Improved Single Pass Algorithms for Resolution Proof Reduction (poster presentation)

Ashutosh Gupta

IST, Austria

An unsatisfiability proof is a series of applications of proof rules on an input formula to deduce *false*. Unsatisfiability proofs for a Boolean formula can find many applications in verification. For instance, one application is automatic learning of abstractions for unbounded model checking by analyzing proofs of program safety for bounded steps [6, 5, 4]. We can also learn unsatisfiable cores from unsatisfiability proofs, which are useful in locating errors in inconsistent specifications [10]. These proofs can be used by higher order theorem provers as sub-proofs of another proof [2].

One of the most widely used proof rules for Boolean formulas is the resolution rule, i.e., if $a \vee b$ and $\neg a \vee c$ holds then we can deduce $b \vee c$. In the application of the rule, a is known as *pivot*. A *resolution proof* is generated by applying resolution rule on the clauses of an unsatisfiable Boolean formula to deduce *false*. Modern SAT solvers (Boolean satisfiability checkers) implement some variation of DPLL that is enhanced with conflict driven clause learning [9, 8]. Without incurring large additional cost on the solvers, we can generate a resolution proof from a run of the solvers on an unsatisfiable formula [11].

Due to the nature of the algorithms employed by SAT solvers, a generated resolution proof may contain redundant parts and a strictly smaller resolution proof can be obtained. Applications of the resolution proofs are sensitive to the proof size. Since minimizing resolution proofs is a hard problem [7], there has been significant interest in finding algorithms that partially minimize the resolution proofs generated by SAT solvers.

In [1], two low complexity algorithms for optimizing the proofs are presented. Our work is focused on one of the two, namely Recycle-Pivots. Lets consider a resolution step that produces a clause using some pivot p. The resolution step is called *redundant* if each deduction sequence from the clause to *false* contains a resolution step with the pivot p. A redundant resolution can easily be removed by local modifications in the proof structure. After removing a redundant resolution step, a strictly smaller proof is obtained. Recycle-Pivots traverses the proofs single time to remove the redundant resolutions partially. From each clause, the algorithm starts from the clause and follows the deduction sequences to find equal pivots. The algorithm stops looking for equal pivots if it reaches to a clause that is used to deduce more than one clause.

In this work, we developed *three algorithms* that are improved version of Recycle-Pivots. For the first algorithm, we observe that each literal from a clause must appear as a pivot somewhere in all the deduction sequences from

the clause to *false*. Therefore, we can extend search of equal pivots among the literals from the stopping clause without incurring additional cost. For the second algorithm, we observe that the condition for the redundant resolutions can be defined recursively over the resolution proof structure. This observation leads to a single pass algorithm that covers even more redundancies but it requires an expensive operation at each clause in a proof. Note that the second algorithm does not remove all such redundancies because the removal of a redundancy may lead to exposure of more. Our third algorithm is parametrized. This algorithm applies the expensive second algorithm only for the clauses that are used to derive a number of clauses smaller than the parameter. The other clauses are handled as in the first algorithm. The parametrization reduces run time for the third algorithm but also reduces the coverage of the redundancy detection.

We have implemented our algorithms in OpenSMT [3] and applied them on unsatisfiable proofs of 198 examples from plain MUS track of SAT11 competition. The original algorithm removes 11.97% of clauses in the proofs of the examples. The first and the second algorithm additionally remove 0.89% and 10.57% of the clauses respectively. The third algorithm removes almost as many clauses as the second algorithm in lesser time for the parameter value as low as 10. We also observe similar pattern in reduction of the unsatisfiable cores of the examples.

References

1. O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman. Linear-time reductions of resolution proofs. In *Haifa Verification Conference*, 2008.
2. S. Böhme and T. Nipkow. Sledgehammer: Judgement day. In J. Giesl and R. Hähnle, editors, *Automated Reasoning (IJCAR 2010)*, volume 6173 of *LNCS*, pages 107–121. Springer, 2010.
3. R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich. The opensmt solver. volume 6015, pages 150–153. Springer, 2010.
4. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *POPL*, 2004.
5. K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.
6. K. L. McMillan and N. Amla. Automatic abstraction without counterexamples. In *TACAS*, pages 2–17, 2003.
7. C. H. Papadimitriou and D. Wolfe. The complexity of facets resolved. *J. Comput. Syst. Sci.*, 37(1):2–13, 1988.
8. J. P. M. Silva, I. Lynce, and S. Malik. Conflict-driven clause learning sat solvers. In *Handbook of Satisfiability*, pages 131–153. 2009.
9. J. P. M. Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.
10. C. Sinz, A. Kaiser, and W. Küchlin. Formal methods for the validation of automotive product configuration data. *AI EDAM*, 17(1):75–97, 2003.
11. L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formulas. In *SAT*, 2003.