System Description: GrAnDe 1.0

Stephan Schulz[1] and Geoff Sutcliffe[2]

[1] Institut für Informatik, Technische Universität München
schulz@informatik.tu-muenchen.de
[2] Department of Computer Science, University of Miami
geoff@cs.miami.edu

1 Introduction

The validity problem for full first-order logic is only semi-decidable. However, there are many interesting problems that, when expressed in clause normal form, have a finite Herbrand universe. They fall into a decidable subclass of first-order logic. Traditionally, such problems have been tackled using conventional first-order techniques. Some implementations, e.g. DCTP [SL01], are decision procedures for this class of problems. An alternative approach, justified by Herbrand's theorem, is to generate the ground instances of such a problem and use a propositional decision system to determine the satisfiability of the resulting propositional problem. The applicability of the grounding approach has led to these problems being called "effectively propositional" (EPR) problems. The TPTP problem library [SS98] v2.4.1 contains 574 EPR problems. Many of these are group theory problems (101 problems) and CNF translations of formulae in propositional multi-modal logic (206 problems).

There have been claims that first-order techniques are at least as good as the grounding approach on EPR problems. In order to test these claims we implemented the grounding approach in the system PizEAndSATO, and entered it (as a demonstration system) into the EPR division of CASC-JC [SSP02,Sut01]. Despite its prototypical nature, PizEAndSATO 0.2 achieved second place. The winner, E-SETHEO csp01, is a compositional system that used both a grounding approach and several different first-order procedures sequentially.

This paper describes the latest incarnation of our system, GrAnDe 1.0 (short for *Gr*ound *An*d *De*cide). GrAnDe has two principal components: the grounding procedure `eground` [Sch02] and the propositional prover ZChaff [MMZ+01]. ZChaff was chosen from a field of powerful and mature propositional systems [HS01].[1] The absence of an adequately powerful and accessible grounding tool, on the other hand, necessitated the development of `eground` from scratch.

This paper provides details of our contributions: The construction of `eground`, the combination of the systems, and an evaluation. For details about ZChaff, see [MMZ+01]. GrAnDe is available on the web at

`http://www.cs.miami.edu/~tptp/ATPSystems/GrAnDe/`

[1] ZChaff replaced SATO [ZS00] due to ZChaff's ability to deal with larger proposition numbers.

2 The Grounding Procedure `eground`

An EPR problem is certainly (un)satisfiable if the set of *all* ground instances of its clauses is (un)satisfiable. However, with n constant symbols, a single clause with m variables has n^m ground instances. Thus, the set of all ground instances is often too large to compute with reasonable (or even reasonably unreasonable) resources. An aim in developing `eground` was to find smaller sets of ground instances that are still equiconsistent with the original set of clauses. In order to achieve this, three techniques were combined: clause splitting, structural constraints on variable instantiation, and propositional simplification.

Basic *clause splitting* takes a clause $C \vee D$, in which C and D do not share any variables, and replaces it by two clauses $C \vee p$ and $D \vee \neg p$, where p is a new propositional symbol. This does not change the satisfiability of the clause set – any model of the original clause set can be extended to a model of the modified clause set, and any model of the modified clause set satisfies the original one [Sch02]. The latest version of `eground` also splits a clause if C and D have common variables, but each part has some variables not occurring in the other. In this case, the variables occurring in both C and D are the arguments of p. If a clause has more than two parts that can be separated then *hyper-splitting* [RV01] is used, i.e., a clause with k parts is replaced in one step by k split clauses and a single link clause.

Clause splitting is performed in `eground` before starting to instantiate the clauses. Although splitting increases the number of clauses, it typically significantly reduces the number of ground instances. If C contains m_C variables, and D contains m_D variables, $C \vee D$ has $n^{m_C} n^{m_D}$ ground instances, where n is the number of constants. The two split clauses have only $n^{m_C} + n^{m_D}$ ground instances.

Structural constraints are used to prohibit variable instantiations that would lead to the generation of pure literals in the resulting ground clauses, and thus approximate the hyper-linking condition [LP92]. An overview of structural constraints is given here; for a full formal definition see [Sch02]. A *local structural constraint* is induced for each triple polarity/predicate symbol/position. Consider a symbol P with arity k. The local structural constraint with positive polarity at position $p \in \{1 \dots k\}$ is induced by all negative literals with symbol P. If there exists such a negative literal L such that $L|_p$ is a variable, the position is unconstrained. Otherwise, the constraint has the form $P|_p = c_1 \vee \dots \vee P|_p = c_i$, where the c_i are exactly the constants that appear at position p in the negative literals with predicate symbol P. Any instance of a positive literal with symbol P that does not fulfill the structural constraint will be pure, and hence ground clauses containing such instances are not needed. Constraints with negative polarity are induced by all positive literals in the same way. The local structural constraints for each variable in a clause are conjoined and simplified into a minimal conjunctive normal form. The acceptable instantiations for each variable can then be read directly off the constraints, so that the generation of the ground instances is done very efficiently. If there are no possible instantiations for some variable in a clause, then no instances of the clause are generated.

Splitting and structural constraints are somewhat complementary. Clause splitting works one clause at a time, while structural constraints are induced by the whole clause set. If variables occur in only a small number of literals, the chance that the clause can be split increases. If a variable occurs in many positions, the likelihood that it is constrained increases.

The third technique used to reduce the size of the ground clause set is *propositional simplification*. Tautology deletion and forward unit subsumption are used to delete redundant ground clauses, and forward unit resolution is used to reduce the size of the generated ground clauses. The forward unit simplifications use existing propositional units to subsume and simplify newly generated ground clauses. Due to the internal data structures used, these simplifications are done at negligible cost.

Despite the optimizations, there are still problems where `eground` runs out of time or, more often, out of memory.[2] However, this does not necessarily mean that the effort has been in vain. While the satisfiability of the original clause set can be shown only by satisfying *all* non-redundant ground instances, unsatisfiability may be shown using only a subset of the instances. In order to allow GrAnDe to take advantage of this possibility, `eground` can be configured to output a set of ground clauses when it runs out of time or memory. In this situation the normal generation procedure is stopped, and a single ground instance of each input clause is added to the already generated clause set. The resultant clause set is *incomplete* (with respect to the test for unsatisfiability).

The ground clauses generated are converted to propositional clauses and output in DIMACS format [DIM], followed by a message indicating whether the clause set is complete or incomplete.

3 Combining the systems

A `Perl` script called `And` is used to combine `eground` and ZChaff. The script invokes `eground` on the EPR problem, allowing it maximally 66% of the CPU time limit. The propositional clauses written by `eground` are captured into a temporary file, which is used as the input for ZChaff. ZChaff is allowed whatever CPU time has not been used by `eground` (more than 34% of the CPU time limit in the cases where `eground` terminates early, either because it runs out of memory or because it has generated a complete ground clause set). The completeness marker line written by `eground` is used by `And` for interpreting the output of ZChaff. If `eground` has generated an incomplete clause set and ZChaff reports that it is satisfiable, then no result is reported by GrAnDe. If `eground` has generated an incomplete clause set and ZChaff reports that it is unsatisfiable, or if `eground` has generated a complete clause set, then ZChaff's result is reported by GrAnDe as the overall result.

[2] Because `eground` generates output in DIMACS format, it can only print the result once the number of clauses and the largest proposition number are known.

4 Experimental Results

GrAnDe 1.0 has been tested on the 574 CNF EPR problems in TPTP v2.4.1. Of these, 413 are unsatisfiable and 161 are satisfiable. Three systems that use first-order techniques were also tested: DCTP 1.0 [SL01], SPASS 1.03 [Wei99], and Vampire 2.0 [RV99]. DCTP is a decision procedure for EPR problems; SPASS is complete for unsatisfiable problems and known to be very strong on satisfiable problems in general; Vampire 2.0 was the winner of the MIX division of CASC-JC. Unfortunately, we were unable to get a working hyper-linking prover, as e.g., OSHL [PZ00], for comparison. PizEAndSATO 0.2 was also tested, to gauge progress in our development. The testing was done on a SUN Ultra-80, with a 450MHz CPU and 400 MB of RAM. A 300 second overall CPU time limit was imposed for each problem. The results are summarized in Table 1.

Table 1. Results

System	Total	Unsatisfiable	Satisfiable
	574	413	161
DCTP 1.0	489	352	137
SPASS 1.03	498	397	101
Vampire 2.0	498	403	95
PizEAndSATO 0.2	543	407	136
GrAnDe 1.0	546	408	138

Of the 574 problems, `eground` was able to create a complete ground clause set for 546 problems. Of these, ZChaff reported 138 as satisfiable, 407 as unsatisfiable, and timed out on one problem. Of the 28 problems for which `eground` was unable to create a complete clause set, ZChaff reported 8 as satisfiable (i.e. no result for GrAnDe), one as unsatisfiable (i.e. the problem was solved despite eground being unable to generate a complete clause set), and timed out on 19 problems.

509 of the problems solved by GrAnDe were solved in less than one second of CPU time, and only 16 problems required more than 10 seconds. Of these 16 "hard" problems, only one required more than 60 seconds. This is the problem for which `eground` generated an incomplete clause set that ZChaff still found to be unsatisfiable. For the other 15 "hard" problems, `eground` always took most of the CPU time, but never more than 40 seconds. There thus seem to be three major scenarios (with the 300 second CPU time limit): either i) the problem is easy for GrAnDe, or ii) `eground` completes reasonably quickly (within 40 seconds) and ZChaff can decide the clause set, or iii) `eground` hits a resource limit and ZChaff is unable to decide the resulting incomplete, typically very large, clause set in the remaining time.

5 Possible Improvements

Despite the fact that `eground` is the most powerful grounding procedure we are aware of, at the moment it still is the weakest link in GrAnDe. There are a number of possible improvements to the program. Two important ideas are the use of more simplification in `eground`, e.g., the use of simple indexing for non-unit subsumption (and possibly subsumption resolution), and the use of more restrictive local unification constraints instead of the simple structural constraints. Additionally, it might be possible to generate better (smaller and more diverse) incomplete clause sets if complete grounding is not possible.

References

[DIM] DIMACS. Satisfiability Suggested Format. ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc/satformat.tex.

[HS01] H. Hoos and T. Stützle. SATLIB: An Online Resource for Research on SAT. In I. Gent, H. van Maaren, and T. Walsh, editors, *Proc. of the 3rd Workshop on the Satisfiability Problem*, 2001. http://www.satlib.org/.

[LP92] S-J. Lee and D.A. Plaisted. Eliminating Duplication with the Hyper-Linking Strategy. *Journal of Automated Reasoning*, 9(1):25–42, 1992.

[MMZ+01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In D. Blaauw and L. Lavagno, editors, *Proc. of the 39th Design Automation Conference*, pages 530–535, 2001.

[PZ00] D.A. Plaisted and Y. Zhu. Ordered Semantic Hyper-linking. *Journal of Automated Reasoning*, 25(3):167–217, 2000.

[RV99] A. Riazanov and A. Voronkov. Vampire. In H. Ganzinger, editor, *Proc. of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 292–296. Springer, 1999.

[RV01] A. Riazanov and A. Voronkov. Splitting without Backtracking. In B. Nebel, editor, *Proc. of the 17th International Joint Conference on Artificial Intelligence* , pages 611–617. Morgan Kaufmann, 2001.

[Sch02] S. Schulz. A Comparison of Different Techniques for Grounding Near-Propositional CNF Formulae. In S. Haller and G. Simmons, editors, *Proc. of the 15th Florida Artificial Intelligence Research Symposium*. AAAI Press, 2002. To appear.

[SL01] G. Stenz and R. Letz. DCTP - A Disconnection Calculus Theorem Prover. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proc. of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 381–385. Springer, 2001.

[SS98] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[SSP02] G. Sutcliffe, C. Suttner, and J. Pelletier. The IJCAR ATP System Competition. *Journal of Automated Reasoning*, To appear, 2002.

[Sut01] G. Sutcliffe. CASC-JC. http://www.cs.miami.edu/ tptp/CASC/JC/, 2001.

[Wei99] C. Weidenbach, et al. SPASS Version 1.0.0. In H. Ganzinger, editor, *Proc. of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 378–382. Springer, 1999.

[ZS00] H. Zhang and M. Stickel. Implementing the Davis-Putnam Method. *Journal of Automated Reasoning*, 24(1/2):277–296, 2000.