DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs

Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt, Jr. *

The University of Texas at Austin

Abstract. The DRAT-trim tool is a satisfiability proof checker based on the new DRAT proof format. Unlike its predecessor, DRUP-trim, all presently known SAT solving and preprocessing techniques can be validated using DRAT-trim. Checking time of a proof is comparable to the running time of the proof-producing solver. Memory usage is also similar to solving memory consumption, which overcomes a major hurdle of resolution-based proof checkers. The DRAT-trim tool can emit trimmed formulas, optimized proofs, and new TraceCheck$^+$ dependency graphs. We describe the output that is produced, what optimizations have been made to check RAT clauses, and potential applications of the tool.

1 Introduction

The DRAT-trim satisfiability (SAT) proof checker and trimming utility is designed to validate proofs of unsatisfiability. DRAT-trim uses the new DRAT proof format which combines the previous DRUP and RAT formats [1,2] and allows all presently known SAT techniques to be expressed. The tool maintains the efficiency of its predecessor, DRUP-trim, while integrating stronger forms of clause redundancy.

SAT solvers, as well as SMT and QBF solvers, have documented bugs [3,4]. Jävisalo et. al. [5] described a subtle error in the blocked clause addition [6] routine of Lingeling [7], one of the best SAT solvers available. Despite this bug, Lingeling was claimed to be "experimentally correct" on millions of satisfiability benchmarks and was used by industry for over a year and half before the bug was discovered. The process inspired a proof system [5] designed to detect this and similar errors.

The DRAT-trim proof checker is based on this proof system and addresses the main limitation of contemporary proof checkers: they cannot validate blocked clause addition and other techniques such as extended resolution [8], extended learning [9], and bounded variable addition [10]. DRAT proofs are easy to emit, require relatively little space on disk, and can be used to check all known solving and preprocessing techniques. DRAT-trim validates unsatisfiability results in a time comparable with solving time and uses far less physical memory than previous checkers.

* The authors are supported by DARPA contract number N66001-10-2-4087.

Proof checking is just one function of the DRAT-trim tool; trimmed formulas, optimized proofs, and TraceCheck$^+$ dependency graphs can also be emitted. These extra results have a variety of applications. Trimmed formulas and dependency graphs can be used as input to MUS extraction tools, similar to the relationship of a preprocessor to a solver. Reduced proofs can speed up proof playback, a feature that may be leveraged by a mechanically-verified proof checking tool [11]. DRAT-trim is the first tool to emit these additional results for proofs generated by techniques such as blocked clause addition and extended resolution.

In the remainder of this tool paper, some of the technical details associated with producing proofs in the DRAT format are described in Section 2. The input formats are detailed in Section 3. Output formats and applications of the tool are presented in Section 4, and implementation details are discussed in Section 5. Finally, conclusions are drawn in Section 6.

2 Redundancy Properties

DRAT-trim is designed to accept proofs in a specific format, which will be described in the next section. The proof checking algorithms are based on clause redundancy properties presented in earlier work [1,2] and are briefly explained below. This brief paper does not contain preliminaries on CNF formulas, resolution, and unit propagation. Readers who are not familiar with these concepts are referred to our earlier work [12] which uses the same notation as this system description. This material is presented for completeness as the definition of the main clause redundancy property has changed slightly from earlier publications [2].

Definition 1 (Asymmetric Literal Addition (ALA) [12]). *Let C be a clause occurring in a CNF formula $F \cup \{C\}$. The clause $\mathsf{ALA}(F, C)$ is the unique clause obtained by applying the following extension rule until fixpoint:*

$$C := C \cup \{\bar{l}\} \textit{ if } \exists\, l_1, \ldots, l_k \in C, (l_1 \vee \ldots \vee l_k \vee l) \in F$$

A clause C has property AT (*Asymmetric Tautology*) with respect to CNF formula F if $\mathsf{ALA}(F, C)$ is a tautology. The property AT is also known as Reverse Unit Propagation (RUP) [13]. A clause C has property RAT (*Resolution Asymmetric Tautology*) with respect to CNF formula F if there exists a literal $l \in C$ such that for all $D \in F$ with $\bar{l} \in D$, it holds that $(D \setminus \{\bar{l}\}) \cup C$ has property AT with respect to F. Due to the monotonicity of asymmetric literal addition, a clause C with property AT with respect to formula F, has property RAT with respect to F. If a clause $C \notin F$ has property RAT with respect to F, then F and $F \cup \{C\}$ are satisfiability equivalent [2].

Example 1. Consider the CNF formula $F = (\bar{a} \vee b) \wedge (a \vee c) \wedge (b \vee \bar{c})$. The clause (a) has property RAT with respect to F, because $(a \vee b)$ has property AT with respect to F. Therefore, (a) can added to F, while preserving unsatisfiability. The clause (b) has property AT (and hence property RAT) with respect to F.

3 Input

DRAT-trim requires two input files: a formula and a proof. Formulas are expected to be in DIMACS CNF format—the conventional input format for SAT solvers. Proofs are expressed in the new DRAT (**D**eletion **R**esolution **A**symmetric **T**autology) notation, which is a generalization of the DRUP (**D**eletion **R**everse **U**nit **P**ropagation) format [1]. The DRAT format has the advantage that all presently-known techniques are expressible in this notation [2].

A mathematical proof of a theorem can be constructed from smaller theorems, or lemmas. We use this terminology to describe clausal proofs. Thus, clausal proofs are sequences of "lemma" additions and "lemma" or input clause deletions. More specifically, each line of the proof file is either a lemma (a sequence of literals terminated by `0`) or a deletion instruction (with a "`d `" prefix). Proof files terminate with the empty clause: a line containing only a zero. Let F be a CNF formula and P be a DRAT proof for F. The number of lines in a proof P is denoted by $|P|$. For each $i \in \{0, \ldots, |P|\}$, a CNF formula is defined F_P^i below. L_i refers to the lemma on line i of P.

$$F_P^i := \begin{cases} F & \text{if } i = 0 \\ F_P^{i-1} \setminus \{L_i\} & \text{if the prefix of } L_i \text{ is ``\texttt{d }''} \\ F_P^{i-1} \cup \{L_i\} & \text{otherwise} \end{cases}$$

Lemma addition steps are validated using both RUP and RAT checks. The RUP check for lemma L_i in proof P for CNF formula F succeeds if L_i has the property AT with respect to F_P^{i-1}. Let l_i denote the first literal in lemma L_i. The RAT check for lemma L_i in proof P for CNF formula F succeeds if and only if L_i has the property RAT on literal l_i with respect to F_P^{i-1}.

Note that the DRUP and DRAT formats are syntactically the same, but DRAT-trim checks each line of the proof for a stronger form of clause redundancy, RAT, when the RUP check has failed. Furthermore, a tool that emits DRUP proofs is compatible with DRAT-trim. Fig. 1 shows an example DIMACS CNF file by Rivest [14] and a DRAT proof file.

Emitting a DRAT proof from a conflict-driven clause-learning (CDCL) [15] solver and preprocessors is relatively easy. CDCL solvers maintain a database containing the original clauses and lemmas. Creating a DRAT proof is typically as simple as printing each modification to the database to a file. Several state-of-the-art SAT solvers support emitting DRUP proofs, the format used for the SAT Competition 2013 [16]. For example, Glucose 3.0 [17] supports emitting proofs in the DRAT format with "`-certified`" and "`-certified-output=`" options.

4 Output and Applications

The default output of the DRAT-trim tool is a message indicating the validity of a proof with respect to an input formula. This information can be very useful while developing SAT solvers, especially since the DRAT format supports checking all existing techniques used in state-of-the-art SAT solvers [2].

CNF formula

```
p cnf 4 10
 1  2 -3 0
-1 -2  3 0
-1 -2 -3 0
 2  3 -4 0
-2 -3  4 0
-1 -3 -4 0
 1  3  4 0
-1  2  4 0
 1 -2 -4 0
-1  2 -4 0
```

DRAT proof

```
  -1 0
d -1 2 4 0
   2 0
   0
```

Fig. 1. DRAT-trim accepts two files as input: a formula in DIMACS CNF format (left), and a proof in the new DRAT format (right). Each line in the proof is either a lemma or a deletion step identified by the prefix "`d` ". Spacing in both examples is to improve readability. If the RUP check fails, the DRAT-trim tool expects that a lemma has RAT on its first literal, as in the RAT format [2]. No check is performed for clause deletion steps, although the deleted clause needs to be present.

Additionally, SAT competitions can benefit from having a fast proof checker with an easy-to-produce input format. As mentioned in Section 1, SAT solvers have been shown to be buggy, even during competitions. DRUP-trim, the predecessor of DRAT-trim, was used to check the results of the unsatisfiability tracks of the SAT Competition 2013 [16]. Some techniques, such as bounded variable addition [10], cannot be validated using only RUP checks, and hence cannot be checked by DRUP-trim. The new DRAT-trim tool now supports all these techniques. Aside from checking the validity of proofs, DRAT-trim can optionally produce three outputs: a *trimmed formula*, an *optimized proof*, and a *dependency graph* in the new TraceCheck$^+$ format. These output files are described below and illustrated in Figure 2.

trimmed formula

```
p cnf 4 8
 1  2 -3 0
-1 -2  3 0
 2  3 -4 0
-2 -3  4 0
-1 -3 -4 0
 1  3  4 0
-1  2  4 0
 1 -2 -4 0
```

optimized DRAT proof

```
  -1 0
d -1 -2  3 0
d -1 -3 -4 0
d -1  2  4 0
   2 0
d  1  2 -3 0
d  2  3 -4 0
   0
```

TraceCheck$^+$ file

```
 1  1  2 -3  0  0
 2 -1 -2  3  0  0
 4  2  3 -4  0  0
 5 -2 -3  4  0  0
 6 -1 -3 -4  0  0
 7  1  3  4  0  0
 8 -1  2  4  0  0
 9  1 -2 -4  0  0
11 -1  0  2  6  8  0
12  2  0  1  4  7 11 0
13  0  5  7  9 11 12 0
```

Fig. 2. Three optional output files from DRAT-trim for the input formula and proof from Fig. 1. A trimmed formula (left) is an ordered subset of the input formula. An optimized proof (middle) is an ordered subset of the input proof with extra deletion instructions. A TraceCheck$^+$ file (right) is a dependency graph that includes the formula and proof. Each line begins with a clause identifier (bold), then contains the literals of the original clause or lemma, and ends with a list of clause identifiers (bold).

Trimmed Formula. The trimmed formula produced by DRAT-trim is a subset of the input formula in DIMACS format, and the remaining clauses appear in the same order as the input file. However, the order of the literals in each clause may have changed. Trimming a formula can be a useful preprocessing step in extracting a Minimal Unsatisfiable Subset (MUS). Since DRAT-trim supports validating techniques that are based on (generalizations of) extended resolution, one can use these techniques to improve MUS extraction tools.

Optimized Proof. The optimized proof contains lemmas as well as deletion information in the DRAT format. Lemmas appearing in the optimized proof will be an ordered subset of the lemmas in the input proof. The first literal of each lemma is the same as the first literal of that lemma in the input proof, however the order of all other literals for the lemma may be permuted. The output proof is called "optimized" because it contains extra deletion information that is obtained during the backward checking process, described in Section 5. The optimized proof file may be larger than the input proof in size (as in Figure 2); however, the additional deletion information helps reduce computation time because fewer clauses are active during each check. A smaller proof is very useful for potentially slower, mechanically-verified solvers. The optimized proof from one round of checking is often much smaller than the original, but it is still far from minimal. One can iteratively apply the checking process to further optimize a proof by submitting the reduced output proof of one round of checking to the tool for another round of checking and trimming.

Dependency Graph. DRUP-trim, the predecessor to this tool, can produce a resolution graph of a proof in the TraceCheck [18,19,20] format. DRAT-trim can, however, validate techniques that cannot be checked with resolution, and we designed a new format that is backward-compatible with TraceCheck, allowing expression of all presently-known solving techniques. Resolution graphs in the TraceCheck format begin each line with a clause identifier, followed by the literals of the clause, a zero, the identifiers of antecedents, finally, followed by another zero. The new TraceCheck$^+$ format uses this syntax as well. The formats only differ in the expressing the reasons for a lemma's redundancy. If the RUP check succeeds, the reasons are the antecedents as in the TraceCheck format. If a RAT check is required, the reasons are the clauses required to let the RAT check succeed.

Dependency graphs have many potential uses. The dependency graph may be supplied to a MUS extractor as input, avoiding the need to recompute clause dependencies. Another use is for solver debugging: a dependency graph gives a step-by-step account of why each clause can be added to a clause database.

5 Methodology and Optimizations

DRAT-trim uses backward checking [21] to track and limit dependencies between clauses and lemmas; this means that lemmas are checked in reverse of the order they occur in the original proof. As each lemma is checked, any clauses or lemmas that were crucial to the check are marked. Marked clauses are preferred over unmarked clauses during unit propagation [1]. The process of marking not only keeps the trimmed formula and optimized proof small, but also reduces the number of lemmas that need to be validated, lowering the run time of the tool.

Clauses are marked by a conflict analysis routine that runs after unit propagation derives a conflict. When a clause is marked, the timestamp and location of the clause are stored as deletion information. The first time a clause is marked (and is determined to be necessary to the proof) during backward checking is the last time a clause is used during forward checking or optimized proof emission. Therefore, as each clause is marked, optimized deletion information is stored. During post-processing, marked lemmas are printed in order and deletion information sharing the same timestamp is printed afterwards.

During a RAT check, DRAT-trim needs access to all clauses containing the negation of the resolution literal. One could build a literal-to-clause lookup table of the original formula and update it after each lemma addition and deletion step. However, these updates can be expensive and the lookup table potentially doubles the memory usage of the tool. Since most lemmas emitted by state-of-the-art SAT solvers can be validated using the RUP check, such a lookup table has been omitted. Instead, when a RUP check failed, the currently active formula is scanned to find all clauses containing the complement of the resolution literal.

6 Performance and Conclusion

The DRAT-trim tool outperforms its RAT checker [2] and DRUP-trim [1] predecessors. To illustrate the performance gain, consider the rbcl_xits_09_unknown benchmark, one of the harder benchmarks [2]. The solving time with Coprocessor and Glucose 3.0 is 95 seconds and the proof checking time with DRAT-trim is 91 seconds, compared to 1096 seconds for a previous RAT proof checker. This particular benchmark can only be solved with bounded variable addition, necessitating a RAT/DRAT proof. On the application suite of SAT Competition 2009, DRAT-trim is 2% faster than DRUP-trim on average, within a range of 85% faster to 13% slower. Thus, the addition of RAT checking in DRAT-trim does not have a noticeable impact on DRUP proof checking.

DRAT-trim[1] and its associated DRAT proof format can be used to validate all contemporary SAT pre-processing and solving techniques with similar time and memory consumption to solving. Resolution-based proof checkers are not subsumed by this effort, but the performance of DRAT-trim appears better on large, industrial-scale examples. Our tool can optionally emit trimmed formulas, optimized proofs, and dependency graphs, and we look forward to seeing creative utilization of these resources by the SAT community. We believe that all SAT solvers should emit proofs in the DRAT format so that their results can be validated. Not only does this provide implementers with a convenient way of debugging, but it gives users the confidence to find new applications for satisfiability solvers.

[1] DRAT-trim is available at `http://cs.utexas.edu/~marijn/drat-trim/`.

References

1. Heule, M.J.H., Hunt, Jr., W.A., Wetzler, N.: Trimming while checking clausal proofs. In: Formal Methods in Computer-Aided Design (FMCAD), IEEE (2013) 181–188
2. Heule, M.J.H., Hunt, Jr., W.A., Wetzler, N.: Verifying refutations with extended resolution. In: International Conference on Automated Deduction (CADE). Volume 7898 of LNAI., Springer (2013) 345–359
3. Brummayer, R., Biere, A.: Fuzzing and delta-debugging SMT solvers. In: International Workshop on Satisfiability Modulo Theories (SMT), ACM (2009) 1–5
4. Brummayer, R., Lonsing, F., Biere, A.: Automated testing and debugging of SAT and QBF solvers. In: Theory and Applications of Satisfiability Testing (SAT). Volume 6175 of LNCS., Springer (2010) 44–57
5. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: International Joint Conference on Automated Reasoning (IJCAR). Volume 7364 of LNCS., Springer (2012) 355–370
6. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics **96-97** (1999) 149–176
7. Biere, A.: Lingeling, Plingeling, and Treengeling. `http://fmv.jku.at/lingeling/` (2014) Accessed: 2014-01-27.
8. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In Siekmann, J., Wrightson, G., eds.: Automation of Reasoning 2. Springer-Verlag (1983) 466–483
9. Audemard, G., Katsirelos, G., Simon, L.: A restriction of extended resolution for clause learning SAT solvers. In Fox, M., Poole, D., eds.: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI), AAAI Press (2010)
10. Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of boolean formulas. In: Proceedings of Haifa Verification Conference (HVC). Volume 6397 of LNCS., Springer (2012) 102–117
11. Wetzler, N., Heule, M.J.H., Hunt, Jr., W.A.: Mechanical verification of SAT refutations with extended resolution. In: Conference on Interactive Theorem Proving (ITP). Volume 7998 of LNCS., Springer (2013) 229–244
12. Heule, M.J.H., Järvisalo, M., Biere, A.: Clause elimination procedures for CNF formulas. In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Volume 6397 of LNCS., Springer (2010) 357–371
13. Van Gelder, A.: Verifying RUP proofs of propositional unsatisfiability. In: International Symposium on Artificial Intelligence and Mathematics (ISAIM). (2008)
14. Rivest, R.L.: Partial-match retrieval algorithms. SIAM J. Comput. **5**(1) (1976) 19–50
15. Marques-Silva, J.P., Lynce, I., Malik, S.: Chapter 4, Conflict-Driven Clause Learning SAT Solvers. In: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (February 2009) 131–153
16. Balint, A., Belov, A., Heule, M., Järvisalo, M.: SAT Competition 2013. `http://www.satcompetition.org/2013/` (2013) Accessed: 2014-01-21.
17. Audemard, G., Simon, L.: Glucose's home page. `http://www.labri.fr/perso/lsimon/glucose/` (2014) Accessed: 2014-01-21.
18. Sinz, C., Biere, A.: Extended resolution proofs for conjoining BDDs. In Grigoriev, D., Harrison, J., Hirsch, E.A., eds.: Computer Science – Theory and Applications. Volume 3967 of LNCS., Springer (2006) 600–611

19. Jussila, T., Sinz, C., Biere, A.: Extended resolution proofs for symbolic SAT solving with quantification. In Biere, A., Gomes, C.P., eds.: Theory and Applications of Satisfiability Testing (SAT). Volume 4121 of LNCS., Springer (2006) 54–60

20. Biere, A.: PicoSAT essentials. Journal on Satisfiability, Boolean Modeling and Computation (JSAT) **4** (2008) 75–97

21. Goldberg, E.I., Novikov, Y.: Verification of proofs of unsatisfiability for CNF formulas. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), IEEE (2003) 10886–10891