GoRRiLA and Hard Reality

Konstantin Korovin[⋆] and Andrei Voronkov[⋆⋆]

The University of Manchester
{korovin|voronkov}@cs.man.ac.uk

Abstract. We call a *theory problem* a conjunction of theory literals and a *theory solver* any system that solves theory problems. For implementing efficient theory solvers one needs benchmark problems, and especially hard ones. Unfortunately, hard benchmarks for theory solvers are notoriously difficult to obtain. In this paper we present two tools: *Hard Reality* for generating theory problems from real-life problems with non-trivial boolean structure and *GoRRiLA* for generating random theory problems for linear arithmetic. Using GoRRiLA one can generate problems containing only a few variables, which however are difficult for all state-of-the-art solvers we tried. Such problems can be useful for debugging and evaluating solvers on small but hard problems. Using Hard Reality one can generate hard theory problems which are similar to problems found in real-life applications, for example, those taken from SMT-LIB [3].

1 Introduction

All modern satisfiability modulo theories (SMT) solvers contain two major parts: the boolean reasoning part and the theory reasoning part. Both boolean and theory reasoning are important for an efficient solver and both are highly optimised. Usually, a theory solver can be seen as a standalone procedure solving satisfiability of sets of theory literals (e.g. if we consider linear real arithmetic then the theory solver is required to solve systems of equations and inequalities). The development of theory solvers is on its own right a highly non-trivial problem. Unfortunately, currently there are no available benchmarks for debugging and evaluating theory solvers. Indeed, almost all problems in the standard SMT benchmark library SMT-LIB are problems with a non-trivial boolean structure and cannot be used for evaluating dedicated theory solvers. Moreover, when evaluating an SMT solver it is usually not clear where the performance bottleneck is: in the theory or the boolean part. In addition to the evaluation problem, one needs a good set of benchmarks for debugging and checking consistency of theory solvers. There are three common ways of generating theory problems: (i) randomly, (ii) based on proof obligations coming from applications, and (iii) based on logging benchmarks generated by running solvers on real-life problems [10].

In this paper we present two tools: GoRRiLA for randomly generating small yet hard theory problems for linear (integer and rational) arithmetic and Hard Reality for generating theory problems from real-life problems with non-trivial boolean structure. The advantage of randomly generated problems is that we can generate many diverse

[⋆] Supported by a Royal Society University Research Fellowship
[⋆⋆] Partially supported by an EPSRC grant

problems, and therefore such problems are well-suited for testing correctness especially of newly developing solvers. We observed that there are a number of randomly generated problems with only few variables that are hard for all SMT solvers we tried. For example in the case of linear integer arithmetic already problems with 4 variables can be hard for many state-of-the-art SMT solvers. Therefore, we believe randomly generated problems can also be used for exploring efficiency issues with theory solvers. Moreover, using our tool we found small linear integer arithmetic problems on which some state-of-the-art SMT solvers return inconsistent results. In Section 2 we present our tool, called GoRRiLA, for randomly generating hard linear arithmetic problems and evaluating solvers on such problems.

Randomly generated problems have one major disadvantage: usually they are very different from the problems occurring in practice. On the other hand, problems coming from applications are not easy to find. In fact, this work appeared as a side-effect of our work on evaluating solvers for linear arithmetic, since we were unable to find good collections of systems of linear inequalities over rationals for testing and benchmarking our system. In Section 3 we present our tool, called Hard Reality, for generating theory problems from real-life quantifier-free SMT problems. Using Hard Reality we have been able to generate a variety of theory problems that at the same time are representative of the problems that the theory parts of SMT solvers are dealing with.

The intended applications of the obtained theory benchmarks by GoRRiLA and Hard Reality are the following:

- testing correctness and efficiency of theory reasoners on both randomly generated and realistic benchmarks.
- evaluating and comparing dedicated theory reasoners and theory reasoners within SMT solvers.

In this paper we leave out evaluation of incrementality of theory solvers. Although incrementality is crucial for theory solvers within SMT framework (see, e.g., [7]), we believe it is an orthogonal issue to the main goal of this paper: generation of hard theory problems. Hard theory problems can also be used to evaluate incrementality of theory solvers, but this goes beyond the scope of this paper.

2 GoRRiLA

It is well-known how to generate random propositional SAT problems and in particular hard SAT problems (see, e.g., [9]). In this section we present a method for randomly generating hard linear arithmetic problems which is inspired by methods used in the propositional case. First we consider the case of linear integer arithmetic and later show that our method is also suitable for linear rational arithmetic. For each integer $n \geq 1$, let X_n denote a set of n variables $\{x_1, \ldots, x_n\}$.

A *linear constraint with integer coefficients over the set of variables X_n*, (or simply a *linear constraint over X_n*), is an expression of the form $a_1x_1 + \ldots + a_nx_n + a_0 \diamond 0$ where $\diamond \in \{\geq, >, =, \neq\}$ and $a_i \in \mathbb{Z}$ for $1 \leq i \leq n$. A *linear problem with integer coefficients over the set of variables X_n* (or simply a *linear problem*) is a set of linear constraints.

Suppose we would like to generate a random linear inequality $a_1x_1 + \ldots + a_nx_n + a_0 \geq 0$. To this end, we fix the following parameters:

– the number n of variables;
– the lower and the upper integer bounds on coefficients;
– the number k of variables that can have non-zero coefficients in the constraint.

Let $\mathscr{C}$ denote the set of all integers between the lower and the upper bound on the coefficients. Then we generate:

1. a random subset $i_1, \dots, i_k$ of X_n of cardinality k, each subset is selected with an equal probability;
2. uniformly and randomly k integers $a_1, \dots, a_k$ in $\mathscr{C} \setminus \{0\}$;
3. uniformly and randomly an integer $a_0 \in \mathscr{C}$.

After that we use $a_1 x_{i_1} + \dots + a_k x_{i_k} + a_0 \geq 0$ as the random inequality. In the same way we can select random constraints of other kinds, for example, strict inequalities.

In order to generate a random system of linear constraints we also fix, $N_{\geq}$, $N_{>}$, $N_{=}$ and $N_{\neq}$ – the numbers of constraints of each kind respectively. When all problem parameters are fixed a *random linear problem* is generated as a set containing $N_{\diamond}$ random constraints of each type $\diamond \in \{\geq, >, =, \neq\}$. Below we assume that $\diamond$ ranges over all constraint types $\{\geq, >, =, \neq\}$. For simplicity, we assume that the set of coefficients $\mathscr{C}$ and the number of variables k in the constraints are uniformly fixed. Similar to the propositional case, we can observe a transition from almost always satisfiable problems to almost always unsatisfiable ones when we increase the number of constraints relative to the number of variables. As in the propositional case, it turned out that problems hard for modern solvers occur in the region where approximately 50% of generated problems are satisfiable. In order to generate a sequence of *hard random problems* we take the number of variables n as the leading parameter and the numbers of constraints of each type as $N_{\diamond} = q_{\diamond} * n$, where $q_{\diamond}$ is a rational number. Generally, since we have different types of constraints, hard problems can occur with different combinations of parameters $q_{\diamond}$, for our purposes we can take any such combination or introduce a bias towards some types of constraints.

Let us remark that any constraint with rational coefficients can be normalised into an equivalent constraint with integer coefficients. Therefore we can use the procedure for generating linear integer problems to generate linear rational problems.

We implemented our random generation procedure in GoRRiLA, where the user can specify problem parameters and number of problems to be generated. Each randomly generated problem is submitted to a specified theory solver for evaluation. As a result of a GoRRiLA run we obtain a file, each line of which encodes: a random problem, name of the theory solver used for the evaluation of this problem, the result of the evaluation and the time used by the solver. Using GoRRiLA, we also can evaluate and compare results with other solvers. GoRRiLA has an option to choose between storing either the generated problems or their codes. Using codes is motivated by the fact that we can generate hundreds of thousands of problems out of which only a small fraction is of interest to be generated explicitly.

Using GoRRiLA we found that many state-of-the-art SMT solvers return inconsistent results already on problems with 3 variables. For example: SMT-COMP'08 versions of MathSAT and Z3 both return incorrect results and the SMT-COMP'09 version of SatEEn returns segmentation fault on small problems generated by GoRRiLA within

$$\begin{array}{ll}
\mathit{random\_set}(L) & \Rightarrow \text{add } L \text{ to } \mathscr{L}; \\
\mathit{random\_set}(F_1 \wedge \ldots \wedge F_n) & \Rightarrow \mathit{random\_set}(F_1);\ \ldots\ ; \mathit{random\_set}(F_n); \\
\mathit{random\_set}(F_1 \vee \ldots \vee F_n) & \Rightarrow \mathit{random\_set}(F_i), \text{ where } i \text{ is a random integer between 1 and } n.
\end{array}$$

Fig. 1: Naive algorithm for formulas in negation normal form

a few seconds. Although these bugs (at least for MathSAT and Z3) seems to be fixed in the current versions of these solvers we believe that if the designers of the systems used GoRRiLA, they would have found these bugs quickly and easily.

Using GoRRiLA we can easily generate linear integer problems with only 4 variables hard for all SMT solvers we tried. For linear rational arithmetic we evaluated SMT solvers over 10^5 easy problems with 3 to 9 variables. All solvers returned consistent results on these problems. Using GoRRiLA we can generate hard linear rational arithmetic problems for all SMT solvers we tried, starting from 30 variables. Below we show the number of problems which turned out to be hard for various SMT solvers, when we generated 1000 linear integer problems (LIA) with 4 variables and 100 linear rational problems (LRA) with 40 variables. All experiments were run on a 2GHz Linux machine with 4GB RAM.

	Barcelogic	CVC3	MathSAT	Z3
LIA (4 vars) > 10s	11	2	16	25
LRA (40 vars) > 10s	75	100	26	100

3 Hard Reality

We will work with a family $\mathscr{A}$ of randomised algorithms which, given a formula F as an input, produce a theory problem G. This formula G will be relevant to F, in the sense described below.

We will use the notion of a *path through (the matrix) of F*, defined in [1,4]. Essentially a path is a conjunction of literals occurring in F, that is, a theory problem built from literals occurring in F. In the simplest case when F is in CNF, every path contains one literal from every clause in F. In general, the set of all paths $G_1, \ldots, G_n$ through F has the property that F is equivalent to $G_1 \vee \ldots \vee G_n$. One can argue that each path through the matrix of F is *relevant* to SMT solving for F since an SMT solver can generate this path as a theory problem. In the rest of this section we will describe several randomized algorithms for generating relevant theory problems as paths through formulas.

Naive algorithm. We start with a naive algorithm shown in Figure 1. In the algorithm L ranges over literals. It uses a set of theory literals $\mathscr{L}$ which initially is empty. When the algorithm terminates, the conjunction of literals in $\mathscr{L}$ is the output theory problem. It is not hard to argue that the algorithm computes a random path through F, moreover, each path is computed with an equal probability.

Improved Algorithm. The naive algorithm *random_set* has a major deficiency that the probability of producing trivially unsatisfiable theory set is very high. For example, if we consider a conjunction consisting of n repetitions of a tautology $(L \vee \neg L)$ then the probability of generating a trivially unsatisfiable theory set containing both L and

```
$random\_set(L)$                          $\Rightarrow$ if $\widetilde{L} \in \mathscr{L}$ then fail;
                                              add $L$ to $\mathscr{L}$;
$random\_set(F_1 \wedge \ldots \wedge F_n)$ $\Rightarrow$ if for some $F_i$ we have $\widetilde{F_i} \in \mathscr{L}$ then fail;
                                              let $G_1, \ldots, G_m$ be all formulas among the $F_i$'s not belonging to $\mathscr{L}$;
                                              add $G_1, \ldots, G_m$ to $\mathscr{L}$;
                                              $random\_set(G_1);\ \ldots\ ; random\_set(G_m)$;
$random\_set(F_1 \vee \ldots \vee F_n)$ $\Rightarrow$ if some $F_i$ belongs to $\mathscr{L}$ then return;
                                              let $G_1, \ldots, G_m$ be all formulas among the $F_i$'s
                                                  such that $\widetilde{F_i}$ does not belong to $\mathscr{L}$;
                                              if $m = 0$, then fail;
                                              let $i$ be a random integer between 1 and $m$;
                                              add $G_i$ to $\mathscr{L}$; $random\_set(G_i)$
```

Fig. 2: Improved algorithm

$\neg L$ is $1 - 1/2^{n-1}$. It is worth noting that unit propagation in SMT solvers guarantees that such sets are never passed to theory solvers. There is an easy fix to address this problem by restricting the random choice of the formula F_i in a disjunction to avoid having complementary literals to $\mathscr{L}$. If we change the algorithm in this way, it may be the case that the disjunctive case may fail: this happens when every F_i in the disjunction is a literal whose complement is in $\mathscr{L}$. In this case we restart the algorithm.

Algorithms of this kind still have a very high probability of generating a trivially unsatisfiable set in some cases, especially when we have to deal with formulas containing equivalence or if-then-else. The problem occurs when the algorithm is first applied to a non-literal F and later to the negation of F. In addition, it may generate problems that we call *unrealistic*: these are problems that would be avoided by SMT solvers. For example, consider the case when we apply it to a formula $F \wedge F$. Then it will process both copies of F and may select different sets of literals from these copies.

Therefore, we modify the algorithm to avoid both kinds of problem. The modified algorithm stores arbitrary formulas in $\mathscr{L}$. We assume that the input formula contains no occurrences of subformulas of the form $\neg\neg F$ (such occurrences of double negation can be trivially eliminated). Similarly to literals, we call formulas F and $\neg F$ *complementary*. The formula complementary to F will be denoted by $\widetilde{F}$. The improved algorithm is show in Figure 2. This algorithm stores in $\mathscr{L}$ formulas to which it has previously been applied. It returns the conjunction of all literals in $\mathscr{L}$. Upon failure, it restarts from scratch. Next we consider connectives that often occur in the SMT problems and have to be handled with care to avoid generating both trivial and unrealistic problems. These are if-then-else, $\leftrightarrow$ and *xor*. In addition, the SMT-LIB language has if-then-else terms that create extra problems for the algorithm. The rule for handling the if-then-else connective is given below, other rules are omitted due to lack of space.

```
$random\_set($if $F$ then $F_1$ else $F_2) \Rightarrow$
    case $F \in \mathscr{L}$ or $\widetilde{F_2} \in \mathscr{L}$: call $random\_set(F \wedge F_1)$;
    case $\widetilde{F} \in \mathscr{L}$ or $\widetilde{F_1} \in \mathscr{L}$: call $random\_set(\widetilde{F} \wedge F_2)$;
    otherwise: call randomly either $random\_set(F \wedge F_1)$ or $random\_set(\widetilde{F} \wedge F_2)$
```

Implementation, Options and Experimental Results. We implemented the improved algorithm (Figure 2) in the Hard Reality tool, which uses an SMT parser provided by the SMT-LIB initiative [8]. A simple way to use Hard Reality is by providing:

- a path to the directory with SMT problems,
- a path to an SMT solver,
- complexity of the resulting theory problem (this is done by specifying lower time limit required by the SMT solver to solve the problem),
- search time limit for the output theory problem.

Given these options, for each file in the input directory Hard Reality will search for randomly generated problems based on the improved algorithm, until a sufficiently hard problem is found (for the given SMT solver), or the search time reaches the specified limit. Hard Reality has also options for generating a maximal satisfiable subset/minimal unsatisfiable subset of the generated theory set (based on the SMT solver). We find that (attempting) generating maximal satisfiable sets can produce hard problems. Generating minimal unsatisfiable problems also gives us an insight to the theory problems occurring in applications.

Let us note that since an SMT solver typically needs to solve thousands of theory problems during the proof search, even theory problems requiring tenths of a second to be solved can be considered as non-trivial. Hard Reality has successfully generated problems using different solvers such as Barcelogic [11], CVC3 [2] and Z3 [6]. The following table summarises the numbers of generated hard theory problems in different theories with the times based on the Z3 solver. It is easier to generate problems hard for other solvers. The theory problems are generated from problems in the SMT-LIB.

Theory	QF_LRA	QF_LIA	QF_BV	QF_AUFBV
Timeout after 120s	79	1	2	14
Solved in $0.1s \leq t \leq 120s$	30	72	79	87

4 Conclusion

We presented two tools: GoRRiLA for random generation of small yet hard problems for linear arithmetic and Hard Reality for randomly extracting hard and realistic theory problems.

GoRRiLA is well-suited for generating diverse problems for linear arithmetic which can be used for debugging and exploring efficiency issues with theory solvers. We have shown that using GoRRiLA it is easy to find bugs in theory solvers. We observe that generated problems with only few variables are already hard for theory reasoners within state-of-the-art SMT solvers. As a future work it would be useful to extend GoRRiLA to other theory domains and combination of theories.

Using Hard Reality we have been able to generate a number of theory problems which are closely related to problems coming from applications and non-trivial for state-of-the-art SMT solvers. One of the issues we considered is how to avoid generating trivially unsatisfiable problems. Other approaches to this problem can be investigated, in particular one can first apply the definitional transformation to the original formula F to obtain a CNF equisatisfiable with F and delegate the problem of choosing relevant theory literals to a SAT solver. Another approach to theory problem generation can be based on logging problems generated on different branches during proof search of an SMT solver, but this would require modifications to a specific SMT solver, whereas in our approach we can use any SMT solver as an input to Hard Reality (or use no SMT solver at all). With our approach we were able to generate theory problems which were even harder for solvers than the original problem used in the generation process. We observed that only when we used most efficient SMT solvers we were able to generate theory problems hard for all solvers. A further application of Hard Reality can be in using generated hard theory problems to evaluate incrementality of the theory solvers.

Our benchmarking and testing of various versions of SMT solvers gave remarkable results: we have found a number of problems on which these solvers were unsound and also many problems on which some versions of solvers were running very long times as compared to other solvers. Although some of these bugs and performance problems have been fixed in the following versions of the solvers these problems could have been fixed much earlier if the designers of the solvers used our tools.

In a related work [5] random problems with non-trivial boolean structure are used for debugging solvers for theories of bitvectors and arrays (the tool was later extended to other theories). The main differences with our work is that we are focusing on generating: (i) theory problems, (ii) which are hard and (iii) related to problems coming from applications. GoRRiLA and Hard Reality with examples of generated problems are freely available at http://www.cs.man.ac.uk/~korovink/hr/.

We thank Leonardo de Moura and Nikolaj Bjørner with whom we exchanged many letters during this work.

References

1. P.B. Andrews. Theorem proving via general matings. *Journal of the ACM*, 28(2):193–214, 1981.
2. C. Barrett and C. Cesare Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *CAV '07*, volume 4590 of *LNCS*, pages 298–302. Springer-Verlag, July 2007. Berlin, Germany.
3. Clark Barrett, Silvio Ranise, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2008.
4. W. Bibel. On matrices with connections. *Journal of the ACM*, 28(4):633–645, 1981.
5. R. Brummayer and A. Biere. Fuzzing and delta-debugging SMT solvers. In *7th Intl. Workshop on on Satisfiability Modulo Theories (SMT'09)*, 2009.
6. L. M. de Moura and N. Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
7. G. Faure, R. Nieuwenhuis, A. Albert Oliveras, and E. Rodríguez-Carbonell. SAT modulo the theory of linear arithmetic: Exact, inexact and commercial solvers. In *SAT 2008*, volume 4996 of *LNCS*, pages 77–90. Springer, 2008.
8. G. Hagen, D. Zucchelli, and C. Tinelli. Smt parser v3.0. available at http://combination.cs.uiowa.edu/smtlib/.
9. D.G. Mitchell, B. Selman, and H.J. Levesque. Hard and easy distributions of SAT problems. In *AAAI 1992*, pages 459–465, San Jose, CA, January 1992. AAAI Press/MIT Press.
10. R. Nieuwenhuis, T. Hillenbrand, A. Riazanov, and A. Voronkov. On the evaluation of indexing techniques for theorem proving. In R. Goré, A. Leitch, and T. Nipkow, editors, *IJCAR 2001*, volume 2083 of *LNAI*, pages 257–271. Springer, 2001.
11. R. Nieuwenhuis and A. Oliveras. Decision Procedures for SAT, SAT Modulo Theories and Beyond. The BarcelogicTools. (Invited Paper). In G. Sutcliffe and A. Voronkov, editors, *LPAR'05*, volume 3835 of *Lecture Notes in Computer Science*, pages 23–46. Springer, 2005.