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

(Extended Abstract)

Nachum Dershowitz and Alexander Nadel

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 (QF_BV) is NExpTime-complete, on account of the fact that the number of propositional variables in the SAT encoding of a QF_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 QF_BV is being used successfully in a wide range of applications. This work points out that the high complexity does not necessarily hold in practical applications of QF_BV. We give two examples of easier families of QF_BV problems. First, we demonstrate that, in a recent critical Intel application of QF_BV in clock routing, the number of propositional variables grows polynomially with the length of the variable declarations, thus establishing that clock routing uses an NP-complete subset of QF_BV. Second, we show that in another application, namely, microcode validation, register width should be viewed as a fixed parameter (more precisely, the logarithm of the width is the parameter), and thus microcode validation uses a para-NP-complete parameterized version of QF_BV. We believe that the same arguments can be applied to a variety of other industrial QF_BV applications to demonstrate that they too use an NP-complete or para-NP-complete subclass of QF_BV.

1 Introduction

Quantifier-free bit-vector (QF_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 QF_BV is as hard as NExpTime [11] – a surprising result, given the apparent practical usefulness of bit-vector reasoning.

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

```smt2
(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 \(2 \cdot 10^9\) 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 CNF in DIMACS format that was 1GB in size, 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 QF\_BV reasoning is the fact that the number of variables grows exponentially with the length of the declarations of variable widths introduced when declaring the input variables. Our main observation is that this is not the case in various practical applications of QF\_BV. We point to two such applications at Intel.

Section 2 shows that the number of variables grows polynomially with the input length (and, in addition, the overall number of literals in the clauses is polynomial in the number of variables) in a recent critical application of QF\_BV in clock routing [4].

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 QF\_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, QF\_BV, parameterized by the (logarithm of the) variable width, is para-NP-complete. Similar arguments can be applied to other applications of QF\_BV in software and hardware validation.

Section 4 summarizes some conclusions.

## 2 Clock Routing

Matching-constrained routing in automatic clock routing is a recent application of QF\_BV [4], one which 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 the nets. This problem can be reduced to bounded-path, that is, the NP-hard problem of finding a simple path, whose cost is bounded by a given range, from the source s to the target t in an undirected positively weighted graph. It is further shown in [4] that the bounded-path problem can be naturally encoded in QF\_BV.

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

We call an edge/vertex active if it appears on the path from s to t, and inactive otherwise. Figure 1 shows the variables required for the encoding. First, each edge e and vertex v is associated with a Boolean variable $a_e$ and $a_v$, respectively, to represent whether the edge or the vertex, respectively, is active (items 1a and 1b in Fig. 1). Second, each edge e is associated with a Boolean variable representing its direction $\text{dir}_e$ on the path from s to t. Finally, each vertex v is associated with a BV variable $c_v$ which represents the cost of the path from the source s to v if v is active (item 2a in Fig. 1). The width of $c_v$ for each v is set to $\lceil \log_2 S \rceil + 1$, where S is the sum of the costs of all the edges in the graph. This is necessary in order to accommodate the cost of all the edges without overflow.

The set of constraints that use the variables in Fig. 1 to encode bounded-path into QF\_BV appears in [4]. In our context, it is sufficient to know that the overall number of literals in the resulting clauses is polynomial in the length of the variable declarations.

The key observation demonstrating that the encoding uses an NP-complete subset of QF\_BV is as follows. The only set of non-Boolean variables in the encoding is $c_v$, but its width grows polynomially with the size of the problem, since the number of bits in all the $c_v$ variables
is proportional to the number of bits required to represent the costs in the original problem. Hence, bounded-path requires an NP-complete subset of QF\_BV.

1. Connectivity variables
   (a) Boolean \( a_e \): \( a_e \) is 1 iff \( e \in E \) is active.
   (b) Boolean \( a_v \): \( a_v \) is 1 iff \( v \in V \) is active.

2. Cost variables
   (a) BV \( c_v \): the cost of the path from \( s \) to \( v \)
   (b) Boolean \( \text{dir}_e \): the direction of \( e \in E \)

Figure 1: Translating bounded-path to QF\_BV: the variables.

3 Microcode Validation

Microcode is a critical component in modern microprocessors, and substantial effort is devoted to verifying its correctness [8, 7]. It has been shown [8, 7] that microcode validation can be efficiently solved with QF\_BV reasoning. The translation of the problem to QF\_BV is done using an intermediate representation, the Intermediate Representation Language (IRL). This is a simple language with all the features necessary for modeling microcode programs. Microcode 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. As mentioned in [7], a variable definition in IRL appears as follows:

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

The IRL variable declaration is similar to a variable declaration in the QF\_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 QF\_BV.

The intuitive claim can be formalized based on parameterized complexity theory [2, 3, 17]. In this theory, 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 QF_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 QF_BV (where $k \approx \log_{10} w$ is the parameter), which belongs to para-NP, since 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, for each $k$. 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 QF_BV language, and applying an operator always results in a bit-vector of the same declared width. It follows that QF_BV is para-NP-complete, since SAT can be trivially reduced to QF_BV.

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 Conclusion

In general, QF_BV is NExpTime-complete, since the number of propositional variables in the SAT encoding of a QF_BV formula grows exponentially with the length of the bit-vector declarations in the input formula. Practical applications of QF_BV are, however, more likely to use an NP-complete subset of the general problem, because those problems that can be solved in practice are more likely to be in NP, rather than NExpTime.

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 QF_BV:

- For clock routing, the number of SAT variables grows only polynomially in the length of the input formula. Hence, the 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 variant of QF_BV, where $k = \log w$ is the parameter.

Note that our arguments are applicable to other applications of QF_BV reasoning. In particular, the variable width is a small fixed parameter in other QF_BV applications in software and hardware validation. We plan to study the complexity of other applications of QF_BV in the future.

Recent works study the complexity of various subclasses of QF_BV [11, 9, 12]. Our results imply that to close the gap between the theoretical high-complexity results and the apparent real-life applicability of QF_BV, it makes sense to study the parameterized complexity of QF_BV and its subclasses.

## Acknowledgments

The authors would like to thank Paul Inbar for editing the paper.
Is Bit-Vector Reasoning as Hard as NExpTime

Dershowitz and Nadel

References
