iProver-Eq: An Instantiation-Based Theorem Prover with Equality

Konstantin Korovin*
*University of Manchester
korovin@cs.man.ac.uk

Christoph Sticksel†
†University of Manchester
csticksel@cs.man.ac.uk

1 Introduction

Instantiation-based methods are a class of deduction calculi for first-order clausal logic. The common idea is to instantiate clauses and to employ efficient propositional or more general ground reasoning methods in order to prove unsatisfiability or to find a model. Among other important properties, Instantiation-based methods naturally decide the first-order logic fragment of effectively propositional logic (EPR) which has interesting applications (see, e.g., Baumgartner (2007) for an overview).

iProver-Eq is an implementation of an instantiation-based calculus Inst-Gen-Eq which is complete for first-order logic with equality and decides the EPR fragment modulo equality. The system is an extension of the successful iProver system and preserves the characteristic feature of combining first-order reasoning with efficient ground satisfiability checking where the latter is delegated in a modular way to any state-of-the-art SMT solver, i.e. a solver for ground satisfiability modulo theories.

In the following we outline the iProver-Eq system and present its calculus for equational reasoning to generate instances.

2 System Overview

The basic idea of the Inst-Gen method, introduced in Ganzinger and Korovin (2003), is as follows. The set of first-order clauses is abstracted to a set of ground clauses by mapping all variables to the same ground term. An SMT solver is harnessed to check if the ground abstraction of the clauses is unsatisfiable, in which case the set of first-order clauses is also unsatisfiable. Otherwise, there is a ground model for the abstraction that is used to guide an instantiation process. The model is represented as a set of abstracted literals and an attempt is made to extend it to a model of the first-order clauses by reasoning on the first-order literals corresponding to the abstracted literals in the model. When this fails, new (not necessarily ground) instances of clauses are generated in a way that forces the ground solver to refine the model in the next iteration.

The saturation process in iProver-Eq is outlined in Figure 1. Two major components there are unit superposition for equational reasoning on literals and an SMT solver for ground reasoning, which are both non-trivial processes. Unit superposition will be described in the next section. The ground solver is regarded as a black box.

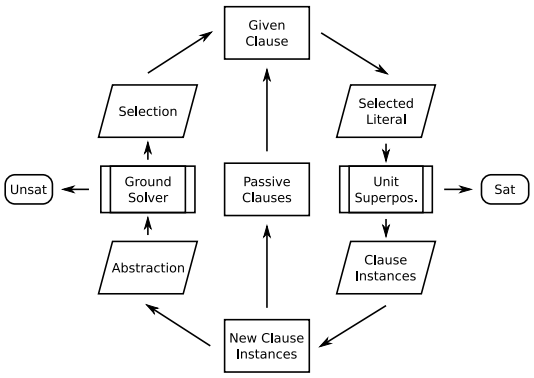

Figure 1: Saturation process in the iProver-Eq system

The saturation process is based on a given clause algorithm, which partitions the set of clauses into two disjoint sets, namely the Inst-Active and the Inst-Passive clauses. Initially, there are no Inst-Active clauses, all input clauses are considered to be new instances, their ground abstractions are passed to the ground solver which is then invoked to either return a model of the abstraction or to conclude its unsatisfiability. The new clauses are moved to the Inst-Passive set from where in each step of the process a clause, called the given clause, is chosen and put into the Inst-Active set. Using the current model of the ground abstraction, one of the literals in the given clause is selected and passed to the unit superposition calculus. If a subset of the selected literals is found to be inconsistent by the unit superposition calculus, then corresponding instances of clauses are added to the set of new clauses. The process continues by adding the abstractions of the new clauses to the SMT solver, running the solver on the extended set of ground clauses and moving the new clauses to the Inst-Passive set. The process maintains the invariant that the ground abstractions of the selected literals in the set of Inst-Active clauses are consistent and have been passed to the unit superposition component. iProver-Eq terminates with a result of unsatisfiable if the ground solver reports an unsatisfiable abstraction. If the Inst-Passive clause set is empty and the selected literals are consistent as stated by the unit superposition component, iProver-Eq terminates with the result satisfiable.

3 The Unit Superposition Calculus

If the set of selected (not necessarily ground) literals is consistent, a model for the set of Inst-Active clauses ex-

Selection

$$\frac{}{\{C\} \mid D\colon L}$$

where $C \in S$ and $\mathrm{sel}(C) = L$ and D is a constraint on clause C

Superposition

$$\frac{\ell_1 \mid D_1\colon l \simeq r \qquad \ell_2 \mid D_2\colon L[l']}{(\ell_1 \cup \ell_2)\theta \mid (D_1 \wedge D_2)\theta\colon L[r]\theta}\ (\theta)$$

where $L[l'] = u[l'] \simeq v$ or $L[l'] = u[l'] \not\simeq v$ and (i) $\theta = \mathrm{mgu}(l, l')$, (ii) l' is not a variable and (iii) for some grounding substitution μ (iiia) $l\theta\mu \succ r\theta\mu$, (iiib) $u[l']\theta\mu \succ v\theta\mu$ and (iiic) μ satisfies the constraint $(D_1 \wedge D_2)\theta$

Equality Resolution

$$\frac{\ell \mid D\colon l \not\simeq r}{\ell\colon \square}\ (\theta)$$

where $\theta = \mathrm{mgu}(l, r)$ and satisfies the constraint D

Figure 2: Labelled Unit Superposition

ists and it has thus been proved satisfiable. Otherwise, there is an inconsistent subset of the selected literals. The clauses that these literals are selected in are instantiated such that the inconsistency is witnessed by the ground abstraction. For non-equational literals it suffices to search for unifiable complementary literal pairs. However, in the presence of equations, we apply the unit superposition calculus in order to find inconsistent literals and to obtain clause instances.

For simplicity, we only consider pure equational logic where all atoms are equations, different clauses are assumed to be variable-disjoint. The inference rules of the unit superposition calculus are shown in Figure 2 and are similar to the standard superposition calculus, see, e.g., Nieuwenhuis and Rubio (1999).

Each literal in the calculus has a label consisting of a set of clauses and a constraint that is used for redundancy elimination which we will not describe here. When the selected literal of the given clause is received, we label it with the given clause and a constraint. The conclusion of an inference is labelled with the union of the labels of its premises where the unifier θ has been applied to each clause. The constraint is the conjunction of the constraints of its premises where again the unifier θ has been applied.

Ganzinger and Korovin (2004) originally defined the unit paramodulation calculus and a way to extract instantiating substitutions from proofs. Our addition of labels to literals replaces their extraction of substitutions. Instead of having to trace a proof tree to the literals at its leaves in order to obtain substitutions to be applied to clauses, our labels directly display the clause instances while still allowing for constraint notions to eliminate redundancy.

The conditions on applicability of an inference are independent of labels which are thus merely an annotation to facilitate the generation of clause instances from sets of inconsistent literals. Therefore, the same literal with different labels has the same conclusions. Further, it is well known from paramodulation approaches, that all variants of a literal allow the same inferences with conclusions that are variants of each other. Obviously, one wants to combine all labels of all variants of a literal in order to avoid duplicating the search for possible inferences for each literal. Although it is possible to merge all labels into one set of clauses, the challenge is to compactly represent combined labels in a way that preserves the constraint notions.

4 Conclusion

In general, not all optimisations from standard paramodulation-based calculi can be lifted to the unit superposition calculus. We have to take care not to omit clause instances that are required by the ground solver to witness inconsistency of its model. Nevertheless, our labelled calculus enables simplification by demodulation and there is a powerful semantic notion of redundancy which can be used to justify further techniques.

iProver-Eq makes use of state-of-the-art implementation techniques like term indexing for unification. It is reasonably efficient on the TPTP benchmark although it is in an early stage.

Acknowledgements

Our work is based on the iProver system, as in described in Korovin (2008) and Korovin and Sticksel (2010). We thank Renate Schmidt for helpful discussions.

References

Peter Baumgartner. Logical Engineering with Instance-Based Methods. In *CADE-21*, volume 4603 of *LNAI*, pages 404–409. Springer, 2007.

Harald Ganzinger and Konstantin Korovin. New Directions in Instantiation-Based Theorem Proving. In *LICS 2003*, pages 55–64. IEEE, 2003.

Harald Ganzinger and Konstantin Korovin. Integrating Equational Reasoning into Instantiation-Based Theorem Proving. In *CSL 2004*, volume 3210 of *LNCS*, pages 71–84. Springer, 2004.

Konstantin Korovin. iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In *IJCAR 2008*, volume 5195 of *LNCS*, pages 292–298. Springer, 2008.

Konstantin Korovin and Christoph Sticksel. iProver-eq – An Instantiation-based Theorem Prover with Equality. submitted to IJCAR 2010, 2010.

Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, 1999.