Using a SAT Solver as a Fast Decision Procedure for Propositional Logic in an LCF-style Theorem Prover

Tjark Weber `webertj@in.tum.de`

Institut für Informatik, Technische Universität München, Germany

Abstract

We describe the integration of a leading SAT solver with Isabelle/HOL, a popular interactive theorem prover. The SAT solver generates resolution-style proofs for (instances of) propositional tautologies. These proofs are verified by the theorem prover. The presented approach significantly improves Isabelle's performance on propositional problems, and furthermore exhibits counterexamples for unprovable conjectures.

Introduction

Interactive theorem provers like PVS, HOL or Isabelle traditionally support rich specification logics. Proof search and automation for these logics however is difficult, and proving a non-trivial theorem usually requires manual guidance by an expert user. **Automated theorem provers** on the other hand, while often designed for simpler logics, have become increasingly powerful over the past few years. New algorithms, improved heuristics and faster hardware allow interesting theorems to be proved with little or no human interaction, sometimes within seconds.

Formal verification is an important application area of interactive theorem proving. Problems in verification can often be reduced to **Boolean satisfiability (SAT)**, and recent SAT solver advances have made this approach feasible in practice. Hence the performance of an interactive prover on propositional problems may be of significant practical importance.

Here we describe the integration of zChaff [MMZ+01], a leading SAT solver, with the Isabelle/HOL [NPW02] prover. We show that using zChaff to prove theorems of propositional logic dramatically improves Isabelle's performance on this class of formulas. Furthermore, while Isabelle's previous decision procedures simply fail on unprovable conjectures, zChaff is able to produce concrete counterexamples.

System Description

To prove a propositional tautology ϕ in the Isabelle/HOL system with the help of zChaff, we proceed in several steps. First ϕ is negated, and the negation is converted into an equivalent formula ϕ^* in conjunctive normal form. ϕ^* is then written to a file in **DIMACS CNF** format, the input format supported by zChaff (and many other SAT solvers). zChaff, when run on this file, returns either "unsatisfiable", or a satisfying assignment for ϕ^*.

In the latter case, the satisfying assignment is displayed to the user. The assignment constitutes a counterexample to the original conjecture. When zChaff returns "unsatisfiable" however, things are more complicated. The LCF-approach [Gor00] demands that we **verify zChaff's claim of unsatisfiability within Isabelle/HOL**. While this is not as simple as the validation of a satisfying assignment, the increasing complexity of SAT solvers has before raised the question of support for independent verification of their results, and in 2003 zChaff has been extended by L. Zhang and S. Malik [ZM03] to generate resolution-style proofs that can be verified by an independent checker. Hence our main task boils down to using Isabelle/HOL as an independent checker for the **resolution proof** found by zChaff.

Preprocessing

Isabelle/HOL offers higher-order logic, whereas zChaff only supports formulas of propositional logic in conjunctive normal form. Therefore the (negated) input formula ϕ must be preprocessed before it can be passed to zChaff. Note that it is not sufficient to convert ϕ into an equivalent formula ϕ' in CNF. Rather, we have to *prove* this equivalence inside Isabelle/HOL. The result is not a single formula, but a **theorem** of the form $\phi = \phi'$. Our main workhorse for the construction of this theorem is a generic function `thm_of`:

```
thm_of decomp t = let (ts, recomb) = decomp t in recomb (map (thm_of decomp) ts)
```

All necessary preprocessing steps can then be handled with proper instantiations for `decomp`. zChaff treats clauses as sets of literals, making implicit use of associativity, commutativity and idempotence of disjunction. Therefore some **further preprocessing** is necessary, aside from conversion to CNF: removal of parentheses, of duplicate literals, and of tautological clauses. Each preprocessing step yields an equivalence theorem that was proved in Isabelle/HOL, and transitivity of $=$ allows us to combine these theorems into a single theorem $\phi = \phi^*$, where ϕ^* is the final result of our conversion.

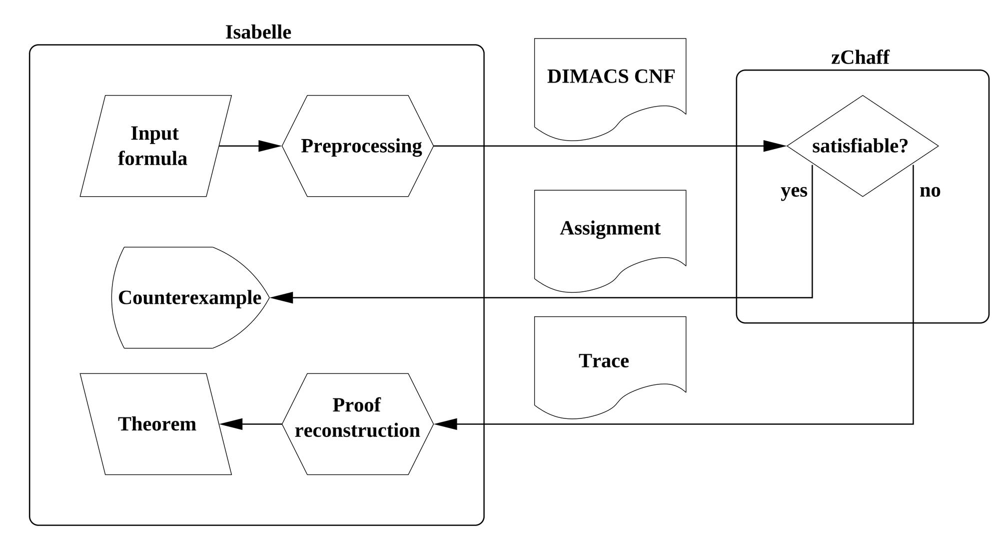

Figure 1: System Architecture

Proof Reconstruction

Proof reconstruction in Isabelle is based on **two simple functions**: one that uses resolution to derive new theorems of the form $\phi^* \longrightarrow c$ from existing theorems $\phi^* \longrightarrow c_1, \ldots, \phi^* \longrightarrow c_n$ (where c and $c_1, \ldots, c_n$ are single clauses), and another function that proves $\phi^* \longrightarrow l$ (where l is a single literal) from l's antecedent $\phi^* \longrightarrow c$. Here c must be a clause that contains l, and for all other literals l' in c a theorem of the form $\phi^* \longrightarrow \neg l'$ must be provable. These functions correspond to the first and second section, respectively, of the text file generated by zChaff.

```
prove_clause clause_id = resolution (map prove_clause (resolvents_of
clause_id))

prove_literal var_id = let th_ante = prove_clause (antecedent_of var_id) in
     let var_ids = filter (fn i => i <> var_id) (var_ids_in_clause th_ante) in
          resolution (th_ante :: map prove_literal var_ids)
```

Proof reconstruction then proceeds in three steps. First the **conflict clause** is proved by a call to `prove_clause`. Then `prove_literal` is called for every literal in the conflict clause, to show that the literal must be false. Finally resolving the conflict clause with these negated literals yields the theorem $\phi^* \longrightarrow \text{False}$.

For efficiency reasons, the actual implementation is slightly different from what is shown above. Theorems that were proved once are stored in two arrays (one for clauses, one for literals), and simply **looked up** – rather than reproved – should they be needed again. Hence our implementation is not purely functional.

Evaluation

Isabelle/HOL offers three major automatic proof procedures: *auto*, *blast*, and *fast*. Details can be found in [NPW02]. We compared the performance of our approach to that of Isabelle's existing proof procedures on all 42 problems contained in version 2.6.0 of the **TPTP library** that have a representation in propositional logic. The problems were negated, so that unsatisfiable problems became provable. All benchmarks were run on a machine with a 3 GHz Intel Xeon CPU and 1 GB of main memory.

19 of these 42 problems are rather easy, and were solved in less than a second each by both the existing procedures and the SAT solver approach. Figure 2 shows the times in seconds required to solve the remaining 23 problems. An **x** indicates that the procedure ran out of memory or failed to terminate within an hour.

Problem	Status	auto	blast	fast	zChaff
MSC007-1.008	unsat.	**x**	**x**	**x**	726.5
NUM285-1	sat.	**x**	**x**	**x**	0.2
PUZ013-1	unsat.	0.5	**x**	5.0	0.1
PUZ014-1	unsat.	1.4	**x**	6.1	0.1
PUZ015-2.006	unsat.	**x**	**x**	**x**	10.5
PUZ016-2.004	sat.	**x**	**x**	**x**	0.3
PUZ016-2.005	unsat.	**x**	**x**	**x**	1.6
PUZ030-2	unsat.	**x**	**x**	**x**	0.7
PUZ033-1	unsat.	0.2	6.4	0.1	0.1
SYN001-1.005	unsat.	**x**	**x**	**x**	0.4
SYN003-1.006	unsat.	0.9	**x**	1.6	0.1
SYN004-1.007	unsat.	0.3	822.2	2.8	0.1
SYN010-1.005.005	unsat.	**x**	**x**	**x**	0.4
SYN086-1.003	sat.	**x**	**x**	**x**	0.1
SYN087-1.003	sat.	**x**	**x**	**x**	0.1
SYN090-1.008	unsat.	13.8	**x**	**x**	0.5
SYN091-1.003	sat.	**x**	**x**	**x**	0.1
SYN092-1.003	sat.	**x**	**x**	**x**	0.1
SYN093-1.002	unsat.	1290.8	16.2	1126.6	0.1
SYN094-1.005	unsat.	**x**	**x**	**x**	0.8
SYN097-1.002	unsat.	**x**	19.2	**x**	0.2
SYN098-1.002	unsat.	**x**	**x**	**x**	0.4
SYN302-1.003	sat.	**x**	**x**	**x**	0.4

Figure 2: Running times (in seconds) for TPTP problems

Conclusions and Future Work

Our results show that the zChaff-based tactic is **clearly superior** to Isabelle's built-in tactics for propositional formulas. With the help of zChaff, many formulas that were previously out of the scope of Isabelle's built-in tactics can now be proved – or refuted – automatically, often within seconds. Isabelle's applicability as a tool for formal verification, where large propositional problems occur in practice, has thereby improved considerably, even though its performance is not yet sufficient to treat huge SAT problems with thousands of clauses.

The approach presented in this paper has **applications beyond propositional reasoning**. The decision problem for (fragments of) richer logics can be reduced to SAT. Consequently, proof reconstruction for propositional logic can serve as a foundation for proof reconstruction for other logics. Based on our work, one only needs a proof-generating implementation of the reduction to integrate the whole SAT-based decision procedure with an LCF-style theorem prover.

Acknowledgments The author would like to thank Sharad Malik and Zhaohui Fu for their help with zChaff, and Tobias Nipkow for his valuable suggestions.

References

[Gor00] M. J. C. Gordon. From LCF to HOL: a short history. In G. Plotkin, Colin P. Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction*. MIT Press, 2000.

[MMZ+01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, Las Vegas, June 2001.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[ZM03] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Design, Automation and Test in Europe (DATE 2003)*, pages 10880–10885. IEEE Computer Society, 2003.