

Testing First-Order Logic Axioms in AutoCert

Ki Yung Ahn[†] and Ewen Denney[‡]

[†] Mission Critical Technologies, Inc. / NASA Ames Research Center
Moffett Field, CA 94035, USA (2009 Summer Internship)
Department of Computer Science, Portland State University
Portland, Oregon, USA
`kya@cs.pdx.edu`

[‡]Stinger Ghaffarian Technologies, Inc. / NASA Ames Research Center
Moffett Field, CA 94035, USA
`Ewen.W.Denney@nasa.gov`

Background: AutoCert [2] is a formal verification tool for machine generated code in safety critical domains, such as aerospace control code generated from MathWorks Real-Time Workshop. AutoCert uses Automated Theorem Provers (ATPs) [5] based on First-Order Logic (FOL) to formally verify safety and functional correctness properties of the code. These ATPs try to build proofs based on user provided domain-specific axioms, which can be arbitrary First-Order Formulas (FOFs). These axioms are the most crucial part of the trusted base, since proofs can be submitted to a proof checker – removing the need to trust the prover – and AutoCert itself plays the part of checking the code generator.

However, formulating axioms correctly (i.e. precisely as the user had really intended) is non-trivial in practice. The challenge of axiomatization arise from several dimensions. First, the domain knowledge has its own complexity. AutoCert has been used to verify mathematical requirements on navigation software that carries out various geometric coordinate transformations involving matrices and quaternions. Axiomatic theories for such constructs are complex enough that mistakes are not uncommon. Second, adjusting axioms for ATPs can add even more complexity. The axioms frequently need to be modified in order to have them in a form suitable for use with ATPs. Such modifications tend to obscure the axioms further. Thirdly, speculating validity of the axioms from the output of existing ATPs is very hard since theorem provers typically do not give any examples or counterexamples.

Our Current Work: We adopt the idea of model-based testing to aid axiom authors in discovering errors in axiomatization. To test the validity of axioms, users define a computational model of the axiomatized logic by giving interpretations to each of the function symbols and constants as computable functions and data constants in a simple declarative programming language. Then, users can test axioms against the computational model with widely used software testing tools. The advantage of this approach is that the users have a concrete intuitive model with which to test validity of the axioms, and can observe counterexamples when the model does not satisfy the axioms.

We have implemented a proof of concept tool using Template Haskell [4], QuickCheck [1], and Yices [3], as illustrated in Figure 1. An axiom in TPTP syntax is parsed and automatically translated into a lambda term using Template Haskell. Given a user provided interpretation (e.g. the logical symbol `lt` can be interpreted as the inequality function `<`), the lambda term becomes an executable function which can then be used as a property in QuickCheck, a property-based testing framework for Haskell. The next step is to use a combination of domain-specific generators and Yices to automatically synthesize test generators that generate only the inputs satisfying the premises of the axiom formula, thus avoiding vacuous tests. We sometimes need to patch or fill in unconstrained values (e.g. solutions for x satisfying $x + 0 = x$) from the results

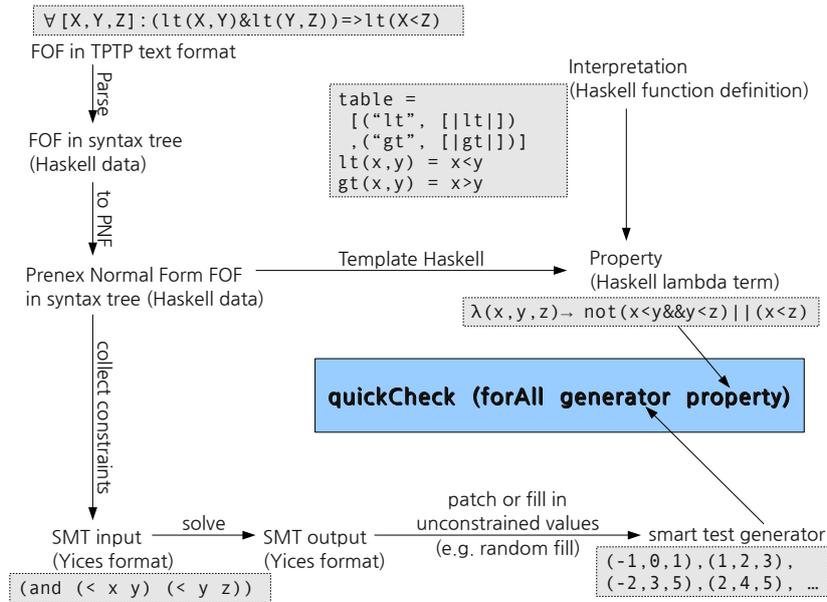


Figure 1: Axiom Testing Framework Design

of Yices. Then, we can invoke QuickCheck over the property combined with the test generator: i.e. `quickCheck (forAll generator property)`.

Future Work: We are currently extending the framework in several ways. First, we have begun to apply the axiomatic testing framework to test mathematical library functions implemented in C or C++, via the foreign function interface of Haskell. The functions are specified using logical formulas and the testing machinery can be used to generate black box tests. Second, we are looking for ways to generate test cases for certain non-linear constraints (e.g. constraints involving trigonometry) that are not solvable by SMT solvers like Yices. Third, we aim to test the verification conditions (VCs) that are generated by AutoCert. VCs are first-order formulae that are sufficient to show that given safety requirements hold on the analyzed code. Failure of a VC either means that (1) the code is unsafe, (2) the prover ran out of resources, or (3) the axiomatization is incorrect or incomplete. If testing fails to generate a counterexample to a VC, then it is more likely to be (2) or (3), rather than (1).

References

- [1] Koen Claessen and John Hughes. QuickCheck: a lightweight tool for random testing of haskell programs. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, pages 268–279, September 2000.
- [2] Ewen Denney and Steven Trac. A software safety certification tool for automatically generated guidance, navigation and control code. In *IEEE Aerospace Conference*, March 2008.
- [3] Bruno Dutertre and Leonardo de Moura. The YICES SMT solver. Tool paper at <http://yices.csl.sri.com/tool-paper.pdf>, 2006.
- [4] Tim Sheard and Simon Peyton Jones. Template metaprogramming for Haskell. In *ACM SIGPLAN Haskell Workshop 02*, pages 1–16. ACM Press, October 2002.
- [5] Geoff Sutcliffe. System description: SystemOn TPTP. In *CADE*, volume 1831 of *Lecture Notes in Computer Science*, pages 406–410. Springer, 2000.