Superposition Reasoning about Quantified Bitvector Formulas

David Damestani
TU Wien

Laura Kovács
TU Wien and Chalmers

Martin Suda
Czech Technical University

***Abstract*—We describe recent extensions to the first-order theorem prover Vampire for proving theorems in the theory of fixed-sized bitvectors, possibly with quantifiers. Details are given on extending both the parser of Vampire as well as the theory reasoning framework of Vampire. We present our experimental results by evaluating and comparing our approach to SMT solvers. Our experiments report also on a few examples that can be solved only by our work.**

I. Introduction

Applications of program analysis and verification require proving program properties, such as invariants and safety assertions, in the combination of various theories such as data structures, integers, lists, and arrays. These properties are typically expressed as first-order formulas with both theories and quantifiers.

There are a number of attempts at proving such properties. SMT-based approaches use E-matching heuristics for extending quantifier-free theory reasoning with efficient handling of quantifiers [1], [3], [10]. On the other hand, first-order theorem provers combine complete methods for quantified reasoning with incomplete but sound theory axiomatizations [5], [7], in addition to combination with SMT-based proving [11], [13]. Despite such recent advancements, proving properties in full first-order theories remains one of the main challenges in automated reasoning.

In this paper, we approach this challenge and consider first-order formulas in the theory of quantified fixed-sized bitvectors (QBV). While quantifier-free bitvectors have sophisticated decision procedures implemented in most SMT solvers, including Z3 [3], CVC4 [1], and Boolector [10], proving bitvector formulas with quantifiers became an active research topic only recently, e.g. in [9], by extending SMT reasoning over quantifier-free bitvectors with heuristic-driven strategies for quantifiers. In this paper, we go in the opposite direction: we use superposition reasoning in first-order theorem proving and extend it with sound axioms of quantified bitvectors. Our workhorse is the theorem prover Vampire [7].

This paper discusses our work on proving quantified bitvector formulas in the setting of Vampire. As such, our work focuses on proving quantified bitvector theorems, and not on generating models for satisfiable non-theorems. We first extended Vampire to parse quantified bitvector formulas, written in the SMT-Lib input syntax (Section IV). Next, we added built-in theory support for quantified bitvector symbols and axioms to Vampire (Section V). We evaluated our work and compared it with SMT-based approaches and report on our experiments (Section VI). While still at a very early stage of development, Vampire already managed to prove bitvector theorems no other (SMT) solver was able to. The present paper aims to (i) overview implementation details of our approach, (ii) summarize our extensions and improvements to Vampire for theory reasoning and (iii) report on our experimental findings.

This research was supported by the Austrian FWF grant S11409-N23 (RiSE/SHiNE), the ERC Consolidator grant AI4REASON 649043, ERC Starting Grant SYMCAR 639270, the ERC Proof of Concept Grant SYMELS 842066, the Swedish Wallenberg Academy Fellowship TheProSE, and the OMAA grant 101öu8.

II. Motivating Example

We motivate our work with the following formula:

$$\forall x_{[n]}.\exists y_{[n]}.\forall z_{[n]}.\exists u_{[n]}(x+y)=(z\times u), \tag{1}$$

where n is a positive integer and $x_{[n]}, y_{[n]}, z_{[n]}, u_{[n]}$ respectively denote variables of bitvectors of size n. For simplicity, bitvector addition over bitvectors of size n is simply written as $+$, instead of $+_{[n]}$. The operator $\times$ denotes here the multiplication operation over bitvectors of size n. Note that arguments as well as the result of the binary bitvector operations $+$ and $\times$ are bitvectors of the same size (that is, n). We also note that for a concrete fixed value of n the formula (1) is a first-order QBV formula with alternations of quantifiers.

With our QBV reasoning extension to Vampire, it proves (1) for almost arbitrary values of n, such as $n=4$ or $n=32$. That is, Vampire proves validity, e.g., of

$$\forall x_{[4]}.\exists y_{[4]}.\forall z_{[4]}.\exists u_{[4]}(x+y)=(z\times u). \tag{2}$$

with the help of the following bitvector axiom schemas:

$$\begin{array}{l}\forall x_{[n]}.\forall y_{[n]}(x+y=y+x)\\ \forall x_{[n]}.\forall y_{[n]}.\forall z_{[n]}((x+y\neq z)\vee(y=z+(-x)))\\ \forall x_{[n]}.(x\times 0_{[n]}=0_{[n]})\end{array} \tag{3}$$

More precisely, Vampire instantiates the schemas (3) and generates concrete axioms by instantiating n by the appropriate value; the obtained axioms are then used together with superposition reasoning for proving respective instance of the QBV formula (1). This is however not the case with SMT solvers, for example when using Z3 (version 4.8.4)

and CVC4 (version 1.7). Both Z3 and CVC4 quickly find a proof for the QBV formula (1) with $n = 4$, but cannot prove the instance for $n = 32$ within a 60 seconds time limit. Our QBV formula (1) shows the advantage of our approach when compared to the bit-blasting techniques of SMT solving.

III. What exactly is meant by QBV?

In this work, we extend Vampire with the support for the SMT-Lib theory of fixed-sized bitvectors BV.[1] In this theory, bitvector sizes are always explicitly specified and fixed as concrete integers in the input problem. To stress that we focus on problems combining theory reasoning *with quantifiers*, for which the superposition technology could have an advantage compared to standard SMT solvers, we informally add the qualifier Q. As usual, theories can be combined and so this added support for bitvector reasoning allows Vampire to tackle problems from various logics containing BV, such as AUFBVDTLIA used in our experiments (see Section VI). For more details on the theory of fixed-sized bitvectors and its complexity, we refer the reader to [8].

IV. Parsing QBV Formulas in Vampire

We now detail the main steps of our work for extending superposition reasoning in Vampire with built-in theory support for QBV. As a first step, we solved the practical challenge of extending the Vampire parser for parsing QBV formulas, as follows.

Input syntax: QBV formulas are passed to Vampire as input problems written in the SMT-Lib syntax [2]. For example, the QBV formula of (2) can be passed to Vampire using the SMT-Lib notation:

```
(assert
  (forall ((x (_ BitVec 4)))
    (exists ((y (_ BitVec 4)))
      (forall ((z  (_ BitVec 4)))
        (exists ((u  (_ BitVec 4)))
          (= (bvadd x y) (bvmul z u)))))))
```

In addition, the Vampire input of the QBV formula (2) also sets the SMT-Lib logic to the theory `BV` of fixed size bitvectors, that is: `(set-logic BV)`.

Note that quantifier-free bitvector formulas are special cases of QBV formulas; as such, Vampire supports quantifier-free and quantified bitvector formulas, written in the SMT-Lib syntax. All operations of the bitvector fragment of SMT-Lib are supported.

Bitvector sort: To parse QBV formulas, we extended Vampire with a new built-in sort representing the bitvector sort. Internally, Vampire implements sorts as structured and unstructured sorts: for example, integers are unstructured, whereas arrays are structured sorts. We extended Vampire with a new structured sort of bitvectors, denoted as `BitVectorSort`. The sort `BitVectorSort` stores the size of the bitvector that it is representing as an unsigned integer. In the above SMT-Lib representation of the QBV formula (2), the sort `BitVec 4` of bitvectors of size 4 is used from `BitVectorSort`.

Bitvector symbols: Parsing QBV formulas requires parsing bitvector function and predicate symbols, such as bitvector constants, the addition operation or the comparison operator. We extended Vampire with a new wrapper class `BitVectorConstantType` for bitvector constants, for example the constant 512_{10} denoting the decimal number 512 as a constant bitvector of size 10. We use the boolean type (one byte) to represent a single bit of a bitvector constant; as such, the `BitVectorConstantType` class stores an array of booleans, with its least significant bit at position 0 and each boolean representing a single bit constant. Corresponding to the SMT-Lib standard, we extended Vampire to support both hexadecimal and decimal notation of bitvector constants. As the size of a bitvector in the input problem is fixed, say bitvector of size 32, Vampire can parse and store a bitvector of any size, given enough memory. However, when converting from a decimal to a binary representation, the result is modulo the size of the bitvector, yielding possible truncations. For example, attempting to store 512_{10} in a bitvector of size 2 will result in truncation.

We extended the built-in theory symbols of Vampire with bitvector function and predicate symbols, such as addition and the multiplication operation. Internally, Vampire considers polymorphic functions (such as `bvadd`) with different sized bitvector arguments as different symbols. Bitvector operations (predicate/function) are essentially parametrized by the size of their bitvector arguments.

For our running example, parsing the afore given SMT-Lib format of the QBV formula (2) yields interpreting $+$ as binary addition `bvadd` whereas $\times$ is interpreted as the multiplication operator `bvmul`, both operations over bitvectors of size 4. Further, $0_{[4]}$ is mapped to the bitvector constant `#b0000` of size 4.

V. Proving QBV Formulas in Vampire

After parsing QBV input formulas, Vampire first clausifies these formulas using its clausifier [12]: the main steps of clausification involve expanding equivalences, naming subformulas and Skolemisation, with a number of optimizations, for example, on mini-scoping and (anti-)prenexing. We refer to [12] for details on clausification. After clausification in Vampire, the resulting clausal normal form of the QBV input formula is subjected to generic preprocessing and further passed to the main saturation loop. For proving QBV (clausal) formulas, we extended Vampire as follows.

[1] See http://smtlib.cs.uiowa.edu/theories-FixedSizeBitVectors.shtml.

Table I
EXPERIMENTAL RESULTS USING VAMPIRE WITH DIFFERENT OPTIONS.

AVATAR	Theory Axioms	Theory Instantiation	Benchmarks solved	BV	UFBV	AUFBVDTLIA	Uniques
on	on	all	4555	4182 (2)	7	366 (7)	9
off	on	all	4552	4266 (2)	9	317 (7)	9
on	on	off	4433	4074 (2)	7	352	2
off	on	off	4407	4092 (2)	7	308	2
off	off	all	3534	3488	6	40	0
on	off	all	3527	3483	4	40	0
off	off	off	3465	3423	2	40	0
on	off	off	3464	3422	2	40	0

Evaluation of QBV formulas: We added a `BitVectorEvaluator` class to Vampire for evaluating and simplifying bitvector terms in analogy to how it was already done for other theories [11]. For example, we evaluate ground bitvector terms and simplify multiplications by the zero bitvector constant. By using `BitVectorEvaluator`, Vampire for example simplifies `(bvadd #b1110 #b0001)` into the bitvector constant `#b1111` of size 4.

AVATAR and theory instantiation: For proving QBV formulas, we extended the SAT/SMT-based AVATAR architecture [11], [15] of Vampire with bitvector support. To this end, we can optionally translate Vampire bitvector symbols to Z3 symbols and pass the ground parts of the clauses to the SMT solver. Additionally, the interface to Z3 enables us to invoke the theory instantiation inference [13] for bitvectors, which we experimented with.

We note that in our experiments, we only used SAT-based AVATAR. That is, we did not rely on Z3 to reason about ground formulas generated by AVATAR; this way, theory reasoning about QBV formulas in Vampire's AVATAR framework was performed by relying on bitvector axioms (see below) in combination with superposition reasoning.

Bitvector axioms: Once a particular finite set of bitvector sizes is fixed (note that only finitely many are mentioned in any concrete input problem) the corresponding first-order theory of bitvectors is in principle finitely axiomatizable, because the mentioned domains are finite and each bitvector operation can be fully specified. However, the exponential blowup inherent in this idea is obviously prohibitive.

In our work, we therefore considered a sound, but incomplete set of axioms. More precisely, we extended Vampire with built-in theory reasoning for QBV by adding a sound, but incomplete set of bitvector axioms to the theory reasoning engine of Vampire. All together, we added 101 QBV axioms as built-in theory axioms of Vampire. These axioms were chosen experimentally: we analyzed the bitvector operations used in the our set of benchmarks and formulated properties axiomatizing properties of these operations. The such resulting set of 101 QBV axioms allowed us to solve most of the benchmarks we considered; a subset of 2018-Preiver-cav (https://clc-gitlab.cs.uiowa.edu/SMT-LIB-benchmarks/BV/tree/master/2018-Preiner-cav18). Yet, our experiments (see Section VI) demonstrate that our set of 101 QBV axioms are not sufficient: there are still quite many problems in our current benchmark set that could not be proven by Vampire as further QBV theory axioms would be needed during proof search. In general, understanding which QBV axioms should be used and chosen is a challenging task on its own, which we leave for future work.

Each bitvector axiom we consider is a quantified QBV formula arising from an axiom schema by supplying concrete integer values for bitvector sizes. Most axioms require only one bitvector size n to be specified. In some cases, the axiom holds only when $n > 1$. Sometimes more than one numerical parameters needs to be supplied satisfying additional constraints (for example, the bitvector operation `concat` takes bitvectors of size m and n and outputs a bitvector of size $m + n$).

These theory axiom schemas are instantiated by Vampire by exactly those concrete bitvector sizes that occur in the input problem; for example, in the case of the QBV formula (2), the theory axioms of bitvectors of size 4 are used. During parsing, Vampire stores a list of bitvector function and predicate symbols occurring in the input problem and uses this list later to load relevant built-in theory axioms (i.e. theory axioms using bitvector symbols from the input problem are selected). The selected theory axioms are automatically added to the input and used within Vampire's saturation-based framework for establishing validity.

For our motivating QBV example (1), Vampire loads the following set of bitvector axioms (where $1_{[n]}$ denotes a bitvector having the least significant bit set to one and all others set to zero):

$$\forall x_{[n]}.\forall y_{[n]}.(x + y = y + x)$$
$$\forall x_{[n]}.\forall y_{[n]}.\forall z_{[n]}((x + y \neq z) \vee (y = z + (-x)))$$
$$\forall x_{[n]}.((x + 1_{[n]}) \neq (x + (-1_{[n]}))) \iff n > 1$$
$$\forall x_{[n]}.(x + 1_{[n]} \neq x)$$
$$\forall x_{[n]}.\forall y_{[n]}.(x \times y = y \times x)$$
$$\forall x_{[n]}.(x \times 1_{[n]} = x)$$
$$\forall x_{[n]}.(x \times 0_{[n]} = 0_{[n]})$$

These axioms are added by Vampire due to the presence of bitvector addition and multiplication in (1). In particular the

Table II
NUMBER OF QBV BENCHMARKS SOLVED BY DIFFERENT SOLVERS.

Solver	Total (SAT/UNS)	Uniques	BV	UFBV	AUFBVDTLIA
Z3	6145 (759/5386)	190	5452 (12)	165 (45)	528 (133)
CVC4	5880 (419/5461)	18	5383 (13)	75 (1)	422 (4)
Q3B	5563 (594/4969)	40	5492 (33)	42 (7)	29 (0)
Boolector	5475 (606/4869)	8	5432 (8)	43 (0)	0 (0)
Vampire	4555 (0/4555)	9	4182 (2)	7 (0)	366 (7)

first four axioms are added due to the presence of addition and the remaining three due to the presence of multiplication. Our axiom selection method is quite naive, selecting all relevant axioms and adding these axioms to the formula to be proved. For proving the QBV formula instance (2) of (1) for bitvectors of size 4, Vampire instantiates the above set of theory axioms by setting n to 4 and uses these axiom instances for proving (2) by superposition. For proving (2), Vampire actually needs only a subset of the axioms above. To be precise, only the axiom instances of (3) from Section II are used by Vampire for proving the QBV formula (2).

Axioms for operations such as bitvector concatenation are treated no different than other operations. Axiom schemata which hold for any size bitvector operation are instantiated with the concrete size of the bitvector arguments.

Consider for example:

$$\forall x_{[n]}.\forall y_{[m]}.\forall k_{[n]}.\forall z_{[m]}.((y : x \leq_u z : k) \rightarrow (y \leq_u z))$$

where ":" represents bitvector concatenation, $\leq_u$ is the unsigned less-than-or-equal operator, and n and m are positive integers. Note this axiom holds no matter the size of the bitvector arguments.

VI. EXPERIMENTAL RESULTS

Our QBV extension of Vampire required about 6000 lines of C++ code on top of Vampire and is available at:

https://github.com/vprover/vampire/tree/bitvector_theory

For experiments, we used all the 7660 quantified bitvector (QBV) problems of the SMT-Lib repository[2], which comprise the following benchmark sets:

- BV, with examples in the theory of fixed size bitvectors (5751 problems);
- UFBV, containing benchmarks in theory of uninterpreted functions and bitvectors (200 problems);
- AUFBVDTLIA, with problems in the theory of arrays, uninterpreted functions, bitvectors, data types and linear integer arithmetic (1709 problems).

This set of benchmarks contains both satisfiable and unsatisfiable problems, making our benchmark set not friendly for first-order provers, such as Vampire, designed for proving theorems (i.e. validity) instead of satisfiability checking. Collecting more QBV examples corresponding to QBV theorems is an interesting line of future work.

[2]The 2018-05-20 release of SMT-LIB.

We evaluated QBV reasoning in Vampire with the following Vampire options:

- options `-tha on/off` (with `on` as default), for using the built-in bitvector theory axioms of Vampire. Our bitvector axioms implemented now in Vampire were mainly generated from our attempts to solve the benchmarks of [9].
- options `-av on/off`, for using the AVATAR framework of Vampire for SAT-based clause splitting over splittable clauses during proof search. By default, AVATAR is used in Vampire by relying internally on the Minisat SAT solver [4]. While AVATAR in Vampire can be used also with SMT solving instead of SAT solving, in our experiments we only invoked SAT-based AVATAR and hence did not use SMT solving, in particular Z3, for ground bitvector reasoning. This way, unless theory instantiation was also used, bitvector reasoning was only performed using our QBV extension to Vampire.
- options `-thi all/off` (with `off` as default), for using additional inference rules for theory instantiations, as described in [13]. Theory instantiation in Vampire relies on Z3 to generate clause instances and models.

We ran our experiments on the StarExec cluster [14] and set a time limit of 60 seconds per benchmark.

Table I summarizes our results on evaluating QBV reasoning in Vampire on our 7660 benchmarks, by using the (combination of the) above Vampire options as listed in Columns 1–3. The total numbers of problems solved by Vampire are listed in Column 4, whereas Columns 5–7 report the number of problems solved per each benchmark set. For each benchmark set, we also give in parenthesis the number of problems that have been uniquely solved by Vampire by using its considered combination of options. Our experiments show using theory axioms in combination with the theory instantiation rule brings the best performance.

We also compared our work against the SMT solvers Z3, CVC4, Boolector and Q3B [6], on the same set of benchmarks[3]. The experiments were again run on StarExec using each solvers' default configurations from the SMT-COMP 2018 competition. Table II shows the total number of benchmarks solved using a particular solver, by also specifying the number of problems proved to be sat and unsat, respectively (SAT/UNS). We also show the number

[3]We remark that Boolector does not support the AUFBVDTLIA theory.

of problems solved uniquely by a solver. Our experiments show that SMT solvers were more efficient than Vampire; yet, we already managed to solve 9 benchmarks no other (SMT) solver was able to solve. We believe these initial experiments, similarly to Section II, demonstrate that QBV reasoning in first-order provers is worth to be improved, pushing further the state-of-the-art in QBV reasoning.

VII. Conclusions

We described new extensions to the first-order prover Vampire for proving QBV formulas with both theories and bitvector symbols. By implementing simplification rules and bitvector axioms in Vampire, our approach solves already a few problems that existing SMT solvers fail to. Understanding which theory axioms are relevant for improving built-in bitvector support in Vampire is an interesting line of future work, with the potential of allowing Vampire, and, in general, first-order provers, to solve new properties.

References

[1] C. Barrett, H. Barbosa, M. Brain, D. Ibeling, T. King, P. Meng, A. Niemetz, A. Nötzli, M. Preiner, A. Reynolds, and C. Tinelli. CVC4 at the SMT competition 2018. 2018.

[2] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017.

[3] L. De Moura and N. Bjørner. Z3: An efficient SMT Solver. In *TACAS*, pages 337–340, 2008.

[4] N. Eén and N. Sörensson. An Extensible SAT-solver. In *SAT*, pages 502–518, 2003.

[5] B. Gleiss, L. Kovács, and S. Robillard. Loop Analysis by Quantification over Iterations. In *LPAR-22*, pages 381–399, 2018.

[6] M. Jonás and J. Strejcek. Abstraction of Bit-Vector Operations for BDD-Based SMT Solvers. In *ICTAC*, pages 273–291, 2018.

[7] L. Kovács and A. Voronkov. First-Order Theorem Proving and Vampire. In *CAV*, pages 1–35, 2013.

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

[9] A. Niemetz, M. Preiner, A. Reynolds, C. Barrett, and C. Tinelli. Solving Quantified Bit-Vectors Using Invertibility Conditions. In *CAV*, pages 236–255, 2018.

[10] A. Niemetz, M. Preiner, C. Wolf, and A. Biere. Btor2 , BtorMC and Boolector 3.0. In *CAV*, pages 587–595, 2018.

[11] G. Reger, N. Bjørner, M. Suda, and A. Voronkov. AVATAR Modulo Theories. In *GCAI*, pages 39–52, 2016.

[12] G. Reger, M. Suda, and A. Voronkov. New Techniques in Clausal Form Generation. In *GCAI*, pages 11–23, 2016.

[13] G. Reger, M. Suda, and A. Voronkov. Unification with Abstraction and Theory Instantiation in Saturation-Based Reasoning. In *TACAS*, pages 3–22, 2018.

[14] A. Stump, G. Sutcliffe, and C. Tinelli. StarExec: A Cross-Community Infrastructure for Logic Solving. In *IJCAR*, pages 367–373, 2014.

[15] A. Voronkov. AVATAR: The Architecture for First-Order Theorem Provers. In *CAV*, pages 696–710, 2014.