Towards the extraction of clause learning

Ulrich Berger Andrew Lawrence Monika Seisenberger

July 17, 2020

The Boolean satisfiability problem (SAT) consists in deciding whether a propositional formula in conjunctive normal form (CNF) has a satisfying assignment. Since SAT solvers play an increasingly important role in scientific and industrial contexts, big efforts are undertaken to improve their performance. With the increasing level of sophistication of SAT algorithms [6, 1], the question of their correctness becomes a more and more pressing issue. To address this, SAT competitions now usually require of SAT solvers to include in the case of an unsatisfiability result a certificate for its correctness. Creating such certificates may require considerable effort as the example of the recent solution of the Pythagorean triples problem by a SAT solver shows [5]. Still, such certificates only guarantee the correctness of single results but not the overall correctness of the SAT solver. So far, only a few SAT solvers have been formally verified [8, 9]. In earlier work, instead of posthumous verification, we have shown how such a SAT algorithm can be extracted from a completeness proof [3, 7]. The algorithm is generated automatically and correct by construction.

In this talk we investigate whether the concept of learning from unsuccessful attempts, known in the context of SAT solving as conflict driven clause learning (CDCL) [2], can be captured by program extraction as well. CDCL is implemented in a number of modern SAT solvers [11, 10, 1] significantly improving their performance. CDCL analyzes the structure of unsatisfiability proofs of sub-goals (known as conflicts). Such a proof of a conflict defines a directed graph, called conflict graph, whose nodes are labelled by literals. Each (horizontal) cut through the graph (assuming logical inference proceeds vertically) consists of literals whose conjunction implies the contradiction. Hence the disjunction of the negations of these literals is valid and can therefore be added as a learned clause to every other sub-goal making certain splittings of the search space unnecessary (this is sometimes called 'non-chronological backtracking'). Heuristics, called learning schemes, are needed to decide which of these learned clauses should be added since learning too many clauses can be problematic as it comes at the expensive of memory; a good clause learning scheme must choose clauses that speed up the proof search and delete those that do not. This poses a fundamental problem as being too aggressive in the deletion and or removal of clauses will make the solver incomplete [1] or worse incorrect.

The question is: Can clause learning, which takes place at a concrete, procedural, level and goes far beyond the mere problem of deciding satisfiability,

be captured at the abstract proof-theoretic level by program extraction?

We present a formalization of CDCL that partially answers this question. Our formalization treats the underlying data structures (for example, sets and sets of sets of literals to model clauses, as well as standard functions on sets) as abstract data types whose implementation is left open as a parameter. Also, the heuristics (choice of splitting literals and learning scheme) are left as uninterpreted parameters. This provides a modular approach to the actual implementation of the extracted solver: The correctness of the core SAT algorithm is guaranteed by the completeness proof while the correctness and efficiency of the implementation of the abstract data types and heuristics involved are isolated as separate problems. Another advantage of our formalization is that conflict graphs and cuts do not need to be modelled explicitly since they are extracted as realizers of unit resolution proofs. A prototype Haskell-implementation of the expected extracted program shows good performance despite the use of a rather inefficient implementation of sets as ordered lists. The implementation of the formal completeness proof in the MINLOG system [4] and ensuing program extraction are ongoing work.

References

[1] G. Audemard and L. Simon. Glucose: a solver that predicts learnt clauses quality. SAT Competition, pages 7–8, 2009.

[2] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. Journal of Artificial Intelligence Research, 22:319–351, 2004.

[3] U. Berger, A. Lawrence, F. Nordvall-Forsberg, and M. Seisenberger. Extracting verified decision procedures: DPLL and resolution. Logical Methods in Computer Science, 11(1), 2015.

[4] U. Berger, K. Miyamoto, H. Schwichtenberg, and M. Seisenberger. Minlog - A Tool for Program Extraction Supporting Algebras and Coalgebras. In A. Corradini, B. Klin, and C. Cîrstea, editors, CALCO-Tools 2011, volume 6859 of Lecture Notes in Computer Science, pages 393–399. Springer, 2011.

[5] L. Cruz-Filipe, J. P. Marques-Silva, and P. Schneider-Kamp. Formally verifying the solution to the boolean pythagorean triples problem. J Autom Reasoning, 63:695–722, 2019.

[6] M. Heule, O. Kullmann, and V. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In International Conference on Theory and Applications of Satisfiability Testing, pages 228–245. Springer, 2016.

[7] A. Lawrence. Verification of Train Control Systems: Tools and Techniques. PhD thesis, Swansea University, 2015.

[8] S. Lescuyer and S. Conchon. A Reflexive Formalization of a SAT Solver in Coq. In Otmane Aït Mohamed, César Muñoz, and Sofiène Tahar, editors, Theorem Proving in Higher Order Logics (TPHOLs 2008), volume 5170 of Lecture Notes in Computer Science, 2008.

[9] F. Marić. Formal verification of a modern SAT solver by shallow embedding into Isabelle/HOL. Theoretical Computer Science, 411(50):4333–4356, 2010.

[10] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In Annual ACM IEEE Design Automation Conference, pages 530–535. ACM, 2001.

[11] K. Pipatsrisawat and A. Darwiche. On the power of clause-learning sat solvers with restarts. In I. P. Gent, editor, Principles and Practice of Constraint Programming, volume 5732 of Lecture Notes in Computer Science, pages 654–668. Springer Berlin / Heidelberg, 2009.