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

Konstantin Korovin and Christoph Sticksel

School of Computer Science
The University of Manchester
korovin@cs.man.ac.uk, sticksel@cs.man.ac.uk

Abstract. iProver-Eq is an implementation of an instantiation-based calculus Inst-Gen-Eq which is complete for first-order logic with equality. iProver-Eq extends the iProver system with superposition-based equational reasoning and maintains the distinctive features of the Inst-Gen method. In particular, first-order reasoning is combined with efficient ground satisfiability checking where the latter is delegated in a modular way to any state-of-the-art SMT solver. The first-order reasoning employs a saturation algorithm making use of redundancy elimination in form of blocking and simplification inferences. We describe the equational reasoning as it is implemented in iProver-Eq, the main challenges and techniques that are essential for efficiency.

1 Introduction

Instantiation-based methods (IMs) are a class of 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, IMs naturally decide the first-order logic fragment of effectively propositional logic (EPR) which has interesting applications as it has been shown recently (see, e.g., [2] for an overview). Let us remark that Inst-Gen-Eq decides the EPR fragment modulo equality.

The basic idea of the Inst-Gen method, introduced in [4], 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. If this ground abstraction is unsatisfiable, then 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 ground satisfiability check and construction of a ground model is delegated to an industrial-strength satisfiability modulo theories (SMT) solver in the presence of equations or to a propositional (SAT) solver if no equational reasoning is required.

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 reasoner to refine the model in the next iteration. Inst-Gen is therefore composed of two parts: ground satisfiability solving on the abstraction of the set of clauses and first-order reasoning on literals corresponding to ground literals in the model of the abstraction.

A characteristic feature of the Inst-Gen method is the delegation of the ground reasoning to a black-boxed off-the-shelf solver. The iProver-Eq system currently uses MiniSat [3] as the SAT solver and either CVC3 [1] or Z3 [7] as the SMT solver.

The iProver system [6] has treated equations only axiomatically, the iProver-Eq system adds equational reasoning based on term rewriting. Following the approach from [5], it implements a superposition-style calculus that both finds sets of inconsistent equational literals and obtains instantiating substitutions from the proof of their inconsistency.

The implementation addresses the combination of three main components:

1. ground reasoning by an SMT solver
2. superposition-based equational reasoning with literals in a candidate model
3. instantiation by extracting substitutions from proofs generated in 2.

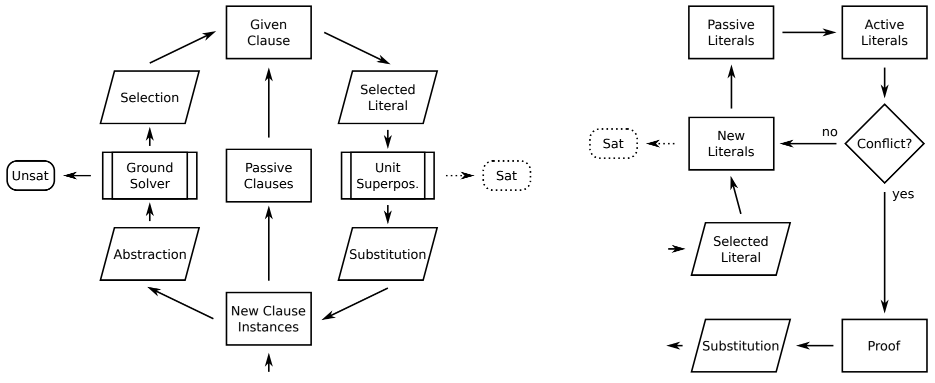
This system description first outlines the structure of the iProver-Eq system. We continue by defining the unit superposition calculus for equational reasoning and demonstrate it by way of an example. We discuss extraction of instantiating substitutions from proofs, giving an example for one of the non-trivial obstacles encountered which render the method incomplete if naively addressed. Finally, we highlight some of the main features of the implementation and conclude with an evaluation and directions for further research.

2 System Overview

Given a set of first-order clauses S we first form its ground abstraction $S\bot$ by mapping all variables to the same ground term, conventionally denoted $\bot$. If the ground abstraction $S\bot$ is unsatisfiable, the original set S is also unsatisfiable and the procedure can terminate. Otherwise, there is a model $I_\bot$ of the ground abstraction $S\bot$ and the first-order instantiation process is guided by means of a selection function based on $I_\bot$. The selection function assigns to each first-order clause C in S exactly one literal L from C such that $I_\bot \models L\bot$. At least one such literal always exists as the ground abstraction of the clause is true in the model $I_\bot$.

If the set of selected (not necessarily ground) literals, seen as unit clauses, is consistent in first-order logic modulo equality, a model for the clause set S exists and it has thus been proved satisfiable. Otherwise, there is a subset of the selected literals which is inconsistent and the clauses these literals are selected in are instantiated such that the inconsistency can already be witnessed in the ground abstraction and thus forces the ground solver to refine it. For non-equational literals it suffices to search for unifiable complementary literal pairs. In the presence of equations, we apply the unit superposition calculus in order to find inconsistent literals and to obtain instantiating substitutions.

iProver-Eq generates instances of clauses in a saturation process outlined in Figure 1(a). Two major components in this process are unit superposition (US) for equational reasoning on literals and an SMT solver for ground reasoning. Both are non-trivial processes and while the equational reasoning will be described in the next section, the ground solver is regarded as a black box. The saturation process is based on



(a) Inst-Gen-Eq-Loop (b) US-Loop

Fig. 1. Sketch of the iProver-Eq System

a given clause algorithm which partitions the set of clauses into two disjoint sets, in the following called the Inst-active and the Inst-passive clauses. The invariant is that the ground abstractions of the selected literals in the set of Inst-active clauses are consistent and have been passed to the US component. Initially, there are no Inst-active clauses, all clauses are considered to be new instances, their ground abstractions are input to the ground solver which is then invoked to return a model of the abstraction or to prove 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, see Figure 1(b). If a subset of the selected literals is found to be inconsistent by the unit superposition calculus, then substitutions are extracted from the proof of the inconsistency and corresponding instances 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. iProver-Eq terminates with a result of unsatisfiable if the ground solver reports an unsatisfiable abstraction. If the passive clause set is empty and the selected literals are consistent as indicated by the unit superposition component, iProver-Eq terminates with the result satisfiable.

3 The Unit Superposition Calculus

In this section we describe the inference rules of the unit superposition calculus for finding inconsistent equational literals and demonstrate it with an example (for a proof of completeness, see [5]). Subsequently we discuss the US-Loop saturation procedure.

For simplicity, we work with pure equational logic where all atoms are equations. The unit superposition calculus is similar to the standard superposition calculus, see, e.g., [8]. Different literals are assumed to be variable-disjoint and as we only work with literals, i.e. unit clauses, we can reduce the inference rules to the following ones.

Definition 1 (Unit Superposition).

$$\frac{l \simeq r \qquad s[l'] \simeq t}{(s[r] \simeq t)\sigma}\,(\sigma) \qquad \frac{l \simeq r \qquad s[l'] \not\simeq t}{(s[r] \not\simeq t)\sigma}\,(\sigma) \qquad \frac{l \not\simeq r}{\Box}\,(\sigma)$$

(i) $\sigma = \mathrm{mgu}(l, l')$, *(ii)* l' *is not a variable, (iii)* $l\sigma\theta \succ r\sigma\theta$ *and (iv)* $s[l']\sigma\theta \succ t\sigma\theta$ *for some grounding substitution* θ

$\sigma = \mathrm{mgu}(l, r)$

A *proof* of the contradiction, denoted $\Box$, is a tree where the leaves are literals selected in the Inst-active set of clauses, inner nodes are obtained by applying inference rules to the parent nodes and the root is the contradiction $\Box$. In order to extract instantiating substitutions we annotate each inference with the substitution σ applied. The composition of the substitutions along the path in the proof tree from a selected literal at a leaf to the contradiction yields a substitution which we call *relevant to the selected literal*. In the Inst-Gen-Eq-Loop we take all clauses whose selected literals are leaves in the proof and instantiate each clause with the substitution relevant to its selected literal.

Example 1. Consider the clauses (1)-(4) below and let the ground abstractions of their selected literals (the first literal in each clause) be as shown to their right.

(1)	$f(f(u)) \simeq f(u)$	$f(f(\bot)) \simeq f(\bot)$
(2)	$g(f(f(x)), f(y)) \simeq h(z) \ \vee\ g(f(x), y) \not\simeq h(c)$	$g(f(f(\bot)), f(\bot)) \simeq h(\bot)$
(3)	$g(f(a), f(b)) \not\simeq h(w)$	$g(f(a), f(b)) \not\simeq h(\bot)$
(4)	$g(f(a), b) \simeq h(c)$	$g(f(a), b) \simeq h(c)$

The clause set is unsatisfiable, but its ground abstraction is satisfiable with a model containing the literals shown. Accordingly, the literals can indeed be selected and we derive the contradiction using the US calculus.

$$\frac{\dfrac{\overset{(1)}{f(f(u)) \simeq f(u)} \qquad \overset{(2)}{g(f(f(x)), f(y)) \simeq h(z)}}{g(f(x), f(y)) \simeq h(z)}\,\{x/u\} \qquad \overset{(3)}{g(f(a), f(b)) \not\simeq h(w)}}{\dfrac{h(z) \not\simeq h(w)}{\Box}\,\{w/z\}}\,\{a/x, b/y\} \quad (\star)$$

By tracing the branches in the proof tree, we extract a relevant substitution for each of the three literals: $\{a/u\}$, $\{a/x, b/y, w/z\}$ and $\{\}$, respectively. We instantiate the clauses with the relevant substitutions of their selected literals. Clause (3) is instantiated to itself, the instances of the first two clauses are:

(5)	$f(f(a)) \simeq f(a)$
(6)	$g(f(f(a)), f(b)) \simeq h(w) \ \vee\ g(f(a), b) \not\simeq h(c)$

The ground abstraction is now unsatisfiable due to the following four clauses.

$(3_\bot)$	$g(f(a), f(b)) \not\simeq h(\bot)$	$(5_\bot)$	$f(f(a)) \simeq f(a)$
$(4_\bot)$	$g(f(a), b) \simeq h(c)$	$(6_\bot)$	$g(f(f(a)), f(b)) \simeq h(\bot) \ \vee\ g(f(a), b) \not\simeq h(c)$

Implementation of Unit Superposition. Figure 1(b) on page 3 is a sketch of the saturation procedure in the unit superposition component in the Inst-Gen-Eq-Loop. The saturation algorithm is a given literal algorithm, a variant of the given clause algorithm described above. Literals are either US-active or US-passive and the invariant is that all inferences between two US-active literals have been drawn. The US-passive set is populated with selected literals from the Inst-Gen-Eq-Loop in Figure 1(a). In every step a given literal is chosen from the US-passive set, put into the US-active set and all conclusions from inferences between the given literal and US-active literals are drawn. The given literal is considered to be US-active in order to enable inferences with itself. If a conclusion is contradictory, substitutions are extracted from its proof as shown above and passed to the Inst-Gen-Eq-Loop. Otherwise, the conclusion is added to the US-passive set and the US-Loop continues choosing another given literal.

The US-Loop is a sub-procedure inside the Inst-Gen-Eq-Loop and interleaved with it in a fair way such that neither process has to wait for termination of the other process. All clauses from inconsistent subsets of their selected literals have to be instantiated, therefore the US-Loop continues after having found a contradiction. If the US-passive set in the US-Loop is empty, the Inst-Gen-Eq-Loop continues and only if in both processes the sets of US-passive literals and Inst-passive clauses, respectively, are empty, iProver-Eq is in a saturated state and returns satisfiability.

4 Instances from Unit Superposition Proofs

In this section we discuss one of the problems related to the extraction of substitutions from proofs. We keep all clauses variable disjoint and thus the selected literals are also variable disjoint.

It is well-known from standard implementations of paramodulation that for every literal all its variants should be considered to be identical in order to avoid duplicating inferences. However, in the Inst-Gen framework, variants of a literal can have different proofs, in turn resulting in different substitutions which may all be required by the Inst-Gen-Eq-Loop. Moreover, variants of a literal can occur in the same proof, potentially leading to cycles as the following example shows.

Example 2. We modify the clause set from Example 1 by replacing clause (2) with

$$\text{(2')} \qquad g(f(x), f(y)) \simeq h(z) \ \lor \ g(x, y) \not\simeq h(c)$$

The clause set remains unsatisfiable with a satisfiable ground abstraction and again the first literals in each clause are selected. To prove unsatisfiability, instances have to be generated in a way which might seem not immediately obvious.

$$\dfrac{\dfrac{\dfrac{\overset{(1)}{f(f(u)) \simeq f(u)} \qquad \overset{(2')}{g(f(x), f(y)) \simeq h(z)}}{g(f(u), f(y)) \simeq h(z)}\{f(u)/x\} \qquad \overset{(3)}{g(f(a), f(b)) \not\simeq h(w)}}{\dfrac{h(z) \not\simeq h(w)}{\square}\{w/z\}}\{a/u, b/y\} \qquad (\diamond)$$

The literal $g(f(x), f(y)) \simeq h(z)$ is inferred to its variant which may seem redundant as the contradiction could already be derived from clauses (2') and (3). However, due to the substitution $\{f(u)/x\}$ in the inference, different substitutions are extracted from the proof. Indeed, only upon adding the instances of clauses (1) and (2') with the respective substitutions $\{a/u\}$ and $\{f(a)/x, b/y, w/z\}$ which are identical to clauses (5) and (6) from Example 1 the ground abstraction becomes unsatisfiable as shown there.

If we identify the literal variants of $g(f(x), f(y)) \simeq h(z)$ with each other, the proof tree in the example would collapse to a tree with the addition of a cycle on the literal. Bookkeeping information about cycles considerably complicates extraction of substitutions from proofs and approaches are needed that eliminate cycles.

Our main approach is to extract substitutions after each inference step and to use them to label the inferred literal. In order to combine all literal variants we introduce a new inference rule which merges different labels of the same literal. We also modify the inference rules from Definition 1 to accommodate labels which are merely annotations and do not influence the applicability of inferences. The conditions on the inference rules remain as in Definition 1.

Definition 2 (Labelled Unit Superposition).

$$\frac{\ell_1 \colon L \qquad \ell_2 \colon L}{\ell_1 \cup \ell_2 \colon L} \qquad \frac{\ell_1 \colon l \simeq r \qquad \ell_2 \colon L[l']}{\ell_1\sigma \cup \ell_2\sigma \colon L[r]\sigma}\,(\sigma) \qquad \frac{\ell \colon l \not\simeq r}{\ell\sigma \colon \Box}\,(\sigma)$$

In proof ($\diamond$) above, the inferred variant of the literal $g(f(x), f(y)) \simeq h(z)$ has a different label from the variant used as a premise. By first merging the labels of both variants and subsequently deriving the contradiction, we obtain the instances of (1), (2') and (3) from proof ($\diamond$) as well as the instances from (2') and (3) alone. We now do not need to trace a proof tree that potentially contains cycles, the necessary substitutions to generate instances can be read from the label of the contradiction.

For Example 2, we can choose a second approach orthogonal to labelling to tackle cycles by generating instances. We instantiate clause (2') with the substitution $\{f(u)/x\}$ from the cycle to clause (2) from Example 1 and can prove unsatisfiability as shown there. The cyclic proof ($\diamond$) becomes redundant and the relevant substitution for clause (2) leading to its instance (6) can instead be extracted from the proof ($\star$). Although this method of unfolding cycles by eagerly instantiating clauses seems simple and yet powerful, it becomes much more involved when the substitution in the cycle is not proper, i.e. no variable is instantiated to a term, e.g., $\{y/x, y/z\}$. For such non-proper cycles the labelled approach is advantageous and we are investigating the benefits of combining both approaches.

5 Features of the Implementation

iProver-Eq is implemented in the functional language OCaml and uses MiniSat, CVC3 and Z3 as ground (SAT/SMT) solvers via their C/C++ APIs. It processes input in TPTP format and uses the E prover[1] for clausification of non-CNF problems. We briefly mention the most significant features of the implementation, some of which have already been present in the iProver system and were adapted or extended.

Passive Clauses/Literals Both the Inst-passive set of clauses and the US-passive set of literals are maintained in the form of priority queues that allow user-configurable heuristics to prefer promising clauses and literals.

Dismatching Constraints All clauses are annotated with dismatching constraints that make redundancy due to common ground instances between a clause and its instances explicit. Thus we can block redundant instantiations in the Inst-Gen-Eq-Loop and most crucially redundant proofs in the US-Loop.

[1] http://www.eprover.org

Demodulation In addition to superposition inferences, the US-Loop simplifies literals with demodulation inferences with orientable equations obtained from unit clauses.

Indexing Several unification indexes, implemented as non-perfect discrimination trees, make the forward and backward search for unifiable subterms for unit superposition and for matching subterms for demodulation efficient.

Global subsumption iProver-Eq makes use of a global subsumption algorithm for simplifying both ground and non-ground clauses using the ground solver similar to the way it is done in iProver. It also integrates the resolution prover from iProver to obtain short clauses which are propagated to the ground solver and in turn enhance global subsumption.

6 Evaluation

iProver-Eq[2] is still in an early stage of development which has not been focused on efficiency issues yet. We have evaluated the current version of iProver-Eq which integrates CVC3 as its ground solver, on the standard TPTP v4.0.1 benchmark library. Running on Intel Xeon Quad Core machines with 2.33GHz and 3GB of memory, 5004 out of the 13783 problems are solved within 60 seconds. These include three problems that are not known to be solved by any other theorem prover. The success in 1621 problems is due to the equational reasoning and iProver did not succeed on them with the previous axiomatic handling of equations using CVC3 as a SAT solver. [3]

At the moment the core US component taken as a stand-alone reasoner for unit equations is not as efficient as dedicated superposition-based provers. As an obvious next step we are working on strengthening the US component, however, as we have demonstrated in Section 4, not all techniques from state-of-the-art reasoners can be straightforwardly adapted due to the requirement to generate all relevant instances.

References

1. Barrett, C., Tinelli, C.: CVC3. In: CADE 19. LNCS, vol. 4590, pp. 298–302. Springer (2007)
2. Baumgartner, P.: Logical Engineering with Instance-Based Methods. In: CADE 21. LNCS, vol. 4603, pp. 404–409. Springer (2007)
3. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: SAT 2003. Selected Revised Papers. LNCS, vol. 2919, pp. 502–518. Springer (2004)
4. Ganzinger, H., Korovin, K.: New Directions in Instantiation-Based Theorem Proving. In: LICS 2003. pp. 55–64. IEEE (2003)
5. Ganzinger, H., Korovin, K.: Integrating Equational Reasoning into Instantiation-Based Theorem Proving. In: CSL 2004. LNCS, vol. 3210, pp. 71–84. Springer (2004)
6. Korovin, K.: iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In: IJCAR 2008. LNCS, vol. 5195, pp. 292–298. Springer (2008)
7. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)
8. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning. Elsevier (1999)

[2] Available from http://www.cs.man.ac.uk/~korovink/iprover

[3] In SAT solving, CVC3 has an overhead due to its theory handing. For non-equational problems, one should return to the more efficient MiniSat which we did not do in the experiments.