Beaver: Engineering an Efficient SMT Solver for Bit-Vector Arithmetic

Susmit Jha, Rhishikesh Limaye, and Sanjit A. Seshia

EECS Department, UC Berkeley
{jha,rhishi,sseshia}@eecs.berkeley.edu

Abstract. We present the key ideas in the design and implementation of Beaver, an SMT solver for quantifier-free finite-precision bit-vector logic (QF_BV). Beaver uses an eager approach, encoding the original SMT problem into a Boolean satisfiability (SAT) problem using a series of word-level and bit-level transformations. In this paper, we describe the most effective transformations, such as propagating constants and equalities at the word-level, and using and-inverter graph rewriting techniques at the bit-level. We highlight implementation details of these transformations that distinguishes Beaver from other solvers. We present an experimental analysis of the effectiveness of Beaver's techniques on both hardware and software benchmarks with a selection of back-end SAT solvers.
Beaver is an open-source tool implemented in Ocaml, usable with any back-end SAT engine, and has a well-documented extensible code base that can be used to experiment with new algorithms and techniques.

1 Introduction

Decision procedures for quantifier-free fragments of first-order theories, also known as satisfiability modulo theories (SMT) solvers, find widespread use in hardware and software verification. Of the many first-order theories for which SMT solvers are available, one of the most useful is the theory of finite-precision bit-vector arithmetic, abbreviated as QF_BV [14]. This theory is useful for reasoning about low-level system descriptions in languages such as C and Verilog which use finite-precision integer arithmetic and bit-wise operations on bit-vectors. Recently, there has been a resurgence of work on new QF_BV SMT solvers such as BAT [10], Boolector [3], MathSAT [4], Spear [9], STP [8], UCLID [5] and Z3 [6].

In this article, we describe the design and implementation of BEAVER, a new open-source SMT solver for QF_BV that placed third in SMTCOMP'08. The novelty in Beaver is in its application-driven engineering of a *small* set of simplification methods that yield high performance. This set is: online forward/backward constant and equality propagation using event queues, offline optimization of Boolean circuit templates for operators, and the use of and-inverter graph (AIG) as back-end to perform problem specific bit-level simplifications. Additionally, we have done a systematic study of different Boolean encoding techniques for non-linear operations.

The goal in creating BEAVER was to engineer an efficient and extensible SMT solver around a small core of word-level and bit-level simplification techniques. One aim in writing this article is to succinctly describe the techniques we employed and evaluate them on the SMT-LIB benchmarks. We believe this paper could be useful to both users and designers of solvers. For example, our experiments suggest that online equality propagation is critical to software benchmarks while Boolean rewrites are needed for hardware benchmarks. For developers of SMT and SAT solvers, we present a comparison of Beaver with different SAT solvers as back-end. Our main observation is that the

only available non-clausal SAT solver NFLSAT [11] performs significantly better than other SAT solvers.

We do not attempt to make a comprehensive survey and comparison of solvers here; the interested reader can find a survey of current and past bit-vector solvers in recent articles (e.g. [5]), our technical report [17], and the SMTCOMP'08 results [13]. Our focus is on improving the understanding of what makes a good SMT solver for QF_BV, and on identifying features of solvers and benchmarks that could be of interest to the SAT, SMT, and user community.

2 Approach

BEAVER is a satisfiability solver for formulas in the quantifier-free fragment of the theory of *bit-vector arithmetic* (QF_BV). For lack of space, we omit a detailed presentation of the logic and instead direct the reader to the SMT-LIB website [14] for the detailed syntax and semantics.

BEAVER operates by performing a series of rewrites and simplifications that transform the starting bit-vector arithmetic formula into a Boolean circuit and then into a Boolean satisfiability (SAT) problem. This approach is termed the *eager* approach to SMT solving [1]. A major design decision was to be able to use any off-the-shelf SAT solver so as to benefit from the continuing improvements in SAT solving. BEAVER can generate models for satisfiable formulas.

The transformations employed by BEAVER are guided by the following observations about characteristics of formulas generated in verification and testing.

- *Software verification and testing:* Formulas generated during program analysis are typically queries about the feasibility of individual program paths. They tend to be conjunctions of atomic constraints generated from an intermediate representation such as static single assignment (SSA). There are often multiple variables in the formula representing values of the same program variable. Linear constraints abound, and they tend to be mostly equalities.
  Thus, a major characteristic of software benchmarks tends to be the presence of many redundant variables, a simple Boolean structure comprising mostly of conjunctions, and an abundance of equalities amongst the atomic constraints.
- *Hardware verification:* SMT formulas generated in hardware verification arise in bounded model checking (BMC), equivalence checking, and simulation checking. These formulas tend to have a complicated Boolean structure, with several alternations of conjunctions and disjunctions. Many of the variables are Boolean. However, there are often several syntactically different sub-formulas which are nevertheless logically equivalent. These arise from structural repetitions in the formula, such as across multiple time-frames in BMC, or in the intrinsic similarities between two slightly different copies of a circuit being checked for equivalence. Word-level and bit-level rewrite rules are crucial to simplifying the formula and identifying equivalences. Thus, efficient solving of formulas from hardware verification requires word-level and bit-level simplification techniques.
- *Non-linear operations:* Non-linear operations such as multiplication and division are known to be difficult for SAT solvers. While many formulas arising in verification and testing do not contain these operators (or in a manner that makes the problem hard), every SMT solver needs efficient techniques to deal with hard non-linear constraints when they do arise.

How BEAVER works: BEAVER performs a sequence of simplifications starting from the original bit-vector formula F_{bv} and ending in a Boolean formula F_{bool} which is

handed off to an external SAT engine. Currently BEAVER supports both clausal and non-clausal SAT solvers.

We briefly sketch the transformations performed by BEAVER below. (The first three bit-vector transformations are not necessarily performed in the sequence listed below). They transform F_{bv} to an intermediate Boolean formula F'_{bool}. Bit-level simplifications are then performed on F'_{bool} to yield the final formula F_{bool}.

- *Event-driven constraint propagation:* BEAVER uses an online event-driven approach to propagate constants and equality constraints through the formula in order to simplify it. In particular, a simple form of constraint that appears in many software benchmarks is an equality that uses a fresh variable to name an expression, often of constant value. Both backward (from root of the formula to its leaves) and forward constraint propagation are performed. The event-driven approach, similar to event-driven simulation, allows the simplifications to be performed as a succession of rewrite events. Potential applications of a constant/constraint propagation rewrite are put in a queue and execution of each rewrite rule can potentially generate more avenues of rewriting which are also queued. The rewriting continues till the queue is empty and no further constant/constraint propagation is possible. This propagation is in addition to preliminary simplifications using structural hashing.
- *Bit-vector rewrite rules:* BEAVER also uses a small set of bit-vector rewrite rules to simplify the formula. These interact with the above step by creating new opportunities for constraint propagation. For the current SMTCOMP benchmark suite, an example of a very useful rewrite is the removal of redundant extractions from bit-vectors that enables methods such as constraint propagation to further simplify the formula.
- *Non-linear operations:* In BEAVER, we have experimented with a range of techniques to translate non-linear arithmetic operations such as multiplication, division, and remainder into SAT. These include: (1) Using *magic numbers* when one of the arguments is a constant [19]; (2) decomposing a large bit-width operation into a set of smaller bit-width operations by computing residues modulo a set of primes and using the Chinese remainder theorem; and (3) bit-blasting using standard arithmetic circuits for multiplication by expressing $a \div b$ as q where $a = q * b + r \wedge r < b$ and $*$ and $+$ must not overflow.
  Of the three approaches, our experience has been that the third approach performs the best on most of the current SMT-LIB benchmarks. Thus, while all three options are available, we set the third approach to be the default option.
- *Boolean simplifications — offline and online:* After performing the above bit-vector simplifications, the resulting formula F'_{bv} is encoded into a Boolean formula F'_{bool} that is represented as an And-Inverter Graph (AIG). The translation is straightforward except for one novelty: we pre-synthesize optimized netlists from Verilog for different arithmetic operators for various bit-widths. These pre-synthesized *template circuits* are stored as AIGs and can be optimized offline using logic synthesis tools such as the ABC logic synthesis engine [16]. We explore the impact of logic optimization on template circuits in Section 3. When the solver runs on a specific benchmark, it can further perform bit-level rewrites on the AIG obtained after all operators have been instantiated with template circuits. The resulting formula is F_{bool}.
  F_{bool} can be solved using clausal (CNF-based) or non-clausal SAT engines. We have experimented with different CNF generation options from the ABC engine and also with a non-clausal SAT solver. Our results are presented in Section 3.

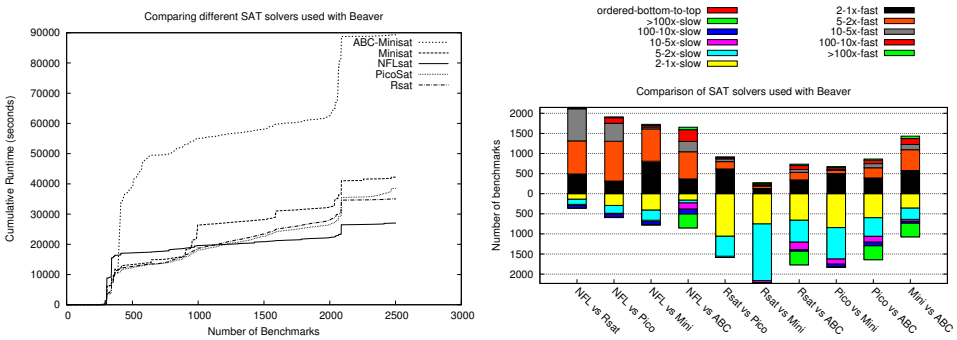
A more detailed description of the above transformations with examples is available from the BEAVER website listed below. BEAVER is implemented in OCaml (linked with the ABC library) and uses an external SAT engine. A source-code release of BEAVER has been publicly available since July 2008; the latest version can be downloaded from the URL `http://uclid.eecs.berkeley.edu/beaver/`.

3 Experimental Evaluation

Setup Experiments were conducted with a selection of benchmarks from all families in the QF_BV section of the publicly available SMT-LIB benchmark suite [15]. A cluster of machines was used for experiments, with each workstation having an Intel(R) Xeon(TM) 3.00 GHz CPU, 3 GB RAM, and running 64-bit Linux 2.6.18. We enforced a memory limit of 1 GB on all runs and the timeout was set to 1 hour. A complete set of experimental data is available at the BEAVER website.

Impact of SAT solvers First, we evaluate the impact of the choice of SAT solver on performance. We used the default setting of bit-vector transformations and Tseitin's encoding to CNF. Five SAT solvers were compared: Picosat v. 846 compiled in optimized mode [2], Minisat 2.0 [7], Rsat [12], ABC's version of Minisat 1.14 [16], and NFLSAT [11]. Of the five, NFLSAT is the only non-clausal SAT solver.

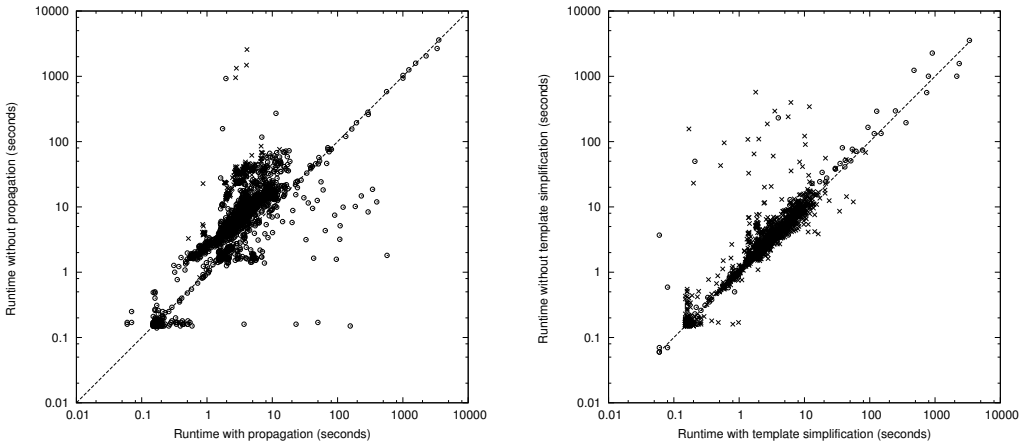
Figure 1 summarizes the comparison of the five SAT solvers. From Fig. 1(a), we can see that NFLSAT exhibits the smallest aggregate run-time over all benchmarks. Fig. 1(b) compares solvers in terms of the degree of speed-up obtained. We see that NFLSAT is the best performing SAT engine, achieving speed-ups by large factors, indicating that the use of a non-clausal SAT solver might be beneficial for bit-vector formulas.



(a) (b)

Fig. 1. Comparing five SAT solvers. (a) Plots of cumulative run-time over all benchmarks; (b) Comparing speed-ups. Each stacked bar compares two SAT solvers by counting the number of benchmarks with various ranges of speed-ups. Different colors represent different range of speed-ups (100x,20x,5x,2x,1x). The portion of a bar above 0 represents the number of benchmarks where the first solver is faster than the second solver and the part below represents benchmarks on which the first is slower than the second.

Detailed pair-wise comparisons of SAT engines have also been performed; however, for lack of space, we omit these here and refer the reader to our technical report [17]. We



(a) (b)

Fig. 2. (a) Impact of constant/equality constraint propagation; (b) Impact of template optimization. All scatterplots use log-scale axes. NFLSAT was the back-end SAT solver. For (a), the path feasibility queries are marked with $\times$; for (b), satisfiable benchmarks are marked with $\times$ and unsatisfiable benchmarks with $\circ$.

have observed that the relative performance of SAT solvers can depend heavily on the benchmark family. For example, in comparing NFLSAT and Minisat 2.0, we found a set of benchmarks where the NFLSAT run-time is fairly stable between 1 and 20 sec. while the Minisat run-time goes from 10 to above 1000 sec. These benchmarks are almost all from the `spear` family, where formulas are verification conditions with rich Boolean structure and several kinds of arithmetic operations. On the other hand, there is also a set of benchmarks on which Minisat's run-time stays under 0.1 sec while NFLSAT's run-time ranges from 1 to 100 seconds. These benchmarks were from mostly from the `catchconv` family, with a few `crafted` benchmarks. Since the `catchconv` family comprises path feasibility queries that are conjunctions of relatively simple constraints (mostly equalities), it appears that the Boolean structure and form of atomic constraints could determine whether non-clausal solving is beneficial.

Impact of word-level simplifications Figure 2(a) shows the impact of constraint propagation. Overall, constraint propagation helps, as can be seen by the big cluster of benchmarks above the diagonal. On a closer examination of the data, we noticed that *all* path feasibility queries (`catchconv`) benchmarks (indicated in plot using $\times$) benefited greatly from the use of constraint propagation, with a typical speed-up for the solver of 5-100X on these formulas.

Impact of circuit synthesis techniques We evaluated two CNF generation techniques available in ABC – 1) standard Tseitin encoding, with some minimal optimizations like detection of multi-input ANDs, ORs and muxes, and 2) technology mapping based CNF generation [18], which uses optimization techniques commonly found in the technology mapping phase of logic synthesis to produce more compact CNF. A plot comparing these two techniques is given in our technical report [17]. The TM-based CNF generation significantly reduced the SAT run-time for the `spear` benchmarks, but actually increased the run-time for the `brummayerbiere` family of benchmarks. For other SAT

solvers, there wasn't an appreciable difference. Also, neither CNF generation technique improved much on NFLSAT.

The use of logic optimization techniques on the template circuits for arithmetic operators was beneficial in general, as can be seen from Fig. 2(b). The speed-up obtained from optimizing the templates was especially large for the `spear` benchmarks. We hypothesize that this is because that family of benchmarks has a wide variety of arithmetic operations, including several non-linear operations, which are the ones for which template optimization helps the most.

Discussion From our analysis of the different options in BEAVER, it seems that different settings of options will benefit different families, which indicates that some form of family-specific auto-tuning might be worth performing. It would also be worth further investigating the better performance of the non-clausal solver NFLSAT over the CNF-based solvers used in this study.

Acknowledgments. This work was supported in part by SRC contract 1355.001, NSF grants CNS-0644436, CNS-0627734, and CCF-0613997, and an Alfred P. Sloan Research Fellowship.

References

1. C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 4, chapter 8. IOS Press, 2009.
2. A. Biere. PicoSAT essentials. *JSAT*, 4:75–97, 2008.
3. R. D. Brummayer and A. Biere. Boolector: An efficient SMT solver for bit-vectors and arrays. In *Proc. of TACAS*, March 2009.
4. R. Bruttomesso, A. Cimatti, A. Franzen, A. Griggio, Z. Hanna, A. Nadel, A. Palti, and R. Sebastiani. A lazy and layered SMT(BV) solver for hard industrial verification problems. In *CAV 2007*, LNCS 4590, pages 547–560.
5. R. E. Bryant, D. Kroening, J. Ouaknine, S. A. Seshia, O. Strichman, and B. Brady. Deciding bit-vector arithmetic with abstraction. In *TACAS 2007*, LNCS 4424, pages 358–372.
6. L. de Moura and N. Bjorner. Z3: An efficient SMT solver. In *Proc. TACAS*, LNCS 4963, 2008.
7. N. Een and N. Sorensson. An extensible SAT-solver. In *Proc. SAT*, LNCS 2919, 2003.
8. V. Ganesh and D. Dill. A decision procedure for bit-vectors and arrays. In *CAV 2007*, LNCS 4590, pages 519–531.
9. F. Hutter, D. Babic, H. H. Hoos, and A. J. Hu. Boosting verification by automatic tuning of decision procedures. In *FMCAD 2007*, pages 27–34. IEEE Press.
10. P. Manolis, S. K. Srinivasan and D. Vroon. BAT Bit-level Analysis tool. In CAV 2007, pages 303-306.
11. H. Jain and E. M. Clarke. Efficient SAT solving for Non-Clausal Formulas using DPLL, Graphs and Watched Cuts. 46th Design Automation Conference, 2009.
12. K. Pipatsrisawat and A. Darwiche. Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.
13. SMT-COMP'08 results. Available at `http://www.smtexec.org/exec/?jobs=311`
14. SMT-LIB QF_BV logic. Available at `http://goedel.cs.uiowa.edu/smtlib/logics/QF_BV.smt`
15. SMT-LIB QF_BV benchmarks. Available at `http://combination.cs.uiowa.edu/smtlib/benchmarks.html`
16. Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification, release 70930. `http://www.eecs.berkeley.edu/~alanmi/abc/`.
17. S. Jha, R. Limaye, and S. A. Seshia. Beaver: Engineering an Efficient SMT Solver for Bit-Vector Arithmetic. Technical Report, EECS Department, UC Berkeley, 2009.
18. N. Een, A. Mishchenko, and N. Sorensson. Applying logic synthesis to speedup SAT. Proc. SAT 2007, LNCS 4501, pages 272–286.
19. H. S. Warren Jr. *Hacker's Delight*. Addison Wesley, 2003.