Deep Bound Hardware Model Checking Instances, Quadratic Propagations Benchmarks and Reencoded Factorization Problems Submitted to the SAT Competition 2017

Armin Biere
Institute for Formal Models and Verification
Johannes Kepler University Linz

***Abstract*—In this benchmark description we describe our three set of benchmarks submitted to the SAT Competition 2016. The first contains bounded model checking problems from the deep bound track of the hardware model checking competition. The second crafted set of benchmarks has the sole purpose to show that the standard watch list implementation has a quadratic corner case. As third set of benchmarks we submitted factoring problems of products of medium sized primes, which seem to be hard for standard SAT solvers, but become trivial if the solution is reencoded back into the CNF by flipping literals appropriately.**

Deep Bound Hardware Model Checking Instances

The Hardware Model Checking Competition (HWMCC) [1] has a deep bound track, in which only safety model checking benchmarks are considered, and which remained unsolved in the main track. Model checkers participating in this track are supposed to print bounds k, as soon they were able to show that a bad state violating the given safety property can not be reached in k steps from an initial state. This track was inspired by the need to run bounded model checking for big industrial models which are too hard to be solved completely. In this setting a model checker is superior to another one if it goes deeper, i.e., it reaches a higher bound k.

For this benchmark set HWMCC15DEEP we used the 135 model checking problems of the deep bound track of the HWMCC'15, see http://fmv.jku.at/hwmcc15, which consists of 123 industrial and 12 academic instances (the latter all from the BEEM family). The deep bound track is dominated by plain bounded model checkers, which after some optimizations, unroll the circuit, and then use a SAT solver directly. In this track our own BLIMC model checker, which is based on Lingeling [2] and runs hors concurs in the competition, performs best. It uses SAT preprocessing to simplify the transition relation once before copying it and running an incremental SAT check for each new bound following [3].

In order to generate non-incremental problems instead, we took the deepest bound k reached by BLIMC on these benchmarks and unrolled the model up to the bounds $k-2$, $k-1$, k, $k+1$, $k+2$ and all the powers of two 2^i with $2^i < k-2$, as well the bound 0 which checks whether an initial state is bad.

Supported by FWF, NFN Grant S11408-N23 (RiSE).

The unrolling process is based on functional substitution [4] as implemented in the AIGUNROLL tool, which comes with the AIGER distribution (see http://fmv.jku.at/aiger). However, if the bound reached by BLIMC in the given time limit of one hour is 100 or more, then the benchmark was not included. This removed 24 models. One of them was as BEEM model. The remaining 109 models are further split in two sets. The first set contains 55 "small" models, where the original sequential model (before unrolling) has less than 100 000 AND gates. The rest makes up the set of 54 "big" models.

The resulting AIGs after unrolling the models are translated into CNF with AIGTOCNF, which yields 433 small CNFs and 330 big CNFs, after removing trivial ones, where the unrolled AIG is constant false. There are 134 non-interesting benchmarks in the small set and 67 non-interesting benchmarks in the big set which can all be solved by MINISAT [5] in less than a minute. There are additional 97 small and 82 big benchmarks which are solved by all five test solvers in 5000 seconds (LINGELING, GLUCOSE, MINISAT, MAPLECOMSPSLRB from the SAT Competition 2016 and CADICAL). At the end we obtain 202 interesting small benchmarks and 181 interesting big benchmarks. Note, that we kept 33 small and 21 big CNFs, which were not solved by any test solvers.

Crafted Quadratic Propagations Benchmarks

The standard implementation of watch lists in MINISAT and its descendants is suboptimal and in some situation might lead to a quadratic overhead. This observation occurs in an JAIR article by Ian Gent [6] from 2013. For some benchmarks from the SAT Competition 2016, we have seen severe slowdown in propagation speed for an earlier version of our new SAT solver CADICAL, which were due to exactly the observation made by Ian Gent. The solution we implemented, which was suggested in this article, is to save the position of the replaced literal and start searching from that position instead of from the beginning of the clause, the next time a watch in that clause has to be replaced.

The article does not really have convincing experimental evidence that this scheme is beneficial in practice and also failed to provide benchmarks, where this quadratic behavior can be observed. The purpose of our BCPSQR benchmark set is to provide exactly such parameterized set of crafted instances, where propagation in MINISAT is quadratic. The basic idea is to have a very long clause, say $x_1 \vee x_2 \vee \ldots \vee x_{1000}$ and force the solver to assign and thus watch all the literals in turn, e.g., assign $x_1 = 0$, $x_2 = 0$, ..., $x_{1000} = 0$, in this order.

However, since MINISAT sorts clauses to remove duplicate literals in increasing variable index order, and then, due to how the binary heap for decision ordering works, decides on largest variable indices first, actually except for the first decision which is always the first variable, this is hard to achieve, e.g., the default decision assignments in MINISAT would be $x_1 = 0$, $x_{1000} = 0$, $x_{999} = 0$, ..., $x_2 = 0$ but the clause $x_1 \vee x_{1000} \vee x_{999} \ldots \vee x_2$, which would trigger the intended bad behavior, becomes $x_1 \vee x_2 \vee \ldots \vee x_{1000}$ after sorting during parsing.

This can be addressed by adding the following binary clauses $(\bar{x}_2 \vee x_{1999})$, $(\bar{x}_3 \vee x_{1998})$, ..., $(\bar{x}_{1000} \vee x_{1001})$ and would result in quadratic propagation.

However, the whole input also has to go through variable elimination untouched. To achieve that, the long clause of n variables is replaced by m copies, adding one new variable (positively too) each time. Then appropriate binary clauses are added which turn these new variables into the output of NAND gates over the old variables.

Further, those new variables are restricted by a size m parity constraint, encoded with 2^{m-1} clauses, which has the all zero assignment as solution. For the submitted benchmarks we use $m = 4, 5, 6$, which all make variable elimination ineffective.

Finally, a binary clauses $(\bar{a} \vee c)$ as above is split into

$$\begin{array}{cccc} & (a \vee b_1) & (b_1 \vee c) & \\ (\bar{a} \vee \bar{b}_1 \vee \bar{b}_2 \vee \bar{b}_3) & (a \vee b_2) & (b_2 \vee c) & (\bar{b}_1 \vee \bar{b}_2 \vee \bar{b}_3 \vee \bar{c}) \\ & (a \vee b_2) & (b_3 \vee c) & \end{array}$$

with new variables b_1, b_2, b_3 ordered after a and before c. Adding less than 3 variables per implication would allow variable elimination to eliminate all additional variables.

These problems are satisfied by MINISAT without producing any conflict, but require substantial propagation overhead starting with $n > 100\,000$.

REENCODED FACTORIZATION PROBLEMS

In MINISAT the heuristic for assigning a decision variable the first time before it was ever assigned during propagation is to assign it to false. Indeed, it seems that solving many real-world instances benefits from this choice of phase decision heuristic, even though there are cases where the opposite is much better, e.g., for miters between correct and incorrect large multipliers [7]. The organizers of the SAT Competition 2014 selected certain benchmarks, which are very hard unless the simple phase heuristic of MINISAT is used, as for instance discussed in [8]. In essence there is the danger of using artificially trivial benchmarks.

In order to stress this point we generated CNFs which model factoring the product of primes. For each benchmark we picked random primes with 9 to 11 decimal digits, computed their product and generated an SMT benchmark in bit-vector logic, which forces the output of a multiplier to the concrete product. One has to make sure that the bit-length of the output is big enough. The inputs are zero-extended, different from one and ordered. Then the SMT benchmark is bit-blasted by BOOLECTOR [9] into an AIG and translated to CNF with AIGTOCNF.

This procedure generates medium to hard satisfiable instances for today's SAT solver, and, of course, can be made arbitrarily hard, by increasing the number of digits. However, since we know the primes, we can easily construct an assignment to the input bits which then satisfies the CNF after unit-propagation. With this knowledge and using the assumption mode of PICOSAT [10] we generated a complete satisfying assignment for each generated CNF. This assignment is then used to flip literals in the CNF as follows. If a variable is assigned to true the variable is flipped (replaced by its negation). The resulting CNF is trivially satisfiable by assigning all variables to false and thus trivial to solve by say MINISAT.

Beside generating the original hard instance, and then the all zero instance, we repeated the procedure, but flip variables which are assigned to false. Now the instance becomes trivially satisfiable by assigning all variables to true. It turns out these instances are also easy to solve for solvers which detect this situation, e.g., LINGELING, or assign to true first, as phase decision heuristic, but they seem to be as hard as the original instance for solvers which assign to false first, e.g., MINISAT.

REFERENCES

[1] G. Cabodi, C. Loiacono, M. Palena, P. Pasini, D. Patti, S. Quer, D. Vendraminetto, A. Biere, and K. Heljanko, "Hardware model checking competition 2014: An analysis and comparison of solvers and benchmarks," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, pp. 135–172, 2014 (published 2016).

[2] A. Biere, "Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016," in *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, and M. Järvisalo, Eds., vol. B-2016-1. University of Helsinki, 2016, pp. 44–45.

[3] S. Kupferschmid, M. D. T. Lewis, T. Schubert, and B. Becker, "Incremental preprocessing methods for use in BMC," *Formal Methods in System Design*, vol. 39, no. 2, pp. 185–204, 2011.

[4] T. Jussila and A. Biere, "Compressing BMC encodings with QBF," *Electr. Notes Theor. Comput. Sci.*, vol. 174, no. 3, pp. 45–56, 2007.

[5] N. Eén and N. Sörensson, "An extensible SAT-solver," in *SAT*, ser. Lecture Notes in Computer Science, vol. 2919. Springer, 2003, pp. 502–518.

[6] I. P. Gent, "Optimal implementation of watched literals and more general techniques," *J. Artif. Intell. Res. (JAIR)*, vol. 48, pp. 231–251, 2013.

[7] A. Biere, "Collection of Combinational Arithmetic Miters Submitted to the SAT Competition 2016," in *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, and M. Järvisalo, Eds., vol. B-2016-1. University of Helsinki, 2016, pp. 65–66.

[8] A. Biere and A. Fröhlich, "Evaluating CDCL variable scoring schemes," in *SAT*, ser. Lecture Notes in Computer Science, vol. 9340. Springer, 2015, pp. 405–422.

[9] A. Niemetz, M. Preiner, and A. Biere, "Boolector 2.0 system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, pp. 53–58, 2014 (published 2015).

[10] A. Biere, "Picosat essentials," *JSAT*, vol. 4, no. 2-4, pp. 75–97, 2008.