EPiC Series in Computing
Volume 51, 2017, Pages 46–50
ARCADE 2017. 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements

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 their results, 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 solvers to produce satisfiability and unsatisfiability certificates in the main track. Since 2014, the DRAT format (Wetzler et al. [10]) is the de facto standard for unsatisfiability certificates.[1]

Before this year (2017), the unsatisfiability certificates had to be checked by an independent program, DRAT-trim, which needed to be trusted. Now we have hybrid verifiers (by Cruz-Filipe et al. [2] and Lammich [8], both presented at CADE-26), which are based on an untrusted preprocessor that trims and annotates a DRAT proof, followed by a formally verified (in Coq or ACL2 [2] respectively Isabelle [8]) 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, without significantly impact on the verification time.

A summary of the DRAT format is given in Section 2. One key advantage of the DRAT format is that it is very simple and nevertheless covers most preprocessing (and inprocessing) techniques used in SAT solving [7] in a straightforward manner. I discuss a notable exception, symmetry breaking, in Section 3. My thesis, which I elaborate in Section 4, is that for techniques

[*]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*.

like symmetry breaking, encoding proofs into DRAT is inferior to having a standalone certifier or an extension of the DRAT format for that purpose. Finally, Section 5 discusses the proposal, based on the discussion during ARCADE.

2 Unsatisfiability and DRAT

A SAT instance is a propositional formula in conjunctive normal forms (CNFs); the objective is to find an assignment of the propositional variables to true or false such that the formula becomes true, or to demonstrate that no such assignment exists. Formally, a CNF F is a finite set of clauses, where each clause C is a finite set of literals, and each literal is either a variable $x, y, \ldots \in X$ or a negation of a variable $\neg x, \neg y, \ldots$. Given a valuation $\alpha : X \to \{\mathsf{T}, \mathsf{F}\}$, which we extend to literals by letting $\alpha(\neg x) = \neg\alpha(x)$, the value of F is given by

$$\alpha(F) = \bigwedge_{C \in F} \bigvee_{l \in C} \alpha(l).$$

The CNF F is satisfiable if $\alpha(F) = \mathsf{T}$ for some assignment α; otherwise, it is unsatisfiable. It is very easy to verify satisfiability; all that is required is a satisfying assignment.

For certifying unsatisfiabilty, we need to prove the absence of satisfying assignments. DRAT is a clausal proof format which works by modifying the CNF while maintaining satisfiability, until a CNF containing the empty clause is reached, which is trivially unsatisfiable. Each step consists of either deleting a clause, which trivially preserves satisfiability, or adding a clause that is a *resolution asymmetric tautology* (RAT), which ensures that satisfiability is preserved. We extract the following observation from the proof that adding a RAT clause preserves satisfiability (cf. Järvisalo et al. [7, Proposition 1])

Observation 1. *If α is a satisfying assignment for F and C has RAT on $l \in C$ w.r.t. F, then either α is a satisfying assignment for $F \cup \{C\}$, or we obtain a satisfying assignment from α by assigning the literal l the value T in α.*

As a consequence of this observation, any clause addition step of a DRAT proof can update at most one value of a satisfying assignment. (This restriction is not as severe as it may sound, because clause additions can be used to define new variables that do not occur in F, covering extended resolution [4].)

Remark 1. The polynomial time checkable *propagation redundancy* (PR) property [6] (presented at CADE-26) is more versatile than the RAT property. Nevertheless, the way in which assignments are affected is still rather restricted. To wit, 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 the proof [6, Theorem 1].)

3 Symmetry Breaking and DRAT

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. [3]). The resulting formula is equisatisfiable to the original one, i.e., 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. As a concrete example, the CNF $F = (x \vee \neg y) \wedge (\neg x \vee y)$ allows adding a lex-leader constraint $x \vee y$ (this is equivalent to $(x, y) \geq_{\text{lex}} (\neg y, \neg x)$), but both x and y need to be updated simultaneously in order to obtain a satisfying assignment for $F \wedge (x \vee y)$ when starting with the satisfying assignment $\alpha = \{x \mapsto \mathsf{F}, y \mapsto \mathsf{F}\}$.

Recall that each step of a DRAT proof may change at most one literal of a satisfying assignment (cf. Observation 1), so the lex-leader constraint cannot be added directly with DRAT steps. Nevertheless, Heule et al. [5] showed that it is possible to encode (some) symmetry breaking with DRAT proofs, deriving lex-leader constraints for copies of the existing clauses using fresh variables. 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 2. As we saw in Remark 1, the PR property is less restricted than the RAT property. In fact, it is possible to update $\alpha = \{x \mapsto \mathsf{F}, y \mapsto \mathsf{F}\}$ to $\alpha = \{x \mapsto \mathsf{T}, y \mapsto \mathsf{T}\}$ in a single step. (A similar update is performed in the encoding of the pigeon hole principle using PR steps in [6].) However, this does not scale to large symmetries; a simultaneous swap of $x_1, \ldots, x_n$ with $y_1, \ldots, y_n$ without using fresh variables would have to be done one assignment at a time, and therefore require an exponential number of PR steps.

4 Proposal

As we have seen in Section 3, there is a mismatch between DRAT, which is good at *local* modification of satisfying assignments that change one variable at a time and at making inferences that do not change the satisfying assignment at all, and *global* techniques 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

[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 [9]. With symmetry breaking, polynomial sized proofs exist.

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.

5 Discussion

Extending the DRAT format has costs and benefits, which should be weighed. For each extension, a format needs to be specified, checkers have to be extended or possibly new checkers written, and solvers respectively preprocessors that want to use the extension have to be adapted.

There is also the question, raised by M. Heule during the discussion at ARCADE, whether DRAT should be extended at all. Currently, a great advantage of the DRAT format is that it is a very simple, fixed format. If we start extending the format, we move away from this sweet spot and there is a danger that the process of adding extensions will never stop. There have been requests to extend the DRAT format in the past. These requests concerned

1. symmetry breaking (as discussed above);
2. cutting planes for cardinality and pseudo-boolean constraints; and
3. Gaussian elimination for XOR constraints.

I believe that symmetry breaking stands out from this list in two ways. First, it is applicable to plain CNF, while a proper treatment of cutting planes and Gaussian elimination would require dealing with pseudo-boolean or XOR constraints, or somehow recognizing their encodings in a CNF. Secondly, the latter two extensions are about inferences that do not affect satisfying assignments, so we can realistically expect that they can be encoded in DRAT economically. In contrast, as argued earlier, symmetry breaking requires making copies of all affected clauses with new variables, which is expensive.

A final question that should be asked of extensions is whether they are useful. In the case of symmetry breaking, BreakIDCOMiniSatPS won No-Limit track in SAT 2016 competition [1] but didn't participate in the main track because no DRAT output for symmetry breaking was implemented, presumably because the procedure for doing that is very complex. This indicates that dedicated support for symmetry breaking would be helpful.

[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 $\wedge$.

References

[1] T. Balyo, M. Heule, and M. Järvisalo. SAT competition 2016: Recent developments. In *Proc. 33rd AAAI Conference on Artificial Intelligence*, pages 5061–5063, 2017.

[2] 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*, pages 220–236, 2017.

[3] 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.

[4] M. Heule, W.A. Hunt, and N. Wetzler. Verifying refutations with extended resolution. In *Proc. 24th International Conference on Automated Deduction*, pages 345–359, 2014.

[5] 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.

[6] M. Heule, B. Kiesl, and A. Biere. Short proofs without new variables. In *Proc. 26th International Conference on Automated Deduction*, pages 130–147, 2017.

[7] 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.

[8] P. Lammich. Efficient verified (UN)SAT certificate checking. In *Proc. 26th International Conference on Automated Deduction*, pages 237–254, 2017.

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

[10] 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.