Appears in A. Balint, A. Belov, M.J.H. Heule, and M. Järvisalo (eds.), *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, volume B-2013-1 of Department of Computer Science Series of Publications B, University of Helsinki 2013. ISBN 978-952-10-8991-6.

Quantifier-Free Bit-Vector Formulas with Binary Encoding: Benchmark Description

Gergely Kovásznai, Andreas Fröhlich, Armin Biere
Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria

***Abstract*—This document describes several sets of benchmarks corresponding to quantifier-free bit-vector formulas. A generation script first creates all benchmarks in SMT2 format and then uses Boolector to generate CNF instances in DIMACS format by bit-blasting.**

I. Introduction

Bit-precise reasoning over fixed-size bit-vector logics (QF_BV) is important for many practical applications of Satisfability Modulo Theories (SMT), particularly for hardware and software verifcation. In [1], we argued that a *logarithmic* (w.l.o.g. *binary*) encoding, as used e.g. in the SMT-LIB format [2], leads to NExpTime-completeness of the underlying decision problem. Bit-blasting, as used in most current SMT solvers, therefore produces exponentially larger CNF formulas on certain QF_BV formulas. We provide generation scripts for several sets of QF_BV benchmarks in SMT-LIB format where this is the case and use bit-blasting to generate SAT benchmarks out of the original SMT2 specifications. All scripts and generated benchmarks are available at `http://fmv.jku.at/smtbench`.

II. Benchmarks

Our benchmark sets can be divided into two main categories: Expressing common bit-vector operations by other operations and general properties that can be expressed by a fragment of QF_BV with a restricted set of operations.

A. Translating Bit-Vector Operations

The first category contains 13 different benchmark sets and was used for verifying correctness of various translations between bit-vector operators. Having proved that *bitwise operations*, *equality*, and *slicing* suffice to derive NExpTime-hardness theoretically, we also wanted to give concrete examples of how to replace common bit-vector operations by those *base operations*. To check correctness, we encoded all translations into SMT2 and verified that no counter-example exists. We did this for 13 different operations. All benchmarks are unsatisfiable:

addition (`bvadd`), subtraction (`bvsub`), multiplication (`bvmul`), unsigned division (`bvudiv`), signed division (`bvsdiv`), unsigned remainder (`bvurem`), signed remainder (`bvsrem`), signed modulo (`bvsmod`), logical shift right (`bvlshr`), arithmetic shift right (`bvashr`), shift left (`bvshl`), unsigned less than (`bvult`), and signed less than (`bvslt`).

To give one specific example, addition can be expressed by base operations as follows:

$t_1{}^{[n]} + t_2{}^{[n]}$ is replaced by $ts_1{}^{[n]} \oplus ts_2{}^{[n]} \oplus c_{in}{}^{[n]}$ and additional constraints

1) $ts_1{}^{[n]} = t_1{}^{[n]}$
2) $ts_2{}^{[n]} = t_2{}^{[n]}$
3) $c_{out}{}^{[n]} = (ts_1{}^{[n]} \ \& \ ts_2{}^{[n]}) \mid (ts_1{}^{[n]} \ \& \ c_{in}{}^{[n]}) \mid (ts_2{}^{[n]} \ \& \ c_{in}{}^{[n]})$
4) $c_{in}{}^{[n]} = c_{out}{}^{[n]} \ll 1^{[n]}$

are added. Now again, $c_{out}{}^{[n]} \ll 1^{[n]}$ can be replaced by $ts_3{}^{[n]}$ and additional constraints

1) $ts_3{}^{[n]}\,[n:1] = c_{out}{}^{[n]}\,[n-1:0]$
2) $ts_3{}^{[n]}\,[0:0] = 0^{[1]}$

are added.

While this is well-known for the example of addition, expressing multiplication or other operations by using only those base operations is much more complicated and cannot be detailed in the scope of this description. On the other hand, this already explains the benefit of verifying correctness by using our benchmarks.

B. Bit-Vector Properties in PSpace

The second category consists of QF_BV benchmark sets with a reduced set of operations. In [3], we showed that QF_BV becomes PSpace-complete under certain restrictions on the set of allowed operations. While bit-blasting still produces exponentially larger formulas, the original benchmarks could be solved more efficiently, e.g. by using model checkers. It will be interesting to see whether any of the SAT solvers can also profit from this fact.

The 4 benchmark sets contained in this category are the following ones:

`ndist.a`: We verify that, for two bit-vector variables $x^{[n]}$, $y^{[n]}$, it holds that $x^{[n]} < y^{[n]}$ implies $(x^{[n]} + 1^{[n]}) \leq y^{[n]}$. The instances are unsatisfiable.

`ndist.b`: We give a counter-example (due to overflow) to the claim that, for two bit-vector variables $x^{[n]}$, $y^{[n]}$, it holds that $(x^{[n]} + 1^{[n]}) \leq y^{[n]}$ implies $x^{[n]} < y^{[n]}$. The instances are satisfiable.

`power2sum`: We verify that, for two bit-vector variables $x^{[n]} = 2^j$, $y^{[n]} = 2^k$, with $j \neq k$, $x^{[n]} + y^{[n]}$ cannot be a power of 2. The instances are unsatisfiable.

`shift1add`: We verify that for an arbitrary bit-vector $x^{[n]}$, there exists no bit-vector $y^{[n]} \neq x^{[n]}$ with $(x^{[n]} + y^{[n]}) = (x^{[n]} \ll 1^{[n]})$. The instances are unsatisfiable.

III. SMT2 and CNF generation

For each of the 17 benchmark sets, an individual generation script is provided. The scripts generate several instances of the given problem set, starting from a minimal bit-width up to a maximal bit-width, incrementing the bit-width by a given step size. Given those parameters as input, they output several SMT2 formulas with bit-vector variables of corresponding bit-widths. Additionally, a `generate.sh` script is included. This script automatically calls all individual generation scripts with appropriate parameters (i.e. bit-widths that create challenging but not too-hard instances) and afterwards calls *Boolector* [4] with argument `-de` to bit-blast the SMT2 instances and create CNF formulas in DIMACS format, therefore directly providing the input benchmarks for the SAT solvers. Additional CNF instances corresponding to different bit-widths can be created manually by using the individual scripts with custom parameters and then translating the output with Boolector.

IV. Practical Considerations

All benchmarks were originally created to evaluate the performance of SMT solvers. While most benchmarks were challenging for all SMT solvers, some solvers turned out to perform particularly well on specific instances. So far, it is not clear whether this difference in performance is due to SMT rewriting rules, differences in bit-blasting, or because of the underlying SAT solvers. It therefore will be interesting to see how various SAT solvers perform on the bit-blasted version of our benchmarks.

Acknowledgment

This work is supported by FWF, NFN Grant S11408-N23 (RiSE).

References

[1] G. Kovásznai, A. Fröhlich, and A. Biere, "On the complexity of fixed-size bit-vector logics with binary encoded bit-width," in *Proc. SMT'12*, 2012.

[2] C. Barrett, A. Stump, and C. Tinelli, "The SMT-LIB standard: Version 2.0," in *Proc. SMT'10*, 2010.

[3] A. Fröhlich, G. Kovásznai, and A. Biere, "More on the complexity of quantifier-free fixed-size bit-vector logics with binary encoding," in *Proc. CSR'13*, 2013.

[4] R. Brummayer and A. Biere, "Boolector: An efficient smt solver for bit-vectors and arrays," in *Proc. TACAS'09*, 2009.