Compressing Propositional Refutations Using Subsumption

Hasan Amjad
*University of Cambridge Computer Laboratory
15 J J Thomson Avenue, Cambridge CB3 0FD, UK
Hasan.Amjad@cl.cam.ac.uk

Abstract

We describe ongoing work on the use of subsumption to remove redundant inferences from propositional resolution refutation proofs of the style generated by conflict driven clause learning SAT solvers. This is used for faster LCF-style proof replay in interactive theorem provers. There may also be an application in the extraction of small unsatisfiable cores.

1 Introduction

Recent years have seen a trend towards tool combination in the automated reasoning community, where specialised high-performance proof engines are integrated within more mature and expressive frameworks. The hope is to combine the expressiveness and maturity of the framework with the performance of the specialised tool.

One such framework is the "HOL" family of LCF style higher-order logic interactive theorem provers, e.g., Isabelle, HOL4, HOL Light etc(Gordon and Melham, 1993). Being LCF style, only a small core of code is trusted. This assurance comes at a price: all non-trusted code must ultimately delegate to the core.

A SAT solver is a specialised high-performance tool for deciding propositional satisfiability. Many important problems in verification can be solved by representing them as instances of the satisfiability problem. zChaff (Moskewicz et al., 2001) is, among others, a well known SAT solver that can produce a resolution refutation proof in case of unsatisfiability.

In earlier work we constructed an LCF style interface between HOL provers and proof producing SAT solvers (Weber and Amjad, 2007), dramatically improving the provers' propositional proof capability. In this paper we describe ongoing work on compressing SAT solver proofs using subsumption. The aim is to reduce proof checking time in the theorem prover by giving it a shorter proof.

We assume the reader is familiar with the terms resolution and subsumption as used in automated reasoning, and related terminology.

2 Our Contribution

We can prove propositional tautologies in interactive provers by asking a SAT solver to prove that the negation of the tautology is unsatisfiable. The SAT solver outputs a proof of unsatisfiability. The input term to the SAT solver must be in conjunctive normal form. The proof is then output as a ground resolution refutation from the initial clauses of the input term. This proof is then replayed in the LCF style prover.

Subsumption on propositional clauses is just subset inclusion, where clauses are considered as sets of literals. The core idea is that replacing clauses by smaller ones may allow us to skip inferences further down the proof, and also give faster convergence to the empty clause.

The proof trace from the SAT solver is just a log of all conflict clause derivations. Each derivation is a linear *chain* of resolutions. Since the SAT solver uses backtracking, many chains are never used later in the proof. As a preprocessing step, we discard such chains via a DFS of the resolution graph starting with the empty clause.

A chain can be represented by the notation $C_0(p_1)C_1(p_2)C_2 \ldots (p_n)C_n$, i.e., resolve clauses C_0 and C_1 with pivot variable p_1, then resolve the resolvent with C_2, with pivot p_2, and so on. Let $R_1, R_2, \ldots$ be the intermediate resolvents and R_n be the final resolvent. The C_i are either initial clauses or the final resolvents of earlier chains.

We scan resolvents in order of derivation, looking for subsumption by initial clauses or earlier final resolvents. During the scan of a chain, a subsumption may replace, say, R_1 by D, such that $p_2 \notin D$. We can then reduce the original chain to $C_0(p_1)C_1(p_3)C_3 \ldots (p_n)C_n$ which derives a possibly smaller final resolvent. Suppose C_1 was an earlier such resolvent, so perhaps now $p_3 \notin C_1$. If $p_3 \notin C_0$ also, we can further reduce the chain to $C_0(p_1)C_1 \ldots (p_n)C_n$.

The scan for subsuming clauses must be done on the fly, since proofs are often too big to read into memory. We use forward subsumption since backward subsumption, while faster, suffers from serious problems, which we shall touch upon later. We considered ideas for fast forward subsumption used in first-order provers but were unable to usefully adapt them to our setting. Instead, we employ the following ideas to speed up forward subsumption:

1. Given how conflict clauses are derived, we may assume $\forall i.p_i \notin R_n$. Then either $\exists D.D \subseteq R_n$, or R_n is not subsumed. If it is, and if also $\exists E, i < n.E \subseteq R_i \wedge E \nsubseteq R_n$, then $\exists j > i.p_j \in E$. On the other hand, if R_n is not subsumed, then if $\exists E, i < n.E \subseteq R_i$, the same restriction on E applies. Thus, when checking subsumption for any R_i, it suffices to look at clauses containing p_j for $j > i$. In fact, it suffices to check only those containing p_{i+1}.

2. The above optimisation does not help when checking subsumption of R_n, where clauses containing any of the literals of R_n must be examined. However, by ordering literals, it suffices to check only those clauses whose least literal is in R_n, by checking clauses containing literals of R_n in ascending order of literal.

3. Once we have rejected obvious non-candidates by considering their literals, we can further narrow down the search by considering clause characteristics. Consider $C = \{p_1, \ldots, p_n\}$. For $\{q_1, \ldots, q_m\} \subseteq C$, it must be that $m \leq n$, $q_1 \geq p_1$ and $q_m \leq p_n$, if literals are ordered. Then we can partition clauses by size and for each partition:

   (a) Map a clause $D = \{q_1, \ldots, q_m\}$ to the point $(q_m, q_1) \in \mathbb{R}^2$.

   (b) Record the projection D_π of (q_m, q_1) considered as a position vector, onto the unit position vector $(1/\sqrt{2}, 1/\sqrt{2})$.

   (c) Then for $D \subseteq \{p_1, \ldots, p_n\}$ to hold, we need $\sqrt{2}p_1 \leq D_\pi \leq \sqrt{2}p_n$.

   Thus we quickly narrow the search to promising clauses. The worst-case complexity is as bad as a brute force linear scan of all candidate clauses because the range is an over-approximation, but the method works well in practice.

4. Finally, we use a fast but incomplete test for subset inclusion (Eén and Biere, 2005), falling back to a more expensive check that is linear in the length of the smaller clause.

3 Concluding Remarks

We have so far achieved compression ratios of three to seven percent with our prototype implementation (ignoring the compression from the obvious preprocessing step), on large industrial problems taken from the SATLIB benchmarks collection. The small figures are unsurprising, as SAT solvers are already very accurate and efficient. The pay-off in terms of net time savings has so far been mixed, with savings between -20% to +4%, but we are confident the pay-offs will all become positive as our implementation improves.

Even though checking backward subsumption is faster than forward subsumption, for each successful check we must confirm that the subsumed clause was not used to derive the subsuming clause, to avoid a circular dependency in the proof. Doing this efficiently is a challenge, because the proofs can be very large, with millions of inferences. Preliminary experiments show compression doubling, but the implementation is too slow at the moment.

Another application of this work is the extraction of small unsatisfiable cores. We just count the initial clauses of the compressed proof. At the moment this count yields about 50% of the performance of the latest algorithm (Gershman et al., 2006). We are therefore hopeful that with the addition of backward subsumption and special heuristics for expressing initial clause preference during scans, we will have a competitive method.

There has been limited related work in this area. Subsumption has been used for minimising SAT problems prior to proof search (Eén and Biere, 2005) but has been mostly ruled out as too costly to be of use during proof search. Though there is much literature on proof compression, particularly in the proof-carrying code community, we have only been able to find theoretical papers as far as shortening of propositional proofs is concerned.

References

Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *SAT*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005.

Roman Gershman, Maya Koifman, and Ofer Strichman. Deriving small unsatisfiable cores with dominators. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification*, volume 4144 of *LNCS*, pages 109–122. Springer, 2006.

M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL : A theorem-proving environment for higher order logic*. Cambridge University Press, 1993.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, pages 530–535. ACM Press, 2001.

Tjark Weber and Hasan Amjad. Efficiently checking propositional refutations in HOL theorem provers. *JAL (special issue on Empirically Successful Computerized Reasoning)*, 2007. In review. Conditionally accepted.