Fuzzing and Verifying RAT Refutations with Deletion Information

Walter Forkel
TU Dresden, Germany

Tobias Philipp
TU Dresden, Germany

Adrin Rebola-Pardo
TU Wien, Austria

Elias Werner
TU Dresden, Germany

Abstract

Contemporary SAT solvers emit proofs of unsatisfiability in the DRAT format to guarantee correctness of their answers. Therefore, correctness of SAT solvers is reduced to correctness of DRAT checkers, which are relatively small programs that decide whether a DRAT refutation is correct. We present a new fuzzing technique that automatically finds bugs in DRAT checkers by comparing the outputs of two DRAT checkers. In case their outputs are different a mechanically verified DRAT checker finally decides which checker has given the correct answer. Experiments show that our method finds bugs in available checkers, and also demonstrate that a common design choice in efficient DRAT checkers is inconsistent with the specification.

Introduction

The Boolean satisfiability problem (SAT) asks whether there is a satisfying interpretation for a propositional formula and is one of the most prominent problems in computer science and artificial intelligence. Historically, SAT was the first problem shown to be NP-*complete* by Cook and Levin, demonstrating that every problem in the complexity class NP can be reduced to SAT (Cook 1971). SAT solving has significantly advanced over the last decades, and solvers are evaluated yearly in international SAT competitions. Their performance over, e.g., hardware and software verification (Biere et al. 1999) has improved to the extent of being widespread tools in the industry. Modern SAT solvers are based on the CDCL (conflict directed clause learning) algorithm (Davis, Logemann, and Loveland 1962; Silva and Sakallah 1996) and use many advanced techniques such as *clause learning* (Silva and Sakallah 1996), *clause removal* (Audemard and Simon 2009), and *formula simplifications* (Eén and Biere 2005; Järvisalo, Heule, and Biere 2012). Unfortunately, even intensively-tested SAT solvers contain bugs, that have been detected using fuzzing methods (Brummayer, Lonsing, and Biere 2010; Manthey and Lindauer 2016).

Bugs where SAT solvers end up incorrectly reporting unsatisfiability are particularly hard to detect. In order to have witnesses for the unsatisfiable case, *certificates* for unsatisfiability, also known as *unsatisfiability proofs*, were developed (Zhang and Malik 2003; Gelder 2002). Today, the *DRAT (Deletion Resolution Asymmetric Tautology) format* (Wetzler, Heule, and Hunt Jr 2014) is the *de facto* standard in the SAT community, and proof generation in this format is a requirement in the main track of the SAT Competition 2016. A DRAT refutation traces clause additions and deletions during a run of a SAT solver, i.e. it is a sequence of sentences where each sentence is derived from earlier sentences by applying an inference rule, called *Resolution Asymmetric Tautology* (RAT). Deletion information can be used to shrink the clause database. Independent programs, called *checkers*, decide whether a sequence of these sentences is a DRAT refutation. In case the checker accepts the DRAT refutation together with the input formula, we know that the input formula is unsatisfiable, assuming that the checker works correctly. Recently, the DRAT format received international media attention because SAT solvers solved the Pythagorean Triples Problem. Its 200 Terabytes proof, expressed in this format, became the largest machine-checked proof as of today (Heule, Kullmann, and Marek 2016).

Due to the increasing size of DRAT proofs and their importance in practice, we want to improve the confidence in the correctness of DRAT checkers such as *drat-trim* (Wetzler, Heule, and Hunt Jr 2014) and *proofcheck* (Manthey and Philipp 2015). Our contribution is a new combination of a fuzzing method and a mechanically verified tool that automatically find bugs in these systems. Our method is effective as experiments revealed some bugs affecting completeness in state-of-the-art DRAT checkers. In particular, we show that deletion information is neither adequately handled by *drat-trim* nor by *proofcheck*. Moreover, we observed that *MiniSAT* produces unusual DRAT refutations.

Background

We consider an infinite set of propositional variables $\mathcal{V}$. A literal L is either a propositional variable A or its negation $\neg A$. The complement of a literal L is denoted by $\overline{L}$. Clauses are finite disjunctions of literals, represented by finite sets of literals, and formulas are finite conjunctions of clauses, represented by a multiset of clauses. A *tautological* clause is a clause containing A and $\neg A$ for some variable A. Let C and D be clauses such that $A \in C$ and $\neg A \in D$. Then, the *resolvent* of C and D upon A is the clause $(C \backslash \{A\}) \cup (D \backslash \{\neg A\})$. Interpretations map formulas to truth values respecting the

usual understanding of conjunction, disjunction and negation. A formula F is satisfiable if there is an interpretation I such that I maps F to true. Otherwise, a formula F is unsatisfiable. The Resolution Asymmetric Tautology (RAT) property is based on *asymmetric literal addition (ALA)* (Järvisalo, Heule, and Biere 2012):

$$\mathsf{ALA}_F(C) = C \cup \{\overline{L} \mid \{L_1, \ldots, L_n, L\} \in F \text{ and } \{L_1, \ldots, L_n\} \subseteq C\}$$

We consider the recursive application of ALA:

$$\begin{array}{lcl} \mathsf{ALA}_F(C) \uparrow 0 & = & C \\ \mathsf{ALA}_F(C) \uparrow n+1 & = & \mathsf{ALA}_F(\mathsf{ALA}_F(C) \uparrow n) \end{array}$$

A clause C is an *asymmetric tautology (AT)* w.r.t. the formula F if there is $n \in \mathbb{N}$ such that the clause $\mathsf{ALA}_F(C) \uparrow n$ is a tautology. Notice that there are small technical differences to the original version of ALA (Heule, Järvisalo, and Biere 2010) in the sense that we phrased their computation in terms of a mathematical function.

Example 1. *Consider the following formula*

$$F = \{\{p, q\}, \{p, \neg q, r\}, \{\neg q, \neg r\}\}$$

Note the following fixpoints of $\mathsf{ALA}_F(\{p\})$ *and* $\mathsf{ALA}_F\{q\}$*:*

$$\begin{array}{lcl} \mathsf{ALA}_F(\{p\}) \uparrow 0 & = & \{p\} \\ \mathsf{ALA}_F(\{p\}) \uparrow 1 & = & \{p, \neg q\} \\ \mathsf{ALA}_F(\{p\}) \uparrow 2 & = & \{p, \neg q, \neg r, r\} \\ \mathsf{ALA}_F(\{p\}) \uparrow 3 & = & \{p, \neg q, \neg r, r, q\} \\ \mathsf{ALA}_F(\{p\}) \uparrow 4 & = & \mathsf{ALA}_F(\{p\}) \uparrow 3 \end{array}$$

$$\begin{array}{lcl} \mathsf{ALA}_F(\{q\}) \uparrow 0 & = & \{q\} \\ \mathsf{ALA}_F(\{q\}) \uparrow 1 & = & \{q, \neg p\} \\ \mathsf{ALA}_F(\{q\}) \uparrow 2 & = & \mathsf{ALA}_F(\{q\}) \uparrow 1 \end{array}$$

The clause $\{p\}$ *is an AT in* F*, whereas clause* $\{q\}$ *is not.* □

Note that ALA is monotone in both arguments, and that replacing clauses by ALA preserves semantic equivalence. Learned clauses in CDCL SAT solvers are asymmetric tautologies (Beame, Kautz, and Sabharwal 2004) as well as *tautologies*, *resolvents*, and *subsumed clauses* (Järvisalo, Heule, and Biere 2012).

Järvisalo et al. introduced the following redundancy criterion based on asymmetric tautologies in (Järvisalo, Heule, and Biere 2012): The clause C is a *resolution asymmetric tautology (RAT) upon literal* L *w.r.t. the formula* F if *1.* the clause C is an asymmetric tautology w.r.t. the formula F, or *2.* the literal L occurs in C, and all resolvents of C with any clause $D \in F$ upon L are asymmetric tautologies w.r.t. the formula F.

Example 2. *The clauses* $\{p\}, \{\neg q\}, \{\neg r\}, \{q, r\}$ *are resolution asymmetric tautologies in the formula* F *from Example 1. 1.* $\{p\}$ *is a RAT upon* p *w.r.t.* F *because there is no clause* $D \in F$ *with* $\neg p \in D$*. 2.* $\{\neg q\}$ *is a RAT upon* $\neg q$*, because there is only one resolvent* $\{p\}$*, which is an AT in* F*. 3.* $\{\neg r\}$ *is a RAT upon* $\neg r$*, because there is only one resolvent* $\{p, \neg q\}$*, which is an AT because it is subsumed by the clause* $\{p\}$ *which is an AT in* F*. 4.* $\{q, r\}$ *is a RAT upon* q *because both resolvent* $\{p, r\}$ *and* $\{r, \neg r\}$ *are ATs.* □

Intuitively, a DRAT derivation is a finite sequence of addition and deletion instructions, where each added clause is a RAT w.r.t. the preceding clauses, except those that have been eliminated before. Formally, we consider labeled clauses, i.e., expressions of the form $(\mathsf{a}\, C)$ and $(\mathsf{d}\, C)$ representing clause addition and deletion, respectively. An empty sequence of labeled clauses is denoted by Λ. For a finite labeled clause sequence P and a formula F, we assign *associated formulas*, denoted by $\mathsf{AF}(F, P)$, as follows:

$$\begin{array}{lcl} \mathsf{AF}(F, \Lambda) & = & F \\ \mathsf{AF}(F, P\ (\mathsf{a}\, C)) & = & \mathsf{AF}(F, P) \cup \{C\} \\ \mathsf{AF}(F, P\ (\mathsf{d}\, C)) & = & \mathsf{AF}(F, P) \setminus \{C\} \end{array}$$

DRAT derivations in F are then defined inductively as follows: 1. The empty clause sequence is a DRAT derivation in F. 2. If P is a DRAT derivation in F, then $P\ (\mathsf{d}\, C)$ is a DRAT derivation in F. 3. If P is a DRAT derivation in F and C is a RAT w.r.t. $\mathsf{AF}(F, P)$, then $P\ (\mathsf{a}\, C)$ is a DRAT derivation in F. A *DRAT refutation* P *for the formula* F is a DRAT derivation in F such that we find that $\emptyset \in \mathsf{AF}(F, P)$. Analogously, we define a fragment of DRAT, called *Deletion Reverse Unit Propagation* (*DRUP*) refutations, in which every added clause is required to be an asymmetric tautology.

Example 3. *Consider the unsatisfiable formula*

$$F = \{\{p, q\}, \{\neg p, q\}, \{p, \neg q\}, \{\neg p, \neg q\}\}.$$

A DRAT refutation of F *is given by:*

$(\mathsf{a}\, \{\neg r\})$	*RAT upon the literal* $\neg r$
$(\mathsf{a}\, \{r, p\})$	*AT*
$(\mathsf{a}\, \{r\})$	*AT*
$(\mathsf{d}\, \{p, q\})$	*deletion*
$(\mathsf{d}\, \{\neg p, q\})$	*deletion*
$(\mathsf{d}\, \{p, \neg q\})$	*deletion*
$(\mathsf{d}\, \{\neg p, \neg q\})$	*deletion*
$(\mathsf{a}\, \{\,\})$	*AT*

This is a DRAT refutation, but not a DRUP refutation. The following is a DRUP refutation of F*:* $(\mathsf{a}\, \{p\})\ (\mathsf{a}\, \{\,\})$. □

A formula F is unsatisfiable if and only if there is a DRAT (DRUP, resp.) refutation of F. An advantage of DRAT over DRUP is that it allows exponentially shorter proofs for some fragments.

Verifying RAT Refutations with Deletion Information

Mechanical verification allows us to prove that a program meets its specification. Our specifications and proofs were carried out in the Coq proof assistant. Coq is based on the calculus of inductive constructions and combines higher-order logic with a typed functional programming language. Since 1984, its development is supported by INRIA. In Coq we define functions in the lambda calculus. Moreover, we can express mathematical theorems and can prove them interactively. The syntax of Coq is similar to that of other typed functional programming languages. Accepted Coq proofs can be independently checked by a small certification kernel. Finally, we can automatically extract Haskell programs from Coq theories.

Our work extends (Wetzler, Heule, and Hunt Jr 2013) that describes a RAT checker. Unfortunately, RAT checkers cannot check RAT refutations with deletion information as the following example demonstrates.

Example 4. *Consider the formula* $F = \{\{p, q\}\}$. *Then* $(\mathsf{d}\,\{p, q\})\ (\mathsf{a}\,\{\neg p\})$ *is a DRAT derivation in* F. *However,* $(\mathsf{a}\,\{\neg p\})$ *is not a DRAT derivation in* F, *because* $\{\neg p\}$ *is neither an AT in* F *nor a RAT because the resolvent* $\{q\}$ *is not an asymmetric tautology in* F. □

Our definitions and lemmas closely follow those presented in (Wetzler, Heule, and Hunt Jr 2013), i.e., clauses are represented by lists of literals, formulas by lists of clauses and we use the notion of reverse unit propagation used instead of ALA. We use the inductive definitions of DRAT derivation and refutation. The following theorem was expressed and proven in the Coq proof assistant:

Theorem 1. *For every formula* F *and labeled clause sequence* P, *if* P *is a DRAT refutation of* F, *then* F *is unsatisfiable. Moreover, it is decidable whether* P *is a DRAT refutation of* F.

To show the first part, we proof that for every DRAT derivation P of F, it follows that the unsatisfiability of $\mathsf{AF}(F, P)$ implies the unsatisfiability of F. Since a DRAT refutation is a DRAT derivation that contains the empty clause and the empty clause is unsatisfiable, we infer that the existence of a DRAT refutation for F implies unsatisfiability for F. The second part of the above theorem is shown in a straightforward way.

An executable Haskell program can be extracted from the decidability result. However, the resulting program is very inefficient. Therefore, we developed another Haskell program that is based on the automatically extracted one, but applies the full watcher scheme to improve the efficiency of unit propagation. Mutable data structures from the *vector* package were used. Preliminary experiments have shown that it is significantly faster than the automatically extracted one.

Fuzzing DRAT Refutations

Fuzzing is a technique that provides correct or invalid randomly-generated inputs to a computer program. We then observe whether the program's behavior is as expected. Our method is a six-stage process using a given checker and consists of the following steps:

Step 1 A randomly-generated formula F is constructed using external programs such as the C programs *cnfuzz* and *fuzzsat* (Brummayer, Lonsing, and Biere 2010).

Step 2 A SAT solver, such as *MiniSAT*, *Lingeling*, or *Riss* solves the satisfiability problem of F. In case F is reported satisfiable, we go back to step 1. Otherwise, the formula F is reported unsatisfiable, and a DRAT refutation P of F is provided by the SAT solver.

Step 3 The formula F and P are given to the checker and the tuned Haskell program.

Step 4 In case the two programs inconsistently classify P as accepted or rejected, we know that a bug in one of the checkers exists. Then, the mechanically verified checker decides which program has given the correct answer and our procedure terminates with a bug report. Otherwise, we continue with step 5.

Step 5 We modify the proof by randomly adding or removing a literal in a randomly selected clause. Notice that the addition of a literal in a clause C does not destroy the proof at this particular point, but at a later position where a clause depends on C (either in the AT or RAT computation). Eventually, the removal of a literal in a clause does destroy the refutation at exactly this position. In case the proof consists only of the empty clause, we add a literal to the empty clause.

Step 6 After that, we provide P' to the checkers. In case P' is accepted by both programs, P' is a DRAT refutation. Consequently, we go back to step 5. In case P' is rejected by both programs, we go back to step 1. Otherwise, the checkers behave inconsistently and we observe a potential bug.

Note that we do not introduce or remove deletion information, but instead remove the added clauses, which has the same effect.

Experimental Evaluation

We constructed formulas consisting of 2900 clauses and 800 variables in average, resulting in average proof lengths of 2100. The procedure revealed the following:

Tautological Clauses The checker *drat-trim* rejects DRAT refutations containing tautological clauses. Given some unsatisfiable formula F and a DRAT refutation P of F. Then, $P\ (\mathsf{a}\,\{p, \neg p\})$ is rejected by *drat-trim*. However, a tautological clause is a RAT w.r.t. any formula F.

Deletion of Units The checker *proofcheck* rejects DRAT refutations in which unit clauses were deleted. This can be explained as *proofcheck* assumes that unit clauses are not deleted. However, deleting unit clauses is in line with the specification and ignoring deletion information cannot be done in DRAT (see Example 4). The checker *drat-trim* shows a similar behavior: Suppose q is a variable not occurring in F and P is a DRAT refutation of a formula F. *drat-trim* rejects

$$\underbrace{(\mathsf{a}\,\{q\})\ (\mathsf{d}\,\{q\})\ (\mathsf{a}\,\{\neg q\})\ (\mathsf{d}\,\{\neg q\})}_{Q}\ P.$$

However, the given sequence is a DRAT refutation since $\mathsf{AF}(F, Q) = \emptyset$ and P is a DRAT refutation by assumption.

Proof Emission The DRAT emission procedure in the well-known SAT solver *MiniSAT*, that consists of few lines of code, constructs strange DRAT refutations of the form $P(\mathsf{a}\,\emptyset)(\mathsf{d}\,C)(\mathsf{a}\,\emptyset)$, where P is a DRAT derivation and C is some clause. In an updated version of *MiniSAT*, this error is already fixed. Nevertheless, other *MiniSAT*-based solvers still might contain this bug.

Conclusion

Modern SAT solvers are highly-tuned systematic search procedures that emit unsatisfiability proofs in the DRAT format to guarantee correctness of their answers. Unfortunately, available DRAT checkers contain bugs which might be due to the difficulties in understanding and debugging highly optimized C or C++ code. Our contribution to improve the confidence in DRAT checkers is a new fuzzing technique that automatically finds bugs in DRAT checkers. Our method generates unsatisfiable formulas and inserts errors into DRAT refutations. The modified sequences are then given to two distinct DRAT checkers, such as *drat-trim*, our optimized DRAT checker written in Haskell. In case of inconsistent behavior of the two checkers, we discovered a potential bug. The final decision which checker has given the correct answer is done by a mechanically verified DRAT checker written in Coq. Recently, an efficient *and* mechanically verified DRUP checker was developed (Cruz-Filipe, Marques-Silva, and Schneider-Kamp 2016), which applies *drat-trim* to detect relevant clauses in DRUP refutations. Consequently, they rely on the completeness of *drat-trim* in the sense that it rejects DRUP refutations that are rejected by *drat-trim*. In fact, our fuzzer has shown the incompleteness of *drat-trim* and *proofcheck*. Our verified DRAT checker extends earlier work (Wetzler, Heule, and Hunt Jr 2013; Darbari, Fischer, and Marques-Silva 2010) and, to the best of our knowledge, is the only available, complete and fully mechanically verified DRAT checker. The developed software is available at github.com/drat-tools. In the future, we adapt the verified checker and the fuzzing procedure to other proof formats, such as resolution proofs and IORUP.

Acknowledgements This work has been supported the Austrian National Research Network S11403-N23 (RiSE), the LogiCS doctoral program W1255-N23 of the Austrian Science Fund (FWF), and by the Vienna Science and Technology Fund (WWTF) through grant VRG11-005.

References

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 399–404. Morgan Kaufmann Publishers Inc.

Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligene Research* 22(1):319–351.

Biere, A.; Cimatti, A.; Clarke, E. M.; Fujita, M.; and Zhu, Y. 1999. Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of the 36th Conference on Design Automation*, 317–320.

Brummayer, R.; Lonsing, F.; and Biere, A. 2010. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing*, 44–57. Springer.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151–158. ACM.

Cruz-Filipe, L.; Marques-Silva, J.; and Schneider-Kamp, P. 2016. Efficient certified resolution proof checking. *CoRR* abs/1610.06984.

Darbari, A.; Fischer, B.; and Marques-Silva, J. 2010. Industrial-strength certified SAT solving through verified SAT proof checking. In *Proceedings of the Seventh International Colloquium on Theoretical Aspects of Computing*, 260–274. Springer.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem-proving. *Communications of the ACM* 5(7):394–397.

Eén, N., and Biere, A. 2005. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing*, 61–75. Springer.

Gelder, A. V. 2002. Extracting (easily) checkable proofs from a satisfiability solver that employs both preorder and postorder resolution. In *Proceedings of the Seventh International Symposium on Artificial Intelligence and Mathematics*.

Heule, M.; Järvisalo, M.; and Biere, A. 2010. Clause elimination procedures for CNF formulas. In *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, 357–371. Springer.

Heule, M. J. H.; Kullmann, O.; and Marek, V. W. 2016. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing*, 228–245. Springer.

Järvisalo, M.; Heule, M. J. H.; and Biere, A. 2012. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning*, 355–370. Springer.

Manthey, N., and Lindauer, M. 2016. SpyBug: Automated bug detection in the configuration space of SAT solvers. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing*, 554–561. Springer.

Manthey, N., and Philipp, T. 2015. Checking unsatisfiability proofs in parallel. In *Pragmatics of SAT 2015*. EasyChair.

Silva, J. P. M., and Sakallah, K. A. 1996. GRASP - a new search algorithm for satisfiability. In *Proceedings of the International Conference On Computer Aided Design*, 220–227. Washington: IEEE Computer Society.

Wetzler, N.; Heule, M. J.; and Hunt Jr, W. A. 2013. Mechanical verification of SAT refutations with extended resolution. In *Proceedings of the Fourth International Conference on Interactive Theorem Proving*, 229–244. Springer.

Wetzler, N.; Heule, M.; and Hunt Jr, W. A. 2014. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing*, 422–429. Springer.

Zhang, L., and Malik, S. 2003. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Design, Automation and Test in Europe Conference and Exposition*, 10880–10885. IEEE Computer Society.