Beyond DRAT: Challenges in Certifying UNSAT*

Bertram Felgenhauer

University of Innsbruck
bertram.felgenhauer@uibk.ac.at

Abstract

Contemporary SAT solvers are complex tools and may contain bugs. In order to increase the trust in the results of SAT solving, SAT solvers may produce certificates that can be checked independently. For unsatisfiability certificates, DRAT is the de facto standard. As long as the checkers are not formalized, extending the certificate format would decrease the trust in the checker because its code complexity increases. With the advent of formally verified checkers for DRAT proofs, this is no longer the case. In this note I argue that symmetry breaking is not adequately supported by DRAT proofs, and propose to have dedicated certification for symmetry breaking instead.

1 Introduction

Modern SAT solvers are powerful tools with many applications, for example in software and hardware verification and for combinatorial optimization and designs. Because SAT solvers have become quite complex and applications serious, certification of their output is very important. In fact, the SAT competitions since 2013 have required that solvers produce satisfiability and unsatisfiability certificates in the main track. Since 2014, the DRAT format (Wetzler et al. [8]) is the de facto standard for unsatisfiability certificates.[1]

Before this year (2017), the unsatisfiability certificates were checked by an independent program, DRAT-trim, which needed to be trusted. Now we have hybrid verifiers (by Cruz-Filipe et al. [1] and Lammich [6], both to be presented at CADE-26), which are based on an untrusted preprocessor that trims and annotates a DRAT proof, followed by a formally verified (in ACL2 [1] respectively Isabelle [6]) checker that certifies that the preprocessed proof establishes unsatisfiability of the input problem. Compared to DRAT-trim, which is written in C, this increases the level of trust significantly.

One key advantage of the DRAT format is that it covers most preprocessing (and inprocessing) techniques used in SAT solving [5] in a straightforward manner. I discuss a notable exception, symmetry breaking, in Section 2. My thesis, which I elaborate in Section 3, is that for techniques like symmetry breaking, encoding proofs into DRAT is inferior to having a standalone certifier or an extension of the DRAT format for that purpose.

*This work was supported by Austrian Science Fund (FWF) project P27528

[1] In principle, a DRAT proof is a sequence of additions and deletion of clauses, and each step can be checked rapidly. In practice, proofs are generally checked backwards for efficiency reasons, and verifiers like DRAT-trim are not *obviously correct*.

2 Symmetry Breaking and DRAT

A SAT instance is a propositional formula in conjunctive normal forms; the objective is to find an assignment of the propositional variables to true or false such that the formula becomes true. Many SAT instances (for example those arising from graph coloring) exhibit symmetries: permuting the variables of an assignment and possibly changing their polarity results in another satisfying assignment. For example, if $\alpha : \{x, y\} \to \{\mathsf{T}, \mathsf{F}\}$ is a satisfying assignment of the CNF $F = (x \vee \neg y) \wedge (\neg x \vee y)$, then $\alpha' = \{x \mapsto \neg\alpha(y), y \mapsto \neg\alpha(x)\}$ satisfies F as well. The presence of symmetries may cause an exponential blowup in the size of the SAT solver's search tree[2], and a similar blowup in the size of the resulting unsatisfiability certificates. In order to break such symmetries, one can add so-called *lex-leader* constraints to the SAT instance that assert that applying a given symmetry to the assignment represented in the CNF does not produce a smaller assignment (see, for example, Devriendt et al. [2]). The resulting formula is equisatisfiable to the original one, (if there is a satisfying assignment, but a lex-leader constraint for a particular symmetry is violated, we may switch to the smaller satisfying assignment induced by the symmetry; this process must terminate). The lex-leader constraints have the effect of pruning the search tree, often resulting in faster solving time and smaller certificates.

The main point concerning symmetry breaking for this note is that symmetry breaking relies on picking a lexicographically smallest assignment among a set of assignments that are symmetric to a given one, by an iterative process that may completely change the assignment.

In contrast, each step of a DRAT proof may change at most one literal of a satisfying assignment. Nevertheless, Heule et al. [3] showed that it is possible to encode (some) symmetry breaking with DRAT proofs, deriving lex-leader constraints. In a nutshell, to break a single symmetry (which is restricted to be an involution), one first defines a propositional variable z that is true if the desired lex-leader constraint is violated, then defines copies for each variable affected by the symmetry as either the original variable, if z is false, or the variable under the symmetry, if z is true. One then derives, for each clause, the clause obtained by replacing each variable by the corresponding copy, and the lex-leader constraint. Finally, one can delete the original clauses and the clauses defining z. All these steps can be encoded without too much difficulty in DRAT, but overall the process is anything but straightforward. Note that this is just for breaking a single symmetry; handling multiple symmetries is even more involved.

Remark 1. The polynomial time checkable *propagation redundancy* (PR) property [4] (to be presented at CADE-26) is more versatile than the RAT property, but still quite restrictive: If a clause C is added to a CNF F by PR, and C is violated by an assignment α satisfying F, then α can be modified by updating a fixed set of variables to predetermined values to obtain an assignment satisfying $F \wedge C$. (The updated variables and their values are given by ω in [4, Theorem 1]). Given this restriction to predetermined values, it is likely that even when using PR inferences, the variables and clauses have to be copied to perform a permutation of the variables.

3 Proposal

As we have seen in Section 2, there is a mismatch between DRAT, which is good at *local* modification of satisfying assignments that change one variable at a time, and *global* techniques

[2] In DPLL solvers with learning, the search tree gives rise to a resolution proof of unsatisfiability that is of similar size as the search tree; it is known that for the pigeon hole principle (which exhibits many symmetries), any resolution proof is of exponential size [7]. With symmetry breaking, polynomial sized proofs exist.

like symmetry breaking that rely on changing assignments in a holistic way, for example by focusing on the lexicographically smallest one.

One motivation of encoding symmetry breaking in DRAT is that this way, only a DRAT verifier needs to be trusted. But for a formally verified checker, the trust is not based on the simplicity and maturity of the code, but on having a machine checked correctness proof. Therefore, incorporating checking and breaking of symmetries (and derivation of the corresponding lex-leader constraints) can be accomplished without diminishing the trust.

I believe that there is value in direct certification of *global* techniques. This could be done by separate certifiers, or by extending the DRAT format and the corresponding verification toolchains. For symmetry breaking, the main benefit would be that producing certificates becomes much easier and more compact than encoding them in DRAT format.

As a minimal proposal, consider breaking syntactical symmetries[3]. In that case, the certificate contains a list of permutations, and list of literals defining a lexicographic order of assignments. The certifier would check that the permutations are indeed syntactic symmetries of the input formula, and add lex-leader constraints corresponding to the given permutations in a predetermined order and encoding. I expect that if a solver wants to use a different encoding, it can produce additional DRAT steps to derive it.

Open Questions.

- Which *global* techniques would you like to have support for? Some candidates are symmetry breaking, cutting planes for pseudo-boolean constraints (with cardinality reasoning as a special case), and Gaussian elmination for XOR constraints.
- Should such extensions be handled by standalone checkers or integrated into existing DRAT checkers? The latter approach may pave the way to treating *dynamic* symmetry breaking as well.
- Are symmetries (i.e., permutation of literals) the right level of abstraction? In principle, any map that produces satisfying assignments from other satisfying assignments gives rise to a lex-leader constraint.
- Technical details. What certificate format should be used? How should one handle the fresh variables from the lex-leader constraints?

References

[1] L. Cruz-Filipe, M. Heule, W.A. Hunt, M. Kaufmann, and P. Schneider-Kamp. Efficient certified RAT verification. In *Proc. 26th International Conference on Automated Deduction*, 2017. To appear.

[2] J. Devriendt, B. Bogaerts, M. Bruynooghe, and M. Denecker. Improved static symmetry breaking for SAT. In *Proc. 19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016)*, volume 9710 of *Lecture Notes in Computer Science*, pages 104–122, 2016.

[3] M. Heule, W.A. Hunt, and N. Wetzler. Expressing symmetry breaking in DRAT proofs. In *Proc. 24th International Conference on Automated Deduction*, volume 9195 of *Lecture Notes in Computer Science*, pages 591–606, 2015.

[4] M. Heule, B. Kiesl, and A. Biere. Short proofs without new variables. 2017. To appear.

[3] A syntactical symmetry is one that if applied to the input formula under consideration, results in the same formula again, modulo associativity and commutation of $\vee$ and $\wedge$.

[5] M. Järvisalo, M.J.H. Heule, and A. Biere. Inprocessing rules. In *Proc. 6th International Joint Conference on Automated Reasoning*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370, 2012.

[6] P. Lammich. Efficient verified (UN)SAT certificate checking. In *Proc. 26th International Conference on Automated Deduction*, 2017. To appear.

[7] T. Pitassi, P. Beame, , and R. Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity*, 3(2):97–140, 1993.

[8] N. Wetzler, M.J.H. Heule, and W.A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Proc. 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429, 2014.