Generating Counterexamples for Coq and TLA+ Proof System

Topic: Logic, Verification

Location:

Inria Nancy – Grand-Est & LORIA, Nancy, France
or Max-Planck-Institut für Informatik, Saarbrücken, Germany

Supervisor:

Dr. Jasmin Blanchette (`jasmin.blanchette@inria.fr`, Inria & MPI)
Dr. Simon Cruanes (`simon.cruanes@inria.fr`, Inria)

Background:

Proof assistants (also called interactive theorem provers) make it possible to develop computer-checked, formal proofs of theorems. The primary advantage of formal proofs is the extremely high trustworthiness of the result. Proof assistants are employed for hardware and software verification at AMD and Intel. Two recent groundbreaking applications are a verified C compiler and a verified operating system microkernel.

Coq [1], developed primarily by Inria, is undoubtedly the most popular proof assistant today, with hundreds of users around the world. It is based on type theory, a highly expressive logic with a very rich type system that can capture correctness properties of programs (e.g., "the `sort` function returns a sorted list"). It is implemented in OCaml.

TLA+ Proof System (TLAPS) [2], developed jointly by Microsoft Research and Inria, is a specification and proof language for concurrent and distributed systems. It is based on set theory, which forms the traditional foundation of mathematics. It is also implemented in OCaml.

Objective:

Most "theorems" initially given to a proof assistant are incorrect, whether because of a typo, a missing assumption, or a fundamental flaw. Novices and experts alike can enter invalid formulas and find themselves wasting hours, or even days, on an impossible proof.

For the Isabelle proof assistant, we developed a counterexample generator, called **Nitpick** [3]; unfortunately, it is not available in Coq or TLAPS. We are developing a new counterexample generator, called **Nunchaku**, that works as a stand-alone tool and will be integrated in many proof assistants, notably Coq, HOL4, HOL Light, Isabelle, Lean, and TLAPS.

The goal of this project is to integrate Nunchaku with Coq and TLAPS. Nunchaku takes a logic problem as input and outputs a counterexample if it succeeds at finding one. Consider the formula

$$i \le j \land n \le n \Longrightarrow i \times n + j \times m \le i \times m + j \times n$$

Nunchaku refutes it by exhibiting the counterexample $i = 0$, $j = 1$, $m = 1$, $n = 0$. The formula is wrong due to a typo: "$n \le n$" should have read "$m \leq n$". This example is arithmetic in nature, but counterexamples are equally useful to debug functional programs, with datatypes and recursive functions, and even Prolog-style logical programs.

For Nunchaku to reach Coq and TLAPS users, it must be available in the proof assistant. This means that Coq and TLAPS problems must be translated into Nunchaku problems, and Nunchaku counterexamples must be translated in the other direction. This requires **designing translations** and **implementing them in a pair of plugins** in OCaml. Documentation and testing are also important components of the internship.

The resulting pair of tools would be very helpful to users. As some Coq developers observed:

> Despite the existence of experimental tools, Coq is still lagging behind proof assistants like Isabelle, which provides several mature PBT [property-based testing] tools.

This internship is an ideal opportunity to familiarize oneself with proof assistants and to get acquainted with the exciting research taking place in the **VeriDis** team in Nancy and the **Automation of Logic** group in Saarbrücken.[1] The VeriDis team will also organize the prestigious *Interactive Theorem Proving* conference in August 2016. This work will likely be part of a publication at an international conference (e.g., *Certified Programs and Proofs* or *Interactive Theorem Proving*).

Requirements:

We expect the student to be familiar with the λ-calculus and functional programming (e.g., Haskell or OCaml) and to have a basic understanding of logic. We do *not* expect familiarity with a proof assistant. Knowledge of French or German is not required.

References:

[1] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science, Springer, 2004.

[2] Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts, and Hernán Vanzetto. *TLA$^+$ Proofs.* FM 2012, pp. 147–154.

[3] Jasmin Christian Blanchette and Tobias Nipkow. *Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder.* ITP 2010, pp. 131–146.

[1] `veridis.loria.fr` and `www.mpi-inf.mpg.de/departments/automation-of-logic`