# Q3B: An Efficient BDD-based SMT Solver for Quantified Bit-Vectors

Martin Jonáš(✉) and Jan Strejček

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

**Abstract.** We present the first stable release of our tool Q3B for deciding satisfiability of quantified bit-vector formulas. Unlike other state-of-the-art solvers for this problem, Q3B is based on translation of a formula to a BDD that represents models of the formula. The tool also employs advanced formula simplifications and approximations by effective bit-width reduction and by abstraction of bit-vector operations. The paper focuses on the architecture and implementation aspects of the tool, and provides a brief experimental comparison with its competitors.

## 1 Introduction

Advances in solving formula *satisfiability modulo theories* (SMT) achieved during the last few decades enabled significant progress and practical applications in the area of automated analysis, testing, and verification of various systems. In the case of software and hardware systems, the most relevant theory is the *theory of fixed-sized bit-vectors*, as these systems work with inputs expressed as bit-vectors (i.e., sequences of bits) and perform bitwise and arithmetic operations on bit-vectors. The quantifier-free fragment of this theory is supported by many general-purpose SMT solvers, such as CVC4 [1], MathSAT [7], Yices [10], or Z3 [9] and also by several dedicated solvers, such as Boolector [21] or STP [12]. However, there are some use-cases where quantifier-free formulas are not natural or expressive enough. For example, formulas containing quantifiers arise naturally when expressing loop invariants, ranking functions, loop summaries, or when checking equivalence of two symbolically described sets of states [8,13,17,18,24]. In the following, we focus on SMT solvers for *quantified* bit-vector formulas. In particular, this paper describes the state-of-the-art SMT solver Q3B including its implementation and the inner workings.

Solving of quantified bit-vector formulas was first supported by Z3 in 2013 [25] and for a limited set of *exists/forall* formulas with only a single quantifier alternation by Yices in 2015 [11]. Both of these solvers decide quantified formulas by *quantifier instantiation*, in which universally quantified variables in the Skolemized formula are repeatedly instantiated by ground terms until the resulting quantifier-free formula is unsatisfiable or a model of the original formula is found.

This work has been supported by Czech Science Foundation, grant GA18-02177S.

© The Author(s) 2019
I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11562, pp. 64–73, 2019.
https://doi.org/10.1007/978-3-030-25543-5_4

In 2016, we proposed a different approach for solving quantified bit-vector formulas: by using binary decision diagrams (BDDs) and approximations [14]. For evaluation of this approach, we implemented an experimental SMT solver called Q3B, which outperformed both Z3 and Yices. Next solver that was able to solve quantified bit-vector formulas was Boolector in 2017, using also an approach based on quantifier instantiation [22]. Unlike Z3, in which the universally quantified variables are instantiated only by constants or subterms of the original formula, Boolector uses a counterexample-guided synthesis approach, in which a suitable ground term for instantiation is synthesized based on the defined grammar. Thanks to this, Boolector was able to outperform Q3B and Z3 on certain classes of formulas. More recently, in 2018, support of quantified bit-vector formulas has also been implemented into CVC4 [20]. The approach of CVC4 is also based on quantifier instantiation, but instead of synthesizing terms given by the grammar as Boolector, CVC4 uses predetermined rules based on invertibility conditions, which directly give terms that can prune many spurious models without using potentially expensive counterexample-guided synthesis. The authors of CVC4 have shown that this approach outperforms Z3, CVC4, and the original Q3B. However, Q3B has been substantially improved since the original experimental version. In 2017, we extended it with simplifications of quantified bit-vector formulas using unconstrained variables [15]. Further, in 2018, we added the experimental implementation of abstractions of bit-vector operations [16]. With these techniques, Q3B is able to decide more formulas than Z3, Boolector, and CVC4. Besides the theoretical improvements, Q3B was also improved in terms of stability, ease of use, technical parts of the implementation, and compliance with the SMT-LIB standard. This tool paper presents the result of these improvements: Q3B 1.0, the first stable version of Q3B.

We briefly summarize the SMT solving approach of Q3B. As in most of modern SMT solvers, the input formula is first simplified using satisfiability-preserving transformations that may reduce the size and complexity of the formula. The simplified formula is then converted to a binary decision diagram (BDD) that represents all assignments satisfying the formula, i.e., the *models* of the formula. If the BDD represents at least one model, we say that the BDD is *satisfiable* and it implies satisfiability of the formula. If the BDD represents the empty set of models, we say that it is *unsatisfiable* and so is the formula. Unfortunately, there are formulas for which the corresponding BDD (or some of the intermediate BDDs that appear during its computation) is necessarily exponential in the number of bits in the formula. For example, this is the case for formulas that contain multiplication of two bit-vector variables [5]. To be able to deal with such formulas, Q3B computes in parallel also BDDs underapproximating and overapproximating the original set of models, i.e., BDDs representing subsets and supersets of the original set of models, respectively. The approximating BDDs may be much smaller in size than the precise BDD, especially if the approximation is very rough. Still, they can be used to decide satisfiability of the original formula. If an overapproximating BDD is unsatisfiable, the original formula is also unsatisfiable. If the overapproximating BDD is satisfiable, we take one of its models, i.e., an assignment to the top-level existential variables of the formula, and check whether it is a model of the original formula. If the answer is positive, the original formula is satisfiable. In the other case, we build a more precise overapproximating BDD. Underapproximating BDDs are utilized analogously. The only difference is that for unsatisfiable underapproximating BDD, we check the validity of a countermodel, i.e., an assignment to the top-level universal variables that makes the formula unsatisfiable. The approach is depicted in Fig. 1.

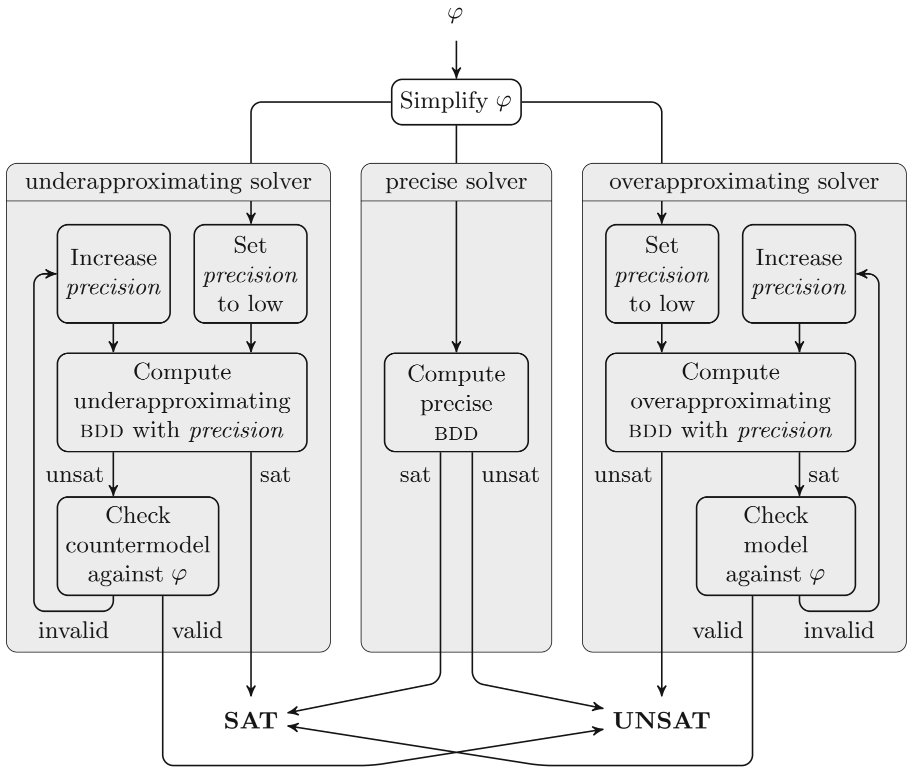

**Fig. 1.** High-level overview of the SMT solving approach used by Q3B. The three shaded areas are executed in parallel and the first result is returned.

Q3B currently supports two ways of computing the approximating BDDs from the input formula. First of these are *variable bit-width approximations* in which the *effective bit-width* of some variables is reduced. In other words, some of the variables are represented by fewer bits and the rest of the bits is set to zero bits, one bits, or the sign bit of the reduced variable. This approach was originally used by the SMT solvers UCLID [6] and Boolector [21]. Q3B extends this approach to quantified formulas: if bit-widths of only existentially quantified variables are reduced, the resulting BDD is underapproximating; if bit-widths of only universally quantified variables are reduced, the resulting BDD is overapproximating. The second way to obtain an approximation is *bit-vector operation abstraction* [16], during which the individual bit-vector operations may not compute all bits of the result, but produce some *do-not-know bits* if the resulting BDDs would exceed a given number of nodes. An underapproximating BDD then represents assignments that satisfy the formula for all possible values of these do-not-know bits. Analogously, an overapproximating BDD represents all assignments that satisfy the formula for some value of the do-not-know bits. Q3B also supports a combination of these two methods, in which both the effective bit-with of variables is reduced and the limit on the size of BDDs is imposed. During an approximation refinement, either the effective bit-width or the size limit is increased, based on the detected cause of the imprecision.

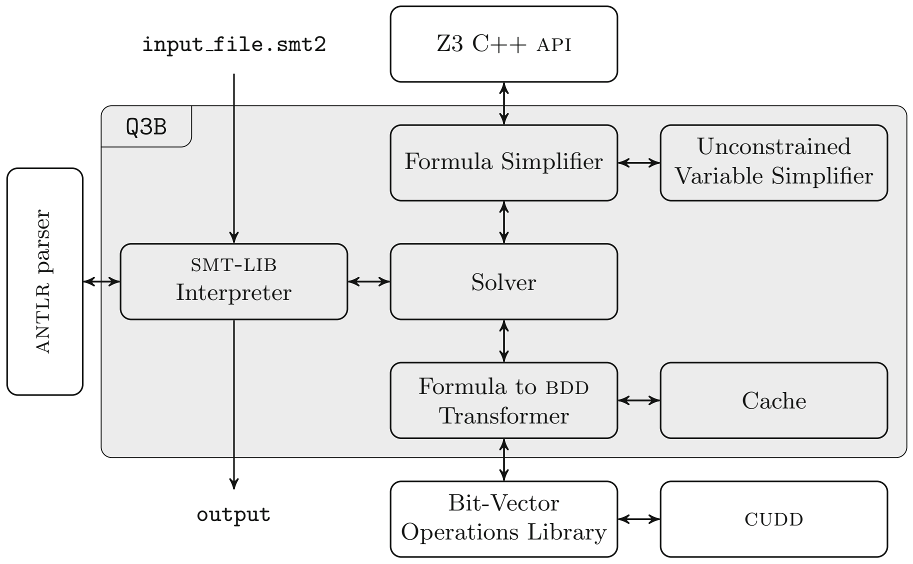

**Fig. 2.** Architecture of Q3B. Components in the shaded box are parts of Q3B, the other components are external.

## 2 Architecture

This section describes the internal architecture of Q3B. The overall structure including internal and external components and the interactions between them is depicted in Fig. 2. We explain the purpose of the internal components:

**SMT-LIB Interpreter** (implemented in `SMTLIBInterpreter.cpp`) reads the input file in the SMT-LIB format [3], which is the standard input format for SMT solvers. The interpreter executes all the commands from the file. In particular, it maintains the assertion stack and the options set by the user, calls solver when `check-sat` command is issued, and queries `Solver` if the user requires the model with the command `get-model`.

**Formula Simplifier** (implemented in `FormulaSimplifier.cpp`) provides interface for all applied formula simplifications, in particular miniscoping, conversion to negation normal form, pure literal elimination, equality propagation, constructive equality resolution (CER) [14], destructive equality resolution (DER) [25], simple theory-related rewriting, and simplifications using unconstrained variables. Most of these simplifications are implemented directly in this component; only CER, DER, and majority of the theory-related rewritings are performed by calling Z3 API and simplifications using unconstrained variables are implemented in a separate component of Q3B. The simplifier also converts top-level existential variables to uninterpreted constants, so their values are also included in a model. Some simplifications that could change models of the formula are disabled if the user enables model generation, i.e., sets `:produce-models` to `true`.

**Unconstrained Variable Simplifier** (implemented in `UnconstrainedVariableSimplifier.cpp`) provides simplifications of formulas that contain unconstrained variables, i.e., variables that occur only once in the formula. Besides previously published unconstrained variable simplifications [15], which were present in the previous versions of Q3B, this component now also provides new *goal-directed* simplifications of formulas with unconstrained variables. In these simplifications, we aim to determine whether a subterm containing an unconstrained variable should be minimized, maximized, sign minimized, or sign maximized in order to satisfy the formula. If the subterm should be minimized and contains an unconstrained variable, the term is replaced by a simpler term that gives the minimal result that can be achieved by any value of the unconstrained variable. Similarly for maximization, sign minimization, and sign maximization.

**Solver** (implemented in `Solver.cpp`) is the central component of our tool. It calls formula simplifier and then creates three threads for the precise solver, the underapproximating solver, and the overapproximating solver. It also controls the approximation refinement loops of the approximating solvers. Finally, it returns the result of the fastest thread and stores the respective model, if the result was `sat`.

**Formula to BDD Transformer** (implemented in the file `ExprToBDDTransformer.cpp`) performs the actual conversion of a formula to a BDD. Each subterm of the input formula is converted to a vector of BDDs (if the subterm's sort is a bit-vector of width $n$ then the constructed vector contains $n$ BDDs, each BDD represents one bit of the subterm). Further, each subformula of the input formula is converted to a BDD. These conversions proceed by a straightforward bottom-up recursion on the formula syntax tree. The transformer component calls an external library to compute the effect of logical and bit-vector operations on BDDs and vectors of BDDs, respectively. Besides the precise conversion, the transformer can also construct overapproximating and underapproximating BDDs. Precision of approximations depends on parameters set by the solver component.

**Cache** (implemented as a part of `ExprToBDDTransformer.cpp`) maintains for each converted subformula and subterm the corresponding BDD or a vector of BDDs, respectively. Each of the three solvers has its own cache. When an approximating solver increases precision of the approximation, entries of its cache that can be affected by the precision change are invalidated. All the caches are internally implemented by hash-tables.

## 3 Implementation

Q3B is implemented in C++17, is open-source and available under MIT license on GitHub: https://github.com/martinjonas/Q3B. The project development process includes continuous integration and automatic regression tests.

Q3B relies on several external libraries and tools. For representation and manipulation with BDDs, Q3B uses the open-source library CUDD 3.0 [23]. Since CUDD does not support bit-vector operations, we use the library by Peter Navrátil [19] that implements bit-vector operations on top of CUDD. The algorithms in this library are inspired by the ones in the BDD library BuDDy[1] and they provide a decent performance. Nevertheless, we have further improved its performance by several modifications. In particular, we added a specific code for handling expensive operations like bit-vector multiplication and division when arguments contain constant BDDs. This for example considerably speeds up multiplication whenever one argument contains many constant zero bits, which is a frequent case when we use the variable bit-width approximation fixing some bits to zero. Further, we have fixed few incorrectly implemented bit-vector operations in the original library. Finally, we have extended the library with the support for do-not-know bits in inputs of the bit-vector operations and we have implemented abstract versions of arithmetic operations that can produce do-not-know bits when the result exceeds a given number of BDD nodes.

For parsing the input formulas in SMT-LIB format, Q3B uses ANTLR parser generated from the grammar[2] for SMT-LIB 2.6 [2]. We have modified the grammar to correctly handle bit-vector numerals and to support `push` and `pop` commands without numerical argument. The parser allows Q3B to support all bit-vector operations and almost all SMT-LIB commands except `get-assertions`, `get-assignment`, `get-proof`, `get-unsat-assumptions`, `get-unsat-core`, and all the commands that work with algebraic data-types. This is in sharp contrast with the previous experimental versions of Q3B, which only collected all the assertions from the input file and performed the satisfiability check regardless of the rest of the commands and of the presence of the `check-sat` command. The reason for this was that the older versions parsed the input file using the Z3 C++ API, which can provide only the list of assertions, not the rest of the SMT-LIB script. Thanks to the new parser, Q3B 1.0 can also provide the user

[1] https://sourceforge.net/projects/buddy/.
[2] https://github.com/julianthome/smtlibv2-grammar.

with a model of a satisfiable formula after calling `get-model`; this important aspect of other SMT solvers was completely missing in the previous versions.

On the other hand, C++ API of the solver Z3 is still used for internal representation of parsed formulas. The Z3 C++ API is also used to perform manipulations with formulas, such as substitution of values for variables, and some of the formula simplifications. Note that these are the only uses of Z3 API in Q3B during solving the formula; no actual SMT- or SAT-solving capabilities of Z3 are used during the solving process.

Some classes of Q3B, in particular `Solver`, `FormulaSimplifier`, and `UnconstrainedVariableSimplifier`, expose a public C++ API that can be used by external tools for SMT solving or just performing formula simplifications. For example, `Solver` exposes method `Solve(formula, approximationType)`, which can be used to decide satisfiability by the precise solver, the underapproximating solver, or the overapproximating solver. `Solver` also exposes the method `SolveParallel(formula)`, which simplifies the input formula and runs all three of these solvers in parallel and returns the first result as depicted in Fig. 1.

## 4 Experimental Evaluation

We have evaluated the performance of QB3 1.0 and compared it to the latest versions of SMT solvers Boolector (v3.0), CVC4 (v1.6), and Z3 (v4.8.4). All tools were used with their default settings except for CVC4, where we used the same settings as in the paper that introduces quantified bit-vector solving in CVC4 [20], since they give better results than the default CVC4 settings. As the benchmark set, we have used all 5751 quantified bit-vector formulas from the SMT-LIB repository. The benchmarks are divided into 8 distinct families of formulas. We have executed each solver on each benchmark with CPU time limit 20 min and RAM limit of 8 GiB. All the experiments were performed in a Ubuntu 16.04 virtual machine within a computer equipped with Intel(R) Core(TM) i7-8700 CPU @ 3.20 GHz CPU and 32 GiB of RAM. For reliable benchmarking we employed BenchExec [4], a tool that allocates specified resources for a program execution and precisely measures their usage. All scripts used for running benchmarks and processing their results, together with detailed descriptions and some additional results not presented in the paper, are available online[3].

Table 1 shows the numbers of benchmarks in each benchmark family solved by the individual solvers. Q3B is able to solve the most benchmarks in benchmark families *2017-Preiner-scholl-smt08*, *2017-Preiner-tptp*, *2017-Preiner-UltimateAutomizer*, *2018-Preiner-cav18*, and *wintersteiger*, and it is competitive in the remaining families. In total, Q3B also solves more formulas than each of the other solvers: 116 more than Boolector, 83 more than CVC4, and 139 more than Z3. Although the numbers of solved formulas for the solvers seem fairly similar, the cross-comparison in Table 2 shows that the differences among the individual solvers are actually larger. For each other solver, there are at least

[3] https://github.com/martinjonas/q3b-artifact.

**Table 1.** For each solver and benchmark family, the table shows the number of benchmarks from the given family solved by the given solver. The column *Total* shows the total number of benchmarks in the given family. The last line provides the total CPU times for the benchmarks solved by all four solvers.

| Family | Total | Boolector | CVC4 | Q3B | Z3 |
|---|---|---|---|---|---|
| 2017-Preiner-keymaera | 4035 | 4022 | 3998 | 4009 | **4031** |
| 2017-Preiner-psyco | 194 | 193 | 190 | 182 | **194** |
| 2017-Preiner-scholl-smt08 | 374 | 312 | 248 | **319** | 272 |
| 2017-Preiner-tptp | 73 | 69 | **73** | **73** | **73** |
| 2017-Preiner-UltimateAutomizer | 153 | 152 | 151 | **153** | **153** |
| 20170501-Heizmann-UltimateAutomizer | 131 | 30 | **128** | 124 | 32 |
| 2018-Preiner-cav18 | 600 | 553 | **565** | **565** | 553 |
| wintersteiger | 191 | 163 | 174 | **185** | 163 |
| Total | 5751 | 5494 | 5527 | **5610** | 5471 |
| CPU time [s] | | 7794 | 5877 | 19853 | **4055** |

**Table 2.** For all pairs of the solvers, the table shows the number of benchmarks that were solved by the solver in the corresponding row, but not by the solver in the corresponding column. The column *Uniquely solved* shows the number of benchmarks that were solved only by the given solver.

| | Boolector | CVC4 | Q3B | Z3 | Uniquely solved |
|---|---|---|---|---|---|
| Boolector | 0 | 123 | 69 | 78 | 8 |
| CVC4 | 156 | 0 | 60 | 171 | 6 |
| Q3B | 185 | 143 | 0 | 208 | 25 |
| Z3 | 55 | 115 | 69 | 0 | 6 |

143 benchmarks that can be solved by Q3B but not by the other solver. We think this shows the importance of developing an SMT solver based on BDDs and approximations besides the solvers based on quantifier instantiation.

## 5 Conclusions and Future Work

We have described the architecture and inner workings of the first stable version of the state-of-the-art SMT solver Q3B. Experimental evaluation on all quantified bit-vector formulas from SMT-LIB repository shows that this solver slightly outperforms other state-of-the-art solvers for such formulas.

As future work, we would like to drop the dependency on the Z3 API: namely to implement our own representation of formulas and reimplement all the simplifications currently outsourced to Z3 API directly in Q3B. We also plan to extend some simplifications with an additional bookkeeping needed to construct a model of the original formula. With these extensions, all simplifications could be used even if the user wants to get a model of the formula. We would also like to implement production of unsatisfiable cores since they are also valuable for software verification.

## References

1. Barrett, C., et al.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_14
2. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa (2017). www.SMT-LIB.org
3. CBarrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard: version 2.0. In: Gupta, A., Kroening, D. (eds.) Proceedings of the 8th International Workshop on Satisfiability Modulo Theories, Edinburgh, UK (2010)
4. Beyer, D., Löwe, S., Wendler, P.: Benchmarking and resource measurement. In: Fischer, B., Geldenhuys, J. (eds.) SPIN 2015. LNCS, vol. 9232, pp. 160–178. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23404-5_12
5. Bryant, R.E.: On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. IEEE Trans. Comput. **40**(2), 205–213 (1991)
6. Bryant, R.E., Kroening, D., Ouaknine, J., Seshia, S.A., Strichman, O., Brady, B.A.: An abstraction-based decision procedure for bit-vector arithmetic. STTT **11**(2), 95–104 (2009)
7. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_7
8. Cook, B., Kroening, D., Rümmer, P., Wintersteiger, C.M.: Ranking function synthesis for bit-vector relations. Form. Methods Syst. Des. **43**(1), 93–120 (2013)
9. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
10. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_49
11. Dutertre, B.: Solving exists/forall problems with Yices. In: Workshop on satisfiability Modulo Theories (2015)
12. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 519–531. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_52
13. Gulwani, S., Srivastava, S., Venkatesan, R.: Constraint-based invariant inference over predicate abstraction. In: Jones, N.D., Müller-Olm, M. (eds.) VMCAI 2009. LNCS, vol. 5403, pp. 120–135. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-93900-9_13
14. Jonáš, M., Strejček, J.: Solving quantified bit-vector formulas using binary decision diagrams. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 267–283. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_17
15. Jonáš, M., Strejček, J.: On simplification of formulas with unconstrained variables and quantifiers. In: Gaspers, S., Walsh, T. (eds.) SAT 2017. LNCS, vol. 10491, pp. 364–379. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66263-3_23

16. Jonáš, M., Strejček, J.: Abstraction of bit-vector operations for BDD-based SMT solvers. In: Fischer, B., Uustalu, T. (eds.) ICTAC 2018. LNCS, vol. 11187, pp. 273–291. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02508-3_15
17. Kroening, D., Lewis, M., Weissenbacher, G.: Under-approximating loops in C programs for fast counterexample detection. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 381–396. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_26
18. Mrázek, J., Bauch, P., Lauko, H., Barnat, J.: SymDIVINE: tool for control-explicit data-symbolic state space exploration. In: Bošnački, D., Wijs, A. (eds.) SPIN 2016. LNCS, vol. 9641, pp. 208–213. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32582-8_14
19. Navrátil, P.: Adding support for bit-vectors to BDD libraries CUDD and Sylvan. Bachelor's thesis, Masaryk University, Faculty of Informatics, Brno (2018)
20. Niemetz, A., Preiner, M., Reynolds, A., Barrett, C., Tinelli, C.: Solving quantified bit-vectors using invertibility conditions. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10982, pp. 236–255. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96142-2_16
21. Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2, BtorMC and Boolector 3.0. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 587–595. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_32
22. Preiner, M., Niemetz, A., Biere, A.: Counterexample-guided model synthesis. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 264–280. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_15
23. Somenzi, F.: CUDD: CU Decision Diagram Package Release 3.0.0. University of Colorado at Boulder (2015)
24. Srivastava, S., Gulwani, S., Foster, J.S.: From program verification to program synthesis. In: Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, 17–23 January 2010, pp. 313–326 (2010)
25. Wintersteiger, C.M., Hamadi, Y., de Moura, L.M.: Efficiently solving quantified bit-vector formulas. Form. Methods Syst. Des. **42**(1), 3–23 (2013)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.