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

Andrés Sicard-Ramírez

Seminar of the PhD in Mathematical Engineering
Universidad EAFIT
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 (Barret, Sebastiani, Seshia, and Tinelli 2009).

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 (Barret, Sebastiani, Seshia, and Tinelli 2009).

Background theories

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

The Alt-Ergo SMT Solver

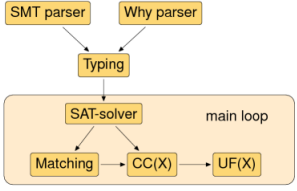
Alt-Ergo

(* AN OCAML SMT-SOLVER FOR SOFTWARE VERIFICATION *)

- Overview
- Download
- People
- Contact

Overview

Alt-Ergo is an open source automatic theorem prover dedicated to program verification. It is an SMT solver based on 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 the following picture.



MathSAT 5

An SMT Solver for Formal Verification & More

Introduction

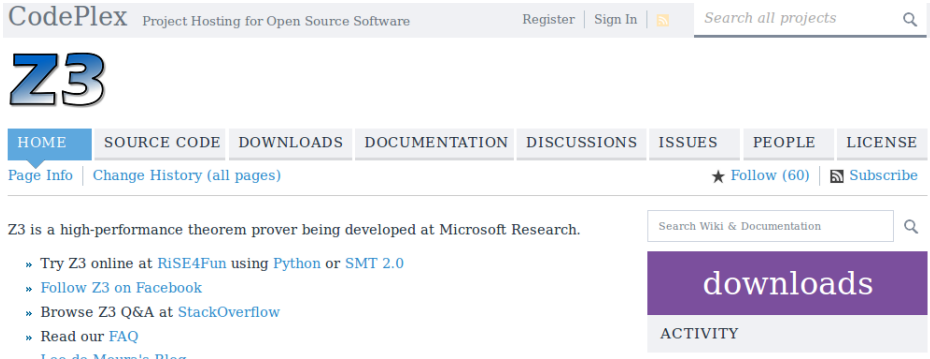
Welcome to the home page of MathSAT 5, an efficient Satisfiability modulo theories (SMT) solver. MathSAT 5 is the successor of MathSAT 4, supporting a wide range of theories (including e.g. equality and uninterpreted functions, linear arithmetic, bit-vectors, and arrays) and functionalities (including 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.

Contents

The veriT SMT Solver

The Z3 SMT Solver



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.

The SMT-LIB v2.0 Language

Example (Satisfiability)

See example from the Z3 tutorial.

The SMT-LIB v2.0 Language

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
^^I(or p (not p)))
(assert (not conjecture))
(check-sat)
```

The SMT-LIB v2.0 Language

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
```

Inductive Theorems Provers

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 (Kapur and Musser 1987), explicit induction (Walther 1994) or *descente infinie* (Wirth 2012)) and heuristics.

The QuodLibet ITP

Formale Methoden und Deduktion
Prof. Dr. J. Avenhaus

FB Inf

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 ITP

spike-prover

a theorem prover

Search projects

Project Home | Downloads | Wiki | Issues | Source

Summary | People

Project Information

Recommend this on Google

Starred by 1 user

Project feeds

Code license

New BSD License

Labels

SPIKE

Members

sorinica@gmail.com

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.

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.

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

- A. Bove, P. Dybjer, and A. Sicard-Ramírez (2012). Combining Interactive and Automatic Reasoning in First Order Theories of Functional Programs. In: *Foundations of Software Science and Computation Structures (FoSSaCS 2012)*. Ed. by L. Birkedal. Vol. 7213. Lecture Notes in Computer Science. Springer, pp. 104–118
- A. Bove, P. Dybjer, and A. Sicard-Ramírez (2009). Embedding a Logical Theory of Constructions in Agda. In: *Proceedings of the 3rd Workshop on Programming Languages Meets Program Verification (PLPV 2009)*, pp. 59–66

Theoretical Framework

First-Order Logic (FOL)

$$
\begin{array}{lll}
\text{Terms } \ni t ::= x & & \text{variable} \\
\quad \mid c & & \text{constant} \\
\quad \mid f(t, \ldots, t) & & \text{function} \\
\text{Formulae } \ni A ::= \top \mid \bot & & \text{truth, falsehood} \\
\quad \mid A \Rightarrow A \mid A \wedge A \mid A \vee A & & \text{logical connectives} \\
\quad \mid \forall x.A \mid \exists x.A & & \text{quantifiers} \\
\quad \mid t = t & & \text{equality} \\
\quad \mid P(t, \ldots, t) & & \text{predicate}
\end{array}
$$

Abbreviations

$$\neg A \stackrel{\text{def}}{=} A \Rightarrow \bot, \qquad t \neq t' \stackrel{\text{def}}{=} \neg(t = t')$$

Theoretical Framework

Subset of Agda expressions

$$
\begin{aligned}
\text{Normal Forms} \ni a ::= {} & x\ a \cdots a && \text{variable} \\
\mid {} & c\ a \cdots a && \text{constant} \\
\mid {} & \lambda x.a && \lambda\text{-abstraction} \\
\mid {} & (x : a) \to a && \text{dependent function type}
\end{aligned}
$$

Theoretical Framework

Agda required type formers and constants

Symbol	Represents
N_0	the empty type
N_1	the unit type
$+$	the disjoint union type
$\times$	the Cartesian product type
Σ	the dependent product type
I	the identity type
f^*	a function symbol f
P^*	a predicate symbol P
D	the domain of quantification

Theoretical Framework

FOL translation into Agda expressions

Terms

$$x^* = x$$

$$c^* = c$$

$$(f(t_1, \ldots, t_n))^* = f^*\ t_1^*\ \cdots\ t_n^*$$

Theoretical Framework

FOL translation into Agda expressions (cont.)

Formulae

$$
\begin{aligned}
\bot^* &= \mathrm{N}_0 \\
\top^* &= \mathrm{N}_1 \\
(A \vee B)^* &= A^* + B^* \\
(A \wedge B)^* &= A^* \times B^* \\
(A \Rightarrow B)^* &= A^* \to B^* \\
(\exists x.A)^* &= \Sigma\ \mathrm{D}\ (\lambda x.A^*) \\
(\forall x.A)^* &= (x : \mathrm{D}) \to A^* \\
(t = t')^* &= \mathrm{I}\ \mathrm{D}\ t^*\ t'^* \\
(P(t_1, \ldots, t_n))^* &= P^*\ t_1^*\ \cdots\ t_n^*
\end{aligned}
$$

Theoretical Framework

Inductive representation of FOL

Truth $\quad$ **data** $\top$: Set **where** tt : $\top$

Falsehood $\quad$ **data** $\bot$: Set **where**

$\bot$-elim : {A : Set} $\to \bot \to$ A
$\bot$-elim ()

Implication $\quad$ A $\to$ B (non-dependent function type)

Theoretical Framework

Inductive representation of FOL (cont.)

Conjunction

$$\textbf{data}\ \_\wedge\_\ (A\ B : \mathrm{Set}) : \mathrm{Set}\ \textbf{where}\ \_,\_ : A \to B \to A \wedge B$$

$$\wedge\text{-proj}_1 : \forall\ \{A\ B\} \to A \wedge B \to A$$
$$\wedge\text{-proj}_1\ (a\ ,\ \_) = a$$

$$\wedge\text{-proj}_2 : \forall\ \{A\ B\} \to A \wedge B \to B$$
$$\wedge\text{-proj}_2\ (\_\ ,\ b) = b$$

Theoretical Framework

Inductive representation of FOL (cont.)

Disjunction

```
data _∨_ (A B : Set) : Set where
  inj₁ : A → A ∨ B
  inj₂ : B → A ∨ B

case : ∀ {A B} → {C : Set} →
       (A → C) → (B → C) → A ∨ B → C
case f g (inj₁ a) = f a
case f g (inj₂ b) = g b
```

Negation

```
¬_ : Set → Set
¬ A = A → ⊥
```

PEM

```
postulate pem : ∀ {A} → A ∨ ¬ A
```

Theoretical Framework

Inductive representation of FOL (cont.)

Domain

$$\textbf{postulate}\ \mathrm{D} : \mathrm{Set}$$

Universal quantifier

$$(\mathrm{x} : \mathrm{D}) \to \mathrm{A} \quad \text{(dependent function type)}$$

Existential quantifier

$$\begin{aligned}&\textbf{data}\ \exists\ (\mathrm{A} : \mathrm{D} \to \mathrm{Set}) : \mathrm{Set}\ \textbf{where}\\ &\quad \_,\_ : (\mathrm{t} : \mathrm{D}) \to \mathrm{A}\ \mathrm{t} \to \exists\ \mathrm{A}\end{aligned}$$

$$\begin{aligned}&\exists\text{-elim} : \{\mathrm{A} : \mathrm{D} \to \mathrm{Set}\}\{\mathrm{B} : \mathrm{Set}\} \to\\ &\qquad \exists\ \mathrm{A} \to (\forall\ \{\mathrm{x}\} \to \mathrm{A}\ \mathrm{x} \to \mathrm{B}) \to \mathrm{B}\\ &\exists\text{-elim}\ (\_\ ,\ \mathrm{Ax})\ \mathrm{h} = \mathrm{h}\ \mathrm{Ax}\end{aligned}$$

Theoretical Framework

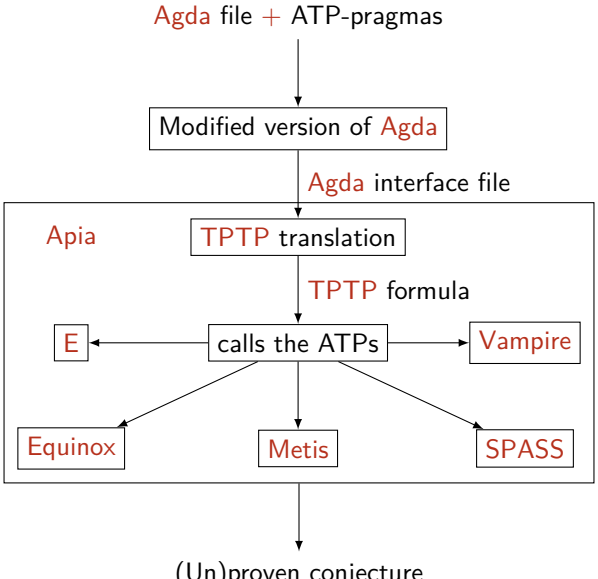
Inductive representation of FOL (cont.)

Equality

```
data _≡_ (x : D) : D → Set where refl : x ≡ x

subst : (A : D → Set) → ∀ {x y} → x ≡ y → A x → A y
subst A refl Ax = Ax
```

Theoretical Framework: The Apia Program



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 (Blanchette, Böhme, and Paulson 2013).

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 (Blanchette, Böhme, and Paulson 2013).
- The Coq proof assistant and the SMTCoq 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 (Armand, Faure, Grégoire, Keller, Théry, and Werner 2011).

References

- Armand, M., G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Werner (2011). A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In: *Certified Programs and Proofs (CPP 2011)*. Ed. by J.-P. Jouannaud and Z. Shao. Vol. 7080. Lecture Notes in Computer Science. Springer, pp. 135–150.
- Barret, C., R. Sebastiani, S. A. Seshia, and C. Tinelli (2009). Satisfiability Module Theories. In: *Handbook of Satisfiability*. Ed. by A. Biere, M. Heule, H. van Maaren, and T. Walsh. IOS Press. Chap. 26.
- Blanchette, J. C., S. Böhme, and L. C. Paulson (2013). Extending Sledgehammer with SMT Solvers. In: *Journal of Automated Reasoning* 51.1, pp. 109–128. DOI: 10.1007/s10817-013-9278-5.
- Bove, A., P. Dybjer, and A. Sicard-Ramírez (2009). Embedding a Logical Theory of Constructions in Agda. In: *Proceedings of the 3rd Workshop on Programming Languages Meets Program Verification (PLPV 2009)*, pp. 59–66.

References

- Bove, A., P. Dybjer, and A. Sicard-Ramírez (2012). Combining Interactive and Automatic Reasoning in First Order Theories of Functional Programs. In: *Foundations of Software Science and Computation Structures (FoSSaCS 2012)*. Ed. by L. Birkedal. Vol. 7213. Lecture Notes in Computer Science. Springer, pp. 104–118.
- Kapur, D. and D. R. Musser (1987). Proof by Consistency. In: *Artificial Intelligence* 31.2, pp. 125–157.
- Walther, C. (1994). Mathematical Induction. In: *Handbook of Logic in Artificial Intelligence and Logic Programming*. Ed. by D. M. Gabbay, C. J. Hogger, and J. A. Robinson. Vol. 2. Oxford University Press, pp. 127–27.
- Wirth, C.-P. (2012). Computer-Assisted Human-Oriented Inductive Theorem Proving by *Descente Infinie*—a Manifesto. In: *Logic Journal of the IGPL* 20.6, pp. 1046–1063.