Towards a Mostly-Automated Prover for Bit-Vector Arithmetic

Iago Abal
HASLab / INESC TEC & Universidade do Minho
Braga, Portugal
iagoabal@di.uminho.pt

Jorge Sousa Pinto
HASLab / INESC TEC & Universidade do Minho
Braga, Portugal
jsp@di.uminho.pt

ABSTRACT

We present work in progress on the development of EasyBV, a specialized theorem prover for fixed-size bit-vector arithmetic.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving

General Terms

Algorithms, Verification

Keywords

Bit-vectors, Theorem proving, SMT, Rewriting

1. INTRODUCTION

The theory of fixed-size bit-vectors has gained momentum as a means to support encodings of verification problems involving word-level operations. While being a decidable theory supported by most SMT solvers, bit-vectors can preserve the high-level structure of a problem that would be lost using traditional propositional encodings. For instance, 32-bit multiplication is a bit-vector operation satisfying well-known properties, but at the bit-level it is represented by thousands of operations with no obvious meaning. Ideally, we want solvers to take advantage of this to handle problems more efficiently than a SAT solver can handle their bit-level counterpart. However, the only decision procedure for bit-vector arithmetic supporting non-linear operations is *bit-blasting*, which consists in translating a bit-vector problem into pure propositional SAT. This means that the high-level structure of the problem will be lost anyway.

Modern SMT solvers include preprocessing steps that perform local and global simplifications on the input problem before bit-blasting [3]. This preprocessing has been instrumental in the performance improvements recently achieved by SMT solvers for the theory of bit-vectors, but it is insufficient for many industrial scale scenarios. In particular, bit-blasting non-linear operations like multiplication is quadratic on the size of the operands. As a consequence, problems with heavy use of non-linear arithmetic tend to be intractable, as their propositional encoding may grow exponentially on the size of bit-vectors. This is a frequent situation in domains like public key cryptography, where operations such as multiplication and modulo are used intensively on large bit-vectors.

In our previous work we have explored the development of a rewriting system for bit-vector arithmetic to simplify equivalence problems in the domain of cryptography [1]. This prototype was tested using a small number of case studies from symmetric, public-key and elliptic-curves cryptography obtaining considerable speedups. Interestingly, we found that SMT solvers are missing straightforward but important algebraic rules, such as $(a_{[n]} \bmod b_{[n]}) \bmod b_{[n]} \equiv a_{[n]} \bmod b_{[n]}$. But the greater speedups were obtained by (sometimes tricky) equivalences that require estimating upper bounds for subterms, thus being substantially expensive to exploit. A simple, yet representative, example is $a_{[n]} \bmod k \equiv a_{[n]}$, which holds provided that we can statically show that any assignment for $a_{[n]}$ will be lesser than the constant k. The overall experience provided evidence that effective simplification of an arbitrary bit-vector problem will hardly be achieved in an automated fashion. The above results motivate us to our current proposal for a mostly-automated theorem prover for bit-vector arithmetic, in which domain-specific rewriting rules play a fundamental role. We believe this as yet unexplored approach will prove to be more efficient than current solutions.

2. OVERVIEW

EasyBV is a specialized theorem prover for the combined theory of bit-vectors, arrays and uninterpreted functions. It combines SMT solving with word-level simplifications to tackle the formula explosion problem created by bit-blasting. Figure 1 depicts the high-level architecture of the EasyBV toolset. The rule compiler parses rule files containing rewriting rules, typechecks each of these rules and stores them in the database. Rules can be checked at any time to guarantee that they state valid equivalences between bit-vector terms. The rule checker combines testing with SMT solving to offer a lightweight but practical approach to rule verification. The proof engine is the core component of the system, performing tactic-driven simplifications on the input problem until this can be handed to an SMT solver. It will either return

C^3*S*^{2}*E-13* 2013, July 10-12, Porto [Portugal]

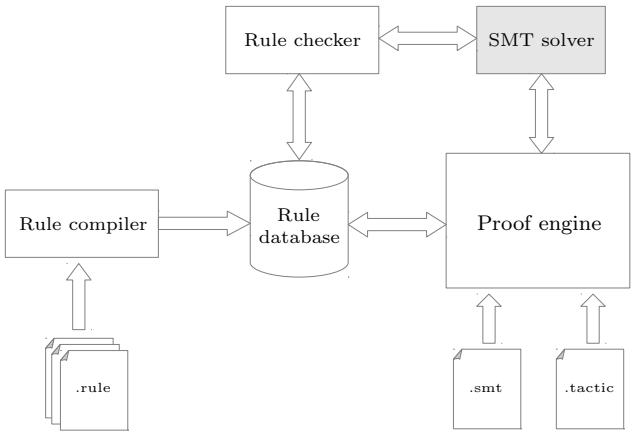

Figure 1: EasyBV architecture.

a proof log in case of success, provide a counterexample if some is reported by the SMT solver, or fail if simplification was insufficient. In the latter case, the user shall refine the rewriting system or the tactic script.

In order to illustrate the proving workflow, let us consider the problem of determining the satisfiability of $b \neq 0 \wedge f(0) = 0 \wedge f((a \bmod b) \bmod b) \neq (a = 0)\ ?\ 0 : f(a \bmod b)$, for bit-vectors of 1024-bits size. This problem is quite representative of those that (so far) we have found in the domain of public key cryptography, and turns out to be intractable by modern SMT solvers (namely Z3 3.2.1, CVC4 1.2, and MathSAT5 5.2.5).

Defining the rewriting system.

EasyBV incorporates a number of builtin rules of general usefulness, but hard problems will require developing domain-specific sets of rules. Rules are organized into modules thus simplifying reusability. Even though EasyBV may provide some hints using simple heuristics, the user is responsible for finding the adequate set of rewriting rules for simplification. Normally, a rule database is built incrementally as new problems arise, and this effort is considerably reduced after a few representative cases. The typechecker guarantees that the specified rules are safe and will not produce runtime errors when used by the rewriter, these errors include size-mismatching and division-by-zero. EasyBV would conveniently suggest the following rule given the above example: $(a_{[n]} \bmod b_{[n]}) \bmod b_{[n]} \rightarrow a_{[n]} \bmod b_{[n]}$.

Verifying the rewriting rules.

At some point, rules should be verified to gain confidence in the proofs performed by EasyBV. However, rewriting rules like the above often state general properties on arbitrary-sized bit-vectors, and cannot be proved automatically. Yet, proving a rule correct for some concrete bit-vector sizes may still be difficult for modern SMT solvers. For instance, our rule is hard to prove for bit-vectors larger than 16 bits. Looking for a compromise, we rely on a combination of random testing and SMT solving: some of the variables are set to randomly generated values until the resulting rule instance is effectively solvable. In our example, we may randomly pick a value for $a_{[n]}$, say k, and then rely on an SMT solver to check that $(k \bmod b_{[n]}) \bmod b_{[n]} \equiv k \bmod b_{[n]}$. In comparison with regular testing, each test vector verifies the equivalence for 2^n values of $b_{[n]}$, thus a few test cases may cover a significant portion of the state space.

Writing the tactic script.

Rewriting is performed automatically and the user can only customize the set of rules to be used. User guidance is mostly important to decide whenever we shall apply expensive global simplifications, or transformations that may be counterproductive. Notice that sometimes it is necessary to apply transformations that will first increase the size of the problem, in order to enable further simplifications. EasyBV offers a declarative tactic language that allows a coarse control of the simplification process. The highly declarative nature of tactics reflects our decision to trade-off efficiency for robustness. EasyBV also provides a standard proving strategy that will usually work for moderately complicated problems.

Solving the problem.

The rewriter is invoked automatically on any subterm that has changed and thus may be subject to further simplification. Rewriting is performed bottom-up and modulo associativity and commutativity (AC). We implement an efficient algorithm for AC rewriting based on bipartite graphs matching [2]. EasyBV will simplify the input problem according to what is specified by the tactic script, and then will call an external SMT solver only if the simplified problem can be solved in reasonable time. If bit-blasting would explode producing a huge boolean SAT problem, then EasyBV will fail indicating that simplification was insufficient. Our problem is reduced to $b \neq 0 \wedge f(0) = 0 \wedge (a = 0)\ ?\ f(a \bmod b) \neq 0 : \bot$ after rewriting and *if-lifting*. This new formula is considerably easier and can be handed to any state-of-the-art solver. Anyway, a global simplification pass would perform *constant propagation* replacing first a and then $f(0)$ with 0, thus showing the problem unsatisfiable.

3. ACKNOWLEDGMENTS

This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds, through FCT - Fundação para a Ciência e a Tecnologia, within project FCOMP-01-0124-FEDER-020486; and by ERDF through the COMPETE Programme, within project FCOMP-01-0202-FEDER-023137.

4. REFERENCES

[1] I. Abal, A. Cunha, J. Hurd, and J. Sousa Pinto. Using term rewriting to solve bit-vector arithmetic problems. SAT'12, pages 493–495. Springer-Verlag, 2012.

[2] S. M. Eker. Associative-commutative matching via bipartite graph matching. *Comput. J.*, 38(5):381–399, 1995.

[3] V. Ganesh, S. Berezin, and D. L. Dill. A decision procedure for fixed-width bit-vectors. Technical Report CSTR 2007-06, Department of Computer Science, Stanford University, 2006.