} \rceil + 1$. (We add an extra bit to the width to take care of overflow in cost propagation, as explained below.) For example, if the given cost range $[c_{min} \mathinner{..} c_{max}]$ is $[1000000 \mathinner{..} 2000000]$, the corresponding SMT2 BV instance will contain the constant `(_ bv2000000 22)`. The BV instance will also contain constants corresponding to the lower bound c_{min} and the edge costs, but c_{max} is the largest of all constants, hence it has maximal width. (Edges with costs greater than c_{max} can be eliminated, since they cannot be part of a bounded path.)

Second, consider the variables. Each edge e is associated with a Boolean variable representing its direction dir_e on the path from s to t. Each vertex v is associated with a BV variable c_v that represents the cost of the path from s to v, if v is active (item 2b in Fig. 1).

The direction and cost variables are used to propagate the cost along the chosen path from s to t using the cost-propagation constraints (items 3a and 3b in Fig. 1). The constructed path is guaranteed to be within bounds at the target, since the target cost c_t is restricted to the range $[c_{min} .. c_{max}]$ (item 4a in Fig. 1).

Finally, the cost of each vertex is bounded by the maximal cost c_{max} (item 4b in Fig. 1). This constraint is valid, since if a path cost reaches $c_{max} + 1$ at any point, the full path to the target could not be within bounds; thus it can be pruned.[1]

The key observation demonstrating that the encoding uses an NP-complete subset of BV is as follows. It is sufficient to set the width of the cost of each vertex v to $w = \lceil \log_{10} c_{max} \rceil + 1$ to be able to accommodate the maximal cost c_{max}, while ensuring that there is no overflow during cost propagation. Thus, translating a cost variable to SAT results in a number of propositional variables polynomial in the number of digits required to write out c_{max} in the BV instance. Hence, the overall number of SAT variables is polynomial in the size of the BV instance. This fact and the fact that the overall number of literals in the resulting SAT clauses is polynomial in the number of SAT variables shows that bounded-path uses an NP-complete subset of BV.

1. Constants
   (a) $BV_{\lceil \log_{10} c_{max} \rceil + 1}\ c_{max}$
2. Cost variables
   (a) Boolean dir_e: the direction of $e \in E$
   (b) $BV_{\lceil \log_{10} c_{max} \rceil + 1}\ c_v$: the cost of the path from s to v
3. Cost-propagation constraints
   (a) $c_s = 0$
   (b) $c_v = c_e + c_u$, for vertex $v \notin \{s, t\}$, where the direction of the path is from u to v via e
4. Cost-bound constraints
   (a) $c_{min} \leq c_t \leq c_{max}$
   (b) $c_v \leq c_{max}$ for each $v \in V$

Figure 1: Translating bounded-path to BV. Constants, variables and constraints, relevant to complexity considerations, are shown.

[1] This constraint does not appear in [4], since it is not necessary for ensuring correctness. We introduce it here, since it allows us to reduce the cost variable width from $\lceil \log_{10} S \rceil + 1$ to $w = \lceil \log_{10} c_{max} \rceil + 1$, where S is the sum of the costs of all the edges in the graph.

3 Microcode Validation

Microcode is a critical component in modern microprocessors, and substantial effort is devoted to verifying its correctness. It has been shown [8, 7] that microcode validation can be efficiently solved with BV reasoning. The translation of the problem to BV is done using an intermediate representation, the Intermediate Representation Language (IRL). This simple language possesses all the features necessary for modeling microcode programs. Programs are translated into IRL by a set of IRL templates that define the translation from microcode instructions into a corresponding sequence of IRL expressions. IRL expressions are constructed using symbolic execution. A variable definition in IRL appears as follows [7]:

```
<declaration> ::= var <variable-list> : BitVector[ <width> ] ;
```

The IRL variable declaration is similar to a variable declaration in the BV language, but in practice the width in the definition above is always replaced by the concrete register width of the underlying computer architecture. Therefore, the number of propositional variables corresponding to a bit-vector variable in a formula is determined by the register width w, where w is a small, fixed value (e.g., 32 or 64) across all the microcode instances for a particular architecture. Thus, intuitively, one could claim that microcode validation uses the NP-complete fixed-width subclass of BV.

This claim can be formalized based on parameterized complexity theory [2, 3, 17]. In this framework, the complexity of a computational problem is measured, not only in terms of the input size (as in classical complexity), but also in terms of a parameter. The central notion of the field is *fixed-parameter tractability*, which refers to solvability in time $f(k) \cdot n^c$, where f is some (possibly exponential) function of the parameter k, c is a constant, and n is the size of the instance with respect to some reasonable encoding. In our context, we need the related notion of the para-NP complexity class, the class of all parameterized decision problems that can be solved in time $f(k) \cdot n^c$ by a *nondeterministic* algorithm [6]. In our case, $k \approx \log_{10} w$ is the length of the vector declarations; f is an exponential function giving the number of bits in a vector; n is the number of vector variables, which is proportional to the size of the formula, excluding the vector declarations; and c is a very small constant. As pointed out in [17], for parameterizations of NP-problems, a para-NP-completeness result is considered to be very negative, but for a problem that is harder than NP, like BV, a para-NP-completeness result may be deemed positive, as it shows that the structure represented by the parameter can be exploited to break the complexity barrier.

Our point is that, as we have argued, microcode validation uses a parameterized version of BV (where $k \approx \log_{10} w$ is the parameter), which belongs to para-NP, since – for each value of k – it can be reduced to SAT, where the number of SAT variables is: (a) exponential in the parameter k, itself logarithmic in w, the architecture-determined variable-width; (b) polynomial in the length of the original formula. The number of clauses in the SAT encoding is polynomial in formula size, since bit-blasting bit-vector operators to SAT results in a polynomial number of clauses for each operator. Note that the width of bit-vector variables is always provided explicitly in the BV language, and applying an operator always results in a bit-vector of the same declared width. Note also that SAT can be trivially reduced to BV.

It follows that BV is para-NP-complete, where the logarithm of the maximal width (which is the register width in the case of microcode validation) is the parameter.

The usage of parameterized complexity in our context of bridging the gap between theoretical performance guarantees and the empirically observed performance of solvers is not surprising, since parameterized complexity has been introduced precisely to bridge the theory-practice gap. See [17] for an excellent survey about parameterized complexity and its applications in constraint satisfaction and reasoning.

4 Related Work

It has been observed in [11] that, if the maximal bit-width can be bounded for a BV family of formulæ, then that family is in NP. There, an infinite set of BV formulæ is defined to be *bit-width bounded* if the maximal bit-width w is bounded by a function $p(n)$, polynomial in the number of terms n in the corresponding formula (where all the bit-widths are provided explicitly in SMT2 formulæ). Any bit-width bounded family of BV formulæ belongs to NP.

The BV family originating in clock routing is *not* bit-width bounded since there is no correlation between the maximal bit-width and the number of terms in the formula. As we have shown in Section 2, that fragment does belong to NP due to a different reason.

On the other hand, any family of formulæ with a *fixed* vector width, like a set of formulæ emerging in microcode validation for to any fixed register width, is trivially bit-width bounded. Let S_k be all BV formulæ whose maximal bit-width is in the range $[10^k\,..\,10^{k+1}-1]$. Parameterizing by k, as we have done in the previous section, partitions BV into the sequence $S_0, S_1, S_2, \ldots$ of NP languages, which is why BV is para-NP.

5 Conclusion

In general, BV is NExpTime-complete, since the number of propositional variables in the SAT encoding of a BV formula grows exponentially with the length of the bit-vector declarations in the input formula. Practical applications of BV are, however, more likely to use an NP subset of the general problem than harder complexity classes.

In particular, we have shown that two practical applications at Intel use either an NP-complete subset or a para-NP-complete parameterized version of BV:

- For the BV fragment used for solving clock routing, the maximal value of its bit-vector variables is written out explicitly as a bit-vector constant. Hence, that fragment is in NP. This should come as no surprise, as the original problem is in NP.
- For microcode validation, the architecture's register width w determines a fixed parameter for the whole class of problems. Hence, microcode validation uses a para-NP-complete characterization of BV, where $k = \log w$ is the parameter.

These results can be generalized to the following characterizations of BV and its subclasses, which can be applied to classify other applications of BV:

- Any BV fragment, such that the maximal value of its bit-vector variables is written out explicitly as a bit-vector constant for every width in the instance, belongs to NP.
- BV itself (not only formulæ for microcode validation), parametrized by the logarithm of the maximal width, is para-NP-complete.

In particular, variable width is a small fixed parameter in other BV applications in software and hardware validation. We plan to study the complexity of other applications of BV in the future.

Recent works [11, 9, 12] have studied the complexity of various subclasses of BV. The results herein suggest that to close the gap between the theoretical high-complexity of BV and its apparent real-life applicability, it makes sense to study the parameterized complexity of BV and its subclasses.

Acknowledgments

We would like to thank Paul Inbar for editing the paper and the referees for their suggestions.

References

[1] Clark Barrett, Aaron Stump, and Cesare Tinelli. The satisfiability modulo theories library (SMT-LIB). `http://www.SMT-LIB.org`, 2010.

[2] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[3] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[4] Amit Erez and Alexander Nadel. Finding bounded path in graph using SMT for automatic clock routing. In *Proc. 27th International Conference on Computer Aided Verification (CAV), San Francisco, CA*, 2015. To appear; draft at `http://goo.gl/xP603S`.

[5] Stephan Falke, Florian Merz, and Carsten Sinz. LLBMC: Improved bounded model checking of C programs using LLVM. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 623–626. Springer, 2013.

[6] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer-Verlag, Secaucus, NJ, 2006.

[7] Anders Franzén. *Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT*. Ph.d. dissertation, University of Trento, 2010.

[8] Anders Franzén, Alessandro Cimatti, Alexander Nadel, Roberto Sebastiani, and Jonathan Shalev. Applying SMT in symbolic execution of microcode. In *Proc. of Formal Methods in Computer-Aided Design (FMCAD)*, pages 121–128. IEEE, 2010.

[9] Andreas Fröhlich, Gergely Kovásznai, and Armin Biere. More on the complexity of quantifier-free fixed-size bit-vector logics with binary encoding. In Andrei A. Bulatov and Arseny M. Shur, editors, *Computer Science – Theory and Applications, Proc. 8th International Computer Science Symposium in Russia (CSR), Ekaterinburg, Russia*, volume 7913 of *Lecture Notes in Computer Science*, pages 378–390. Springer, June 2013.

[10] Michael Katelman and José Meseguer. `vlogsl`: A strategy language for simulation-based verification of hardware. In *Hardware and Software: Verification and Testing*, pages 129–145. Springer, 2011.

[11] Gergely Kovásznai, Andreas Fröhlich, and Armin Biere. On the complexity of fixed-size bit-vector logics with binary encoded bit-width. In Pascal Fontaine and Amit Goel, editors, *Proc. 10th International Workshop on Satisfiability Modulo Theories (SMT), Manchester, UK*, volume 20 of *EPiC Series*, pages 44–56. EasyChair, June 2012.

[12] Gergely Kovásznai, Helmut Veith, Andreas Fröhlich, and Armin Biere. On the complexity of symbolic verification and decision problems in bit-vector logic. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Proc. 39th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part II, Budapest, Hungary*, volume 8635 of *Lecture Notes in Computer Science*, pages 481–492. Springer, August 2014.

[13] Filip Marić and Predrag Janičić. URBiVA: Uniform reduction to bit-vector arithmetic. In *Automated Reasoning*, pages 346–352. Springer, 2010.

[14] Raphaël Michel, Arnaud Hubaux, Vijay Ganesh, and Patrick Heymans. An SMT-based approach to automated configuration. In Pascal Fontaine and Amit Goel, editors, *Proc. 10th International Workshop on Satisfiability Modulo Theories (SMT)*, pages 107–117, 2012.

[15] Mai Nguyen, Tomi Janhunen, and Ilkka Niemelä. Translating answer-set programs into bit-vector logic. In Tompits et al. [18], pages 95–113.

[16] Anthony Romano and Dawson Engler. Expression reduction from programs in a symbolic binary executor. In Ezio Bartocci and C. R. Ramakrishnan, editors, *Proc. 20th International Symposium on Model Checking Software (SPIN), Stony Brook, NY*, volume 7976 of *Lecture Notes in Computer Science*, pages 301–319. Springer, July 2013.

[17] Stefan Szeider. The parameterized complexity of constraint satisfaction and reasoning. In Tompits et al. [18], pages 27–37.

[18] Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors. *Revised Selected Papers of the 19th International Conference on Applications of Declarative Programming and Knowledge Management (INAP) and the 25th Workshop on Logic Programming (WLP), Vienna, Austria*, volume 7773 of *Lecture Notes in Computer Science*. Springer, September 2013.

[19] Robert Wille, Daniel Große, Finn Haedicke, and Rolf Drechsler. SMT-based stimuli generation in the SystemC verification library. In *Advances in Design Methods from Modeling Languages for Embedded Systems and SoCs*, pages 227–244. Springer, 2010.