A Note on Model Representation and Proof Extraction in the First-Order Instantiation-based Calculus Inst-Gen

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

Abstract: We describe the recent extensions of the instantiation-based theorem prover iProver to generate models of satisfiable and proofs for unsatisfiable inputs, both features being demanded by applications [1, 2].

1 Introduction

The main idea behind instantiation-based methods for first-order logic is to combine efficient ground reasoning with smart first-order instantiations. We have been developing the Inst-Gen calculus [4] and its equational variant Inst-Gen-Eq [6], where the ground reasoning is decoupled from instantiation, thus allowing one to employ efficient off-the-shelf SAT and SMT solvers for ground reasoning. This is unlike traditional calculi for first-order logic such as resolution or superposition, where first-order reasoning is tackled purely by applying first-order inference rules.

We focus on two aspects important to applications and users of automated theorem provers, namely the output of models and proofs. In the Inst-Gen method models can be extracted from a saturation of the input and we discuss how to obtain compact model representations for satisfiable input. Since the ground reasoning is delegated to a black-boxed solver, the extraction of proofs of unsatisfiability relies on the ground solver and we demonstrate how to use unsatisfiable cores from the ground solver in proof extraction without a loss of performance during the proof procedure.

2 The Inst-Gen Method

The basic idea of the Inst-Gen method is as follows. The input set of first-order clauses S is abstracted to a set of ground clauses $S\bot$ by mapping all variables to the same ground term, conventionally named $\bot$. If this ground abstraction is unsatisfiable, then the set of first-order clauses is also unsatisfiable. Otherwise, there is a ground model $I_\bot$ for the abstraction that is used to guide an instantiation process. The ground satisfiability check and construction of a ground model is delegated to a solver for satisfiability modulo theories (SMT) in the presence of equations or to a propositional (SAT) solver if no equational reasoning is required.

The ground model $I_\bot$ obtained from the solver 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. 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.

The first-order instantiation process is guided by means of a selection function based on the ground model $I_\bot$. The *selection function* sel assigns to each first-order clause C in S exactly one literal $\mathrm{sel}(C) = L$ 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 $I_\bot$.

We also employ a constraint mechanism for redundancy elimination that becomes crucial in model construction. Intuitively, a clause C represents all its ground instances, hence there are ground instances represented by both the clause C and an instance $C\sigma$ of it. In order to eliminate this duplication we attach a constraint to each clause, effectively blocking all ground instances of clause C that are represented by the instance $C\sigma$. We view such a *dismatching constraint* Φ as a set of substitutions $\{\tau_1, \ldots, \tau_n\}$ and say that a substitution σ *satisfies* the dismatching constraint Φ if it is not more specific than any $\tau_i \in \Phi$.

For simplicity we only consider the non-equational calculus Inst-Gen, the results about model construction and proof extraction apply with some modification also to the equational calculus Inst-Gen-Eq. The following inference rule is applied to the input clause set up to saturation.

Inst-Gen inference rule

$$\frac{C \vee L \mid \Phi \quad D \vee \overline{L'} \mid \Psi}{(C \vee L)\,\sigma \quad \left(D \vee \overline{L'}\right)\sigma}$$

(i) $\sigma = \mathrm{mgu}(L, L')$, (ii) $\mathrm{sel}(C \vee L) = L$, (iii) $\mathrm{sel}(D \vee \overline{L'}) = \overline{L'}$, (iv) σ satisfies Φ and Ψ.

Both clause instances $(C \vee L)\,\sigma$ and $\left(D \vee \overline{L'}\right)\sigma$ are added to the clause set and the dismatching constraints of the premises are extended to $\Phi \cup \{\sigma\}$ and $\Psi \cup \{\sigma\}$.

The Inst-Gen inference rule is similar to the Resolution inference rule, but instead of resolving away the complementary unifiable literals L and $\overline{L'}$ and combining the two premises into the new clause $(C \vee D)\,\sigma$, the clauses are instantiated with the most general unifier σ. We view a literal L as representing all its ground instances, hence having both the literal L and a unifiable complement $\overline{L'}$ selected means that there are ground instances $L\sigma$ and $\overline{L'}\sigma$ represented that are contradictory. The ground model, that only

contains the abstractions $L\bot$ and $\overline{L'}\bot$, therefore cannot be extended to a first-order model due to this conflict between the first-order literals L and $\overline{L'}$. After adding the clause instances with the mgu σ the ground solver can witness this conflict and evolve the model of the ground abstraction appropriately. On the first-order level the dismatching constraint on the premises is extended with σ.

The Inst-Gen method is refutationally complete, that is, from an unsatisfiable input an exhaustive application of the inference rule will eventually lead to an unsatisfiable ground abstraction. On the other hand, if the clause set becomes saturated under the inference rule, it is satisfiable and we can extract a model.

3 Model Representation

If the set of clauses is closed under Inst-Gen inferences and the ground abstraction is satisfiable, then there is no first-order conflict on selected literals and the set of selected literals can indeed be interpreted as a model for the input clause set.

Instantiation-based methods generate a certain class of models that is commonly described with a *disjunction of implicit generalisation (DIG)* [3]. Instead of representing Inst-Gen models as DIGs we represent them as predicate definitions over the term algebra.

For each literal $(\neg)P(t_1,\ldots,t_n) \in \mathrm{sel}(S)$ that contains the variables $\bar{x} = \langle x_1,\ldots,x_m\rangle$ and occurs in a constrained clause $C \mid \Phi$ with $\Phi = \{\tau_1,\ldots,\tau_k\}$ we define

$$\begin{aligned}&\Lambda_{(\neg)P(t_1,\ldots,t_n)}(y_1,\ldots,y_n) \rightleftharpoons\\ &\exists\bar{x}[\, y_1 = t_1 \wedge \cdots \wedge y_n = t_n \wedge\\ &\quad \forall\bar{z}_1(\, x_1 \neq x_1\tau_1 \vee \cdots \vee x_m \neq x_n\tau_n) \wedge \cdots \wedge\\ &\quad \forall\bar{z}_k(\, x_1 \neq x_1\tau_k \vee \cdots \vee x_m \neq x_n\tau_k)],\end{aligned}$$

where $\bar{y} = \langle y_1,\ldots,y_n\rangle$ is a tuple of fresh variables and $\mathrm{var}(\mathrm{rng}(\tau_i)) \subseteq \bar{z}_i$ (here $\bar{z}_i \cap \bar{x} = \emptyset$). The first conjunct containing the existential quantifier corresponds to a flattening of the atom $P(t_1,\ldots,t_n)$ into $P(y_1,\ldots,y_n)$, the remaining conjuncts express the dismatching constraints.

Given an Inst-Gen saturated set of clauses S we can extract several models. For each predicate we can collect either all *positive occurrences* or dually all *negative occurrences* and define the predicate in the model as

$$(\neg)P(y_1,\ldots,y_n) \iff \bigvee_{(\neg)P(t_1,\ldots,t_n)\in\mathrm{sel}(S)} \Lambda_{(\neg)P(t_1,\ldots,t_n)}.$$

Since the sizes of positive and negative representations can be vastly different, for each atom we can chose the smallest. Alternatively we can use *implied definitions* of the form

$$(\neg)P(y_1,\ldots,y_n) \Leftarrow \bigvee_{(\neg)P(t_1,\ldots,t_n\in\mathrm{sel}(S)} \Lambda_{(\neg)P(t_1,\ldots,t_n)},$$

and allow completing the model arbitrarily if undefined.

All model representations have been implemented in the iProver system, such that the user can choose the one most fit for purpose. The implementation keeps both the set of active literals as well as the dismatching constraints compactly stored in discrimination trees, thus the model can be efficiently constructed.

4 Proof Extraction

As soon as the ground solver finds the ground abstraction unsatisfiable, the input clause set has been proved unsatisfiable. Since we regard the ground solver as a black box and proof extraction in SAT solving may cause a considerable degradation in performance, we content ourselves with unsatisfiable cores returned from the solver. Then, a proof of unsatisfiability is a sequence of first-order inferences up to a set of clauses that is propositionally unsatisfiable.

We record each first-order inference step in iProver and run two instances of the ground solver in parallel, both containing the ground abstraction of the current clause set. The first solver instance is frequently called to check satisfiability, the second instance is used only a posteriori to obtain an unsatisfiable core. For this purpose we add a unique literal to each clause in the second instance and assume the complement of each tracking literal. The falsified assumptions represent an unsatisfiable core that we can minimise.

After the first instance of the ground solver reports unsatisfiability of the ground abstraction, the second instance is invoked for the first time. We map the failed assumptions to their first-order clauses and recursively trace each inference back to premises which are input clauses. Since proof extraction is a separate step, the performance of the proof search is not affected.

The iProver system makes use of global propositional subsumption [5], which simplifies a clause with respect to the some grounding. Since justifying each simplification step during the proof search would result in a severe performance hit, we postpone this step until the actual proof extraction. If in tracing the proof tree a clause is encountered that was obtained by global propositional subsumption, the second instance of the ground solver is invoked once more to find the justification for the simplification, a set of clauses younger than the simplified clause. We then continue the search for input clauses from this set.

We note that for many applications proofs in this form are sufficient. In particular the Sledgehammer tool of the Isabelle system [1] uses automated theorem provers essentially to filter an input clause set for a first-order unsatisfiable core, hence it requires much less than a detailed proof and can work with the output generated here.

References

[1] J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT Solvers. In *CADE 23*, pages 116–130, 2011.

[2] M. Emmer, Z. Khasidashvili, K. Korovin, and A. Voronkov. Encoding Industrial Hardware Verification Problems into Effectively Propositional Logic. In *FMCAD 2010*, 2010.

[3] C. G. Fermüller and R. Pichler. Model Representation via Contexts and Implicit Generalizations. In *CADE 20*, pages 409–423, 2005.

[4] H. Ganzinger and K. Korovin. New Directions in Instantiation-Based Theorem Proving. In *LICS 2003*, pages 55–64, 2003.

[5] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In *IJCAR 2008*, pages 292–298, 2008.

[6] K. Korovin and C. Sticksel. Labelled Unit Superposition Calculi for Instantiation-Based Reasoning. In *LPAR-17*, pages 459–473, 2010.