Model Finding for Recursive Functions in SMT[*]

Andrew Reynolds[1], Jasmin Christian Blanchette[2,3], and Cesare Tinelli[4]

[1] École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
[2] Inria Nancy & LORIA, Villers-lès-Nancy, France
[3] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[4] Department of Computer Science, University of Iowa, USA

Many solvers based on SMT (satisfiability modulo theories) can reason about quantified formulas using incomplete instantiation-based methods [3,7]. These methods work well in the context of proving (i..e, showing unsatisfiability), but they are of little help for finding models (i.e., showing satisfiability). Often, a single universal quantifier in one of the axioms of a problem is enough to prevent the discovery of models.

In the past few years, techniques have been developed to find models for quantified formulas in SMT. Ge and de Moura [4] introduced a complete instantiation-based procedure for formulas in the essentially uninterpreted fragment. This fragment is limited to universally quantified formulas where all variables occur as direct subterms of uninterpreted functions—e.g., $\forall x.\ \mathsf{f}(x) \approx \mathsf{g}(x) + 5$. Other syntactic criteria extend this fragment slightly, including cases when variables occur as arguments of arithmetic predicates. Subsequently, Reynolds et al. [8,9] introduced techniques for finding finite models for quantified formulas over uninterpreted types and types having a fixed finite interpretation. These techniques can find a model for a formula such as $\forall x, y : \tau.\ x \approx y \vee \neg\, \mathsf{f}(x) \approx \mathsf{f}(y)$, where τ is an uninterpreted type.

Unfortunately, none of these fragments can accommodate the vast majority of quantified formulas that correspond to recursive function definitions: The essentially uninterpreted fragment does not allow the argument of a recursive function to be used inside a complex term on the right-hand side, whereas the finite model finding techniques are not applicable for functions over infinite domains such as the integers or algebraic datatypes. A simple example where both approaches fail is $\forall x : \mathsf{Int}.\ \mathsf{ite}\big(x \leq 0,\ \mathsf{p}(x) \approx 1,\ \mathsf{p}(x) \approx 2 * \mathsf{p}(x-1)\big)$. This state of affairs is unsatisfactory, given the frequency of recursive definitions in practice and the impending addition of a dedicated command for introducing them, `define-fun-rec`, to the SMT-LIB standard [1].

We present a method for translating formulas involving recursive function definitions into formulas where finite model finding techniques can be applied. The recursive functions must meet a semantic criterion to be admissible. This criterion is met by well-founded (terminating) recursive function definitions.

We define a translation for a class of formulas involving admissible recursive function definitions. The main insight is that a recursive definition $\forall x : \tau.\ \mathsf{f}(x) \approx t$ can be translated to $\forall a : \alpha_\tau.\ \mathsf{f}(\gamma_\mathsf{f}(a)) \approx t[\gamma(a)/x]$, where α_τ is an uninterpreted abstract type and γ_f converts the abstract type to the concrete type. The translation preserves satisfiability and unsatisfiability, and makes finite model finding possible for problems in this class.

[*]This research is partially supported by the Inria technological development action "Contre-exemples Utilisables par Isabelle et Coq" (CUIC).

Our empirical evaluation on benchmarks from the IsaPlanner proof planner [5] and the Leon verifier [2] provides evidence that this translation improves the effectiveness of the SMT solvers CVC4 and Z3 for finding counterexamples to verification conditions. The approach is implemented as a preprocessor in CVC4 .

In future work, it would be interesting to evaluate the approach against other counterexample generators, notably Leon and Nitpick, and enrich the benchmark suite with problems exercising CVC4's support for coalgebraic datatypes [6]. We also plan to integrate CVC4 as a counterexample generator in proof assistants. Finally, future work could also include identifying further sufficient conditions for admissibility, thereby enlarging the applicability of the translation scheme presented here.

This abstract is based on a regular submission to the SMT 2015 workshop, which itself is based on a longer technical report. Both are available online.[1]

References

[1] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB standard—Version 2.5. `http://smt-lib.org/language.shtml`, to appear.

[2] R. Blanc, V. Kuncak, E. Kneuss, and P. Suter. An overview of the Leon verification system—Verification by translation to recursive functions. In *Scala '13*. ACM, 2013.

[3] L. de Moura and N. Bjørner. Efficient E-matching for SMT solvers. In F. Pfenning, editor, *CADE-21*, volume 4603 of *LNCS*, pages 183–198. Springer, 2007.

[4] Y. Ge and L. de Moura. Complete instantiation for quantified formulas in satisfiability modulo theories. In *CAV '09*, volume 5643 of *LNCS*, pages 306–320. Springer, 2009.

[5] M. Johansson, L. Dixon, and A. Bundy. Case-analysis for rippling and inductive proof. In *ITP 2010*, pages 291–306, 2010.

[6] A. Reynolds and J. C. Blanchette. A decision procedure for (co)datatypes in SMT solvers. In A. Felty and A. Middeldorp, editors, *CADE-25*, LNCS. Springer, 2015.

[7] A. Reynolds, C. Tinelli, and L. de Moura. Finding conflicting instances of quantified formulas in SMT. In *FMCAD 2014*, pages 195–202. IEEE, 2014.

[8] A. Reynolds, C. Tinelli, A. Goel, and S. Krstić. Finite model finding in SMT. In N. Sharygina and H. Veith, editors, *CAV 2013*, volume 8044 of *LNCS*, pages 640–655. Springer, 2013.

[9] A. Reynolds, C. Tinelli, A. Goel, S. Krstić, M. Deters, and C. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In M. P. Bonacina, editor, *CADE-24*, volume 7898 of *LNCS*, pages 377–391. Springer, 2013.

[1] `http://lara.epfl.ch/~reynolds/SMT2015-recfun/`