Combining Interactive and Automatic Proofs in First-Order Theories
(research proposal – 2014)

Andrés Sicard-Ramírez

Logic and Computation Seminar
EAFIT University
31 May 2013
Team Work

- Andrés Sicard-Ramírez (main researcher)
- Juan Fernando Ospina-Giraldo (advisor)
- Jorge Ohel Acevedo-Acosta (master student of Applied Mathematics)
- José Luis Echeverri-Jurado (master student of Applied Mathematics)
SMT Solvers

Satisfiability Modulo Theories

The study of automatic methods for checking the satisfiability of first-order formulae with respect to some background theory is called Satisfiability Modulo Theories (SMT), and SMT systems are usually referred as SMT solvers.¹

SMT Solvers

Satisfiability Modulo Theories

The study of automatic methods for checking the satisfiability of first-order formulae with respect to some background theory is called Satisfiability Modulo Theories (SMT), and SMT systems are usually referred as SMT solvers.¹

Background theories

- Real numbers
- Integers
- Lists
- Strings
- ...

**Overview**

Alt-Ergo is an open source automatic theorem prover dedicated to program verification. It is an SMT solver based on $\text{CC}(X)$: a congruence closure algorithm parameterized by an equational theory X. Alt-Ergo is based on a home-made SAT-solver and implements an instantiation mechanism for quantified formulas. Its architecture is summarized by the following picture.
MathSAT 5
An SMT Solver for Formal Verification & More

Introduction

Welcome to the home page of MathSAT 5, an efficient Satisfiability modulo theories (SMT) solver. MathSAT 5 is the successor of MathSAT 4, supporting a wide range of theories (including e.g. equality and uninterpreted functions, linear arithmetic, bit-vectors, and arrays) and functionalities (including e.g. computation of Craig interpolants, extraction of unsatisfiable cores, generation of models and proofs, and the ability of working incrementally).

MathSAT 5 is a joint project of FBK-IRST and DISI-University of Trento.
The veriT SMT Solver

An open, trustable and efficient SMT-solver

What is veriT?

veriT is a SMT (Satisfiability Modulo Theories) solver. It is open-source, proof-producing, and complete for quantifier-free formulas with uninterpreted functions and difference logic on real numbers and integers.

The input format is the SMT-LIB language (both versions 1.2 and 2.0) and DIMACS, but veriT can also be used as a standalone library and incorporated in third-party software. The tool is open-source and distributed under the BSD license.

veriT is complete for the logic of unquantified formulas over uninterpreted symbols, difference logic over integers and real numbers, and the combination thereof. This corresponds to the logics identified as QF_IDL, QF_RDL, QF_UF and QF_UFIDL in the SMT-LIB benchmarks. veriT includes quantifier reasoning capabilities through quantifier instantiation heuristics and the integration of a first-order prover and linear arithmetic for integers and real numbers.

veriT has proof-production capabilities that may be used or checked by external tools. Although not (yet) as fast as the solvers performing best in the SMT competition, veriT has a decent efficiency.

The ancestor of veriT is halRvey. Its web page can still be reached here.

How to contribute?

veriT is under heavy development, and newcomers to the project are most welcome! Check the Job section of this site, and contact us.
The Z3 SMT Solver

Z3 is a high-performance theorem prover being developed at Microsoft Research.

- Try Z3 online at RiSE4Fun using Python or SMT 2.0
- Follow Z3 on Facebook
- Browse Z3 Q&A at StackOverflow
- Read our FAQ
- Leo de Moura's Blog
SMT-LIB

The Satisfiability Modulo Theories Library

SMT-LIB is an international initiative aimed at facilitating research and development in Satisfiability Modulo Theories. Since its inception in 2003, the initiative has pursued these aims by focusing on the following concrete goals:

- provide standard rigorous descriptions of background theories used in SMT systems;
- develop and promote common input and output languages for SMT solvers;
- establish and make available to the research community a large library of benchmarks for SMT solvers.

SMT-LIB was created with the expectation that the availability of common standards and a library of benchmarks would greatly facilitate the evaluation and the comparison of SMT systems, and advance the state of the art in the field in the same way as, for instance, the TPTP library has done for theorem proving, or the SATLIB library has done initially for SAT.
Example (Satisfiability)

See example from the Z3 tutorial.
Example (Validity (excluded-middle.smt))

; QF_UF: Unquantified formulas built over a signature of uninterpreted (i.e., free) sort and function symbols.

(set-logic QF_UF)
(declare-const p Bool)
(define-fun conjecture () Bool (or p (not p)))
(assert (not conjecture))
(check-sat)
Example (cont.)

$ alt-ergo-0.95.1-x86_64 excluded-middle.smt2
unsat

$ cvc4-1.2-x86_64-linux-opt --lang smt2 excluded-middle.smt
unsat

$ z3 -smt2 excluded-middle.smt
unsat
Automatising induction

Automatic inductive theorem proving is an area with a long tradition. Inductive theorems provers (ITPs) are based on different paradigms (for example, implicit induction\(^1\), explicit induction\(^2\) or *descente infinie*\(^3\)) and heuristics.

---


Formale Methoden und Deduktion
Prof. Dr. J. Avenhaus

How to Prove Inductive Theorems? QuodLibet!

Our research activities have their origins in the D4-Project (Reasoning in Equationally Defined Structures) which was funded within the former Sonderforschungsbereich 314. Throughout the last few years our overall goal has been the design and implementation of a rewrite-based first-order inductive theorem prover. As to the prover’s main application area we intend to use it for the (algebraic) specification of and formal reasoning about data types such as the natural numbers, lists, strings, graphs etc.

The formal basis of our inductive theorem proving system QuodLibet is given by a logical framework for inductive theorem proving (ITP) that essentially consists of a specification language for the formalization of data types, a calculus for inductive proofs and so-called proof state graphs as a means of representing the various kinds of dependencies among formulas in proofs.
The SPIKE Prover

SPIKE is an automated theorem prover using formula-based induction. It is written in Objective Caml.

To get the prover, run the command

svn checkout http://spike-prover.googlecode.com/svn/ spike-prover-read-only

then read the README file from the 'trunk' directory.

NEW !!! Spike can be called from the Coq proof assistant using a tactic that automatically performs lazy and mutual induction. We provide a zip file with the Coq scripts using this tactic.
Goals

General goal

Formalise first-order theorems belonging to some first-order theories by combining interactive proofs performed in the Agda proof assistant with automatic proofs performed by SMT solvers.
Goals

General goal
Formalise first-order theorems belonging to some first-order theories by combining interactive proofs performed in the Agda proof assistant with automatic proofs performed by SMT solvers.

Specific goals
- Compare the capabilities of ATPs and SMT solvers with the empty theory.
General goal

Formalise first-order theorems belonging to some first-order theories by combining interactive proofs performed in the **Agda** proof assistant with automatic proofs performed by SMT solvers.

Specific goals

- Compare the capabilities of ATPs and SMT solvers with the empty theory.

- Use SMT solvers with some concrete theories.
Goals

General goal
Formalise first-order theorems belonging to some first-order theories by combining interactive proofs performed in the Agda proof assistant with automatic proofs performed by SMT solvers.

Specific goals
- Compare the capabilities of ATPs and SMT solvers with the empty theory.
- Use SMT solvers with some concrete theories.
- Elaborate case studies related to using the SMT solvers.
Goals

General goal
Formalise first-order theorems belonging to some first-order theories by combining interactive proofs performed in the Agda proof assistant with automatic proofs performed by SMT solvers.

Specific goals
- Compare the capabilities of ATPs and SMT solvers with the empty theory.
- Use SMT solvers with some concrete theories.
- Elaborate case studies related to using the SMT solvers.
- Study the possibility of using ITPs.
Theoretical Framework

Previous work


First-Order Logic (FOL)

Terms $\exists t ::= x$
- $c$ constant
- $f(t, \ldots, t)$ function

Formulae $\exists A ::= T | \bot$
- $A \Rightarrow A | A \land A | A \lor A$ logical connectives
- $\forall x.A | \exists x.A$ quantifiers
- $t = t$ equality
- $P(t, \ldots, t)$ predicate

Abbreviations

$\neg A \overset{\text{def}}{=} A \Rightarrow \bot$
$t \neq t' \overset{\text{def}}{=} \neg(t = t')$
Subset of Agda expressions

Normal Forms $\exists a ::= x \ a \ \cdots \ a$

| $c \ a \ \cdots \ a$                      | constant          |
| $\lambda x. a$                           | $\lambda$-abstraction |
| $(x : a) \rightarrow a$                  | dependent function type |
Theoretical Framework

Agda required type formers and constants

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbb{N}_0 )</td>
<td>the empty type</td>
</tr>
<tr>
<td>( \mathbb{N}_1 )</td>
<td>the unit type</td>
</tr>
<tr>
<td>+</td>
<td>the disjoint union type</td>
</tr>
<tr>
<td>( \times )</td>
<td>the Cartesian product type</td>
</tr>
<tr>
<td>( \Sigma )</td>
<td>the dependent product type</td>
</tr>
<tr>
<td>I</td>
<td>the identity type</td>
</tr>
<tr>
<td>( f^* )</td>
<td>a function symbol ( f )</td>
</tr>
<tr>
<td>( P^* )</td>
<td>a predicate symbol ( P )</td>
</tr>
<tr>
<td>D</td>
<td>the domain of quantification</td>
</tr>
</tbody>
</table>
FOL translation into Agda expressions

Terms

\[ x^* = x \]
\[ c^* = c \]
\[ (f(t_1, \ldots, t_n))^* = f^* \ t_1^* \ \ldots \ t_n^* \]
FOL translation into Agda expressions (cont.)

Formulae

\[ \bot^* = N_0 \]
\[ \top^* = N_1 \]
\[ (A \lor B)^* = A^* + B^* \]
\[ (A \land B)^* = A^* \times B^* \]
\[ (A \Rightarrow B)^* = A^* \rightarrow B^* \]
\[ (\exists x. A)^* = \Sigma D (\lambda x. A^*) \]
\[ (\forall x. A)^* = (x : D) \rightarrow A^* \]
\[ (t = t')^* = I D t^* t'^* \]
\[ (P(t_1, \ldots, t_n))^* = P^* t_1^* \ldots t_n^* \]
Theoretical Framework

Inductive representation of FOL

Truth

\[ \text{data } \top : \text{Set where } \text{tt} : \top \]

Falsehood

\[ \text{data } \bot : \text{Set where} \]

\[ \bot\text{-elim} : \{ A : \text{Set} \} \to \bot \to A \]
\[ \bot\text{-elim }() \]

Implication

\[ A \to B \text{ (non-dependent function type)} \]

Conjunction

\[ \text{data } (\_ \land \_ ) (A B : \text{Set}) : \text{Set where } \_,\_, : A \to B \to A \land B \]

\[ \land\text{-proj}_1 : \forall \{ A B \} \to A \land B \to A \]
\[ \land\text{-proj}_1 (a , \_) = a \]

\[ \land\text{-proj}_2 : \forall \{ A B \} \to A \land B \to B \]
\[ \land\text{-proj}_2 (\_ , b) = b \]
Inductive representation of FOL (cont.)

**Disjunction**

\[
\text{data } \_ \lor \_ (A \ B : \text{Set}) : \text{Set} \ 
\text{where}
\]

\[
i \_1 : A \to A \lor B
\]

\[
i \_2 : B \to A \lor B
\]

\[
\text{case} : \forall \{A \ B\} \to \{C : \text{Set}\} \to
\]

\[
(A \to C) \to (B \to C) \to A \lor B \to C
\]

\[
\text{case } f \ g \ (i \_1 \ a) = f \ a
\]

\[
\text{case } f \ g \ (i \_2 \ b) = g \ b
\]

**Negation**

\[
\neg : \text{Set} \to \text{Set}
\]

\[
\neg A = A \to \bot
\]

**PEM**

\[
\text{postulate pem} : \forall \{A\} \to A \lor \neg A
\]
**Theoretical Framework**

**Inductive representation of FOL (cont.)**

- **Domain**
  - **Postulate** $D : Set$

- **Universal quantifier**
  - $(x : D) \rightarrow A$ (dependent function type)

- **Existential quantifier**
  - **Data** $\exists (A : D \rightarrow Set) : Set$ where
    - $\_,_ : (t : D) \rightarrow A t \rightarrow \exists A$

  - $\exists$-elim : $\{ A : D \rightarrow Set \} \{ B : Set \} \rightarrow$
    - $\exists A \rightarrow (\forall \{ x \} \rightarrow A x \rightarrow B) \rightarrow B$
    - $\exists$-elim $(_ , A) \_ = h A x$

- **Equality**
  - **Data** $\_ \equiv \_ (x : D) : D \rightarrow Set$ where $\text{refl} : x \equiv x$

  - $\text{subst} : (A : D \rightarrow Set) \rightarrow \forall \{ x y \} \rightarrow x \equiv y \rightarrow A x \rightarrow A y$
  - $\text{subst} A \text{refl} A x = A x$
Theoretical Framework: The Apia Program

Agda file + ATP-pragmas

Modified version of Agda

Agda interface file

Apia

TPTP translation

TPTP formula

calls the ATPs

E

Vampire

Equinox

Metis

SPASS

(Un)proven conjecture
State of the Art

- The *Isabelle* proof assistant and the *Sledgehammer* tool

  Allows us to use ATPs and SMT solvers to prove properties arising in the construction of interactive proofs and makes proof term reconstruction.\(^1\)

---


State of the Art

- The **Isabelle** proof assistant and the **Sledgehammer** tool

  Allows us to use ATPs and SMT solvers to prove properties arising in the construction of interactive proofs and makes proof term reconstruction.\(^1\)

- The **Coq** proof assistant and the **SMTCocq** tool

  The tool provides a certified checker for proof witnesses coming from the SMT solver **veriT** and adds a new tactic named verit, that calls **veriT** on any **Coq** goal.\(^2\)

---
