Reflection-based Integration of SMT Solvers into Theorem Provers

by

Yan Peng

B. Engineering, Zhejiang University, 2012

M. Science, University of British Columbia, 2015

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

April 2192

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Reflection-based Integration of SMT Solvers into Theorem Provers

submitted by **Yan Peng** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Computer Science**.

Examining Committee:

Mark R. Greenstreet, Department of Computer Science, University of British Columbia
Supervisor

Alan J. Hu, Department of Computer Science, University of British Columbia
Supervisory Committee Member

Ronald Garcia, Department of Computer Science, University of British Columbia
Supervisory Committee Member

Matt Kaufmann, Department of Computer Sciences, University of Texas at Austin
Supervisory Committee Member

xx, Department
University Examiner

xx, Department
External Examiner

Abstract

Satisfiability Modulo Theories (SMT) solvers are automated decision procedures for domain-specific solving problems including but not limited to boolean, integer, and real theories, uninterpreted function theories, array theories, and bit-vector theories. They have become a main method widely applied in the industry for solving software and hardware formal verification problems. Though theorem provers are largely interactive, they are often resorted to for its powerful induction capability, abstraction capability, and strong proof management support. Any moderate-complexity problem will require both methods but switching between tools requires modelling the same system in each tool. This introduces huge verification overhead and could be erroneous.

Existing work on the integration of SMT solvers and theorem provers focuses on proof reconstruction for soundness, but lacks emphasis on the usability and proof efficiency. To use such integration, the theorem prover term needs to be in a form that is ready for direct translation. On the other hand, since proof reconstruction is hard, it is reported that for some problems the lemma could be time consuming and even unsolvable. In this thesis, We stress the problem of the usability of such integration. We propose using reflection for integrating SMT solvers into theorem provers. We note that using proof reconstruction to remove the dependency on the soundness of the SMT solver is an orthogonal research direction.

We demonstrate our method using the ACL2 theorem prover and the Z3 SMT solver. To bridge the gap between the untyped first-order logic of ACL2 and the many-sorted logic of SMT solvers, we apply reflection over the ACL2 terms to achieve adding user hypotheses, function expansion, type inference, term replacement and many other transformations. By using reflection, the user is freed from

manually transforming theorem prover terms into a shape that is directly translatable. In addition, soundness of these transformations is easily achieved and does not require extra proof time, therefore promotes proof efficiency. The framework is also built to be extensible, allowing new transformations to be integrated. We analyze our tool by conducting formal verification of asynchronous circuits and machine learning convex optimization problems.

Lay Summary

The lay or public summary explains the key goals and contributions of the research/ scholarly work in terms that can be understood by the general public. It must not exceed 150 words in length.

Preface

At University of British Columbia (UBC), a preface may be required. Be sure to check the Graduate and Postdoctoral Studies (GPS) guidelines as they may have specific content to be included.

Table of Contents

List of Tables

List of Figures

Glossary

SMT Satisfiability Modulo Theories

GPS Graduate and Postdoctoral Studies

Acknowledgments

Thank those people who helped you.

Don’t forget your parents or loved ones.

You may wish to acknowledge your funding sources.

Chapter 1

Introduction

1.1 Some Section

some citation[1]

some citation[2]

Chapter 2

Related Work

Chapter 3

The Reflection-based Integration Architecture

The many differences between the ACL2 theorem prover and an SMT solver like Z3 proposes many challenges for the integration.

First, the logic of the ACL2 theorem prover is ~~first-order~~, while the logic of the SMT solvers is many-sorted. Transforming a theorem statement in ACL2 into a logical formula in an SMT solver, requires that such a theorem statement to be well-typed with respect to the type system of the SMT solver.

Second, a theorem statement in ACL2 might not be readily translatable to the SMT language. For example, the term might contain user-defined functions and even recursive functions that are not readily translatable. The term might use polymorphic functions or values, while an SMT solver expects functions or values for a given sort. For example, in ACL2 one can write `(cons '1 'nil)` where `cons` is polymorphic, but in Z3, one needs to write `IntegerList.cons(1, IntegerList.nil)` where `cons` is a constructor function for the sort `IntegerList` and `nil` is a value of sort `IntegerList`. There might exist ACL2 features that do not have direct correspondence in an SMT solver. For example, alist types and the operations over an alist.

Thirdly, even though some functions or values seem to have a direct correspondence in the SMT solver, the slightest discrepancy might cause unsoundness in the integration. For example, adding a number with a boolean value in

ACL2 is equivalent to adding a number to the value 0, so `(+ 1 t)` evaluates to 1. However, in Z3py[1], `1+True` evaluates to 2. Another example is when taking `car` of an empty list. In ACL2, `(car nil)` evaluates to `nil` while in Z3, `IntegerList.car(IntegerList.nil)` returns an arbitrary value of the type `IntegerList`.

To address these challenges, we develop a reflection-based integration architecture for Smtlink. This architecture takes a theorem statement in ACL2, applies several sound transformation steps to generate a goal that is well-typed under the type system of the SMT solver and readily translatable to an SMT term that preserves the logical meaning of the theorem statement. A final transliteration step is then applied to generate an SMT term. This architecture takes heavy use of the metaprogramming and reflection techniques.

Metaprogramming is a programming technique that allows us to take programs as input and operates over the programs. A Metaprogramming program can analyze and transform an input program and possibly output a new program. Reflection refers to the ability of applying metaprogramming to a program in its own programming language. For example, in the LISP programming language, all values are s-expressions, including LISP programs themselves. This makes it very easy to write LISP programs that takes another program, which is basically an s-expression, as input, and interpret and generate new programs.

ACL2 as a computational logic for applicative Common Lisp, benefits from its reflexive metaprogramming features. In ACL2, a theorem is also an s-expression. Reflection allows us to take a theorem as input, inspect and transform a theorem into another theorem. In addition to structural reflection, ACL2 also provides a way of proving that the reflexive metaprogramming preserves the logical meaning of the original theorem. We can then build an architecture by pipelining a series of these reflexive programs. Each reflexive program takes a theorem that's the output of a previous reflexive program, inspect and transform the input theorem and output the resulting theorem to another reflexive program. This way, we modularize the architecture into several well-contained steps. For example, one of the steps is to do function expansion. This allows us to solve the issue that user-defined functions

[1] Z3py is the Python interface for the SMT solver Z3.

are not readily translatable to the SMT language.

The reflection-based Smtlink architecture has several advantages. First, the architecture is modularize and therefore is easily extensible. An intrepid user might want to extend the Smtlink architecture for their own use, and all they have to do is to add a step into the pipeline of reflexive transformations. Second, except for the last step that translates directly from an ACL2 theorem into an SMT term, the former steps in the pipeline are all verified to be sound ~~reflections~~ within ACL2. We prove a theorem in ACL2 that ensures that the output theorem for each reflexive step implies the input theorem.

In the rest of this Chapter, we will discuss the reflection-based architecture of Smtlink. In Section **??**, we will introduce several key concepts that enable the logic-preserving reflection in ACL2. In Section **??**, we focus on the high-level picture of the architecture including several guiding examples. In Section **??**, we will discuss each steps in the reflection-based architecture and how reflection is applied. Section **??** summarizes this chapter.

Chapter 4

Soundness

Chapter 5

Formally Verifying the Asynchronous Pipelines

Chapter 6

Formally Verifying the Cauchy-Schwarz Inequality

Chapter 7

Comparing Experiments

Chapter 8

Conclusions

Bibliography

[1] P. J. Cohen. The independence of the continuum hypothesis, 1963. → page 1

[2] A. Einstein. Concerning an heuristic point of view toward the emission and transformation of light. *Annalen Phys*, pages 132–148, 1905. → page 1

Appendix A

Supporting Materials

This would be any supporting material not central to the dissertation. For example:

- additional details of methodology and/or data;
- diagrams of specialized equipment developed.;
- copies of questionnaires and survey instruments.