Generating Counterexamples for Lean

**Topic:** Logic, Verification

**Location:** Vrije Universiteit Amsterdam, The Netherlands

**Supervisor:** Dr. Jasmin Christian Blanchette [j.c.blanchette@vu.nl](mailto:j.c.blanchette@vu.nl)

**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.

*Lean* [1] is a disruptive proof assistant developed at Microsoft Research and Carnegie Mellon University. Lean draws on decades of experience in interactive and automatic theorem provers (e.g., Coq, Isabelle/HOL, and Z3). It is based on type theory, a highly expressive logic with a very rich dependent type system (similar to Coq’s) that can capture correctness properties of programs (e.g., “the quicksort function returns a sorted list”). It is implemented in C++.

**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* [2]; unfortunately, it is not available in Lean. We are developing a new counterexample generator, called *Nunchaku* [3], 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 Lean. Nunchaku takes a logic problem as input and outputs a counterexample if it succeeds at finding one. Consider the formula

\[ i \leq j \land n \leq n \implies i \times n + j \times m \leq i \times m + j \times n \]

Nunchaku refutes it by exhibiting the counterexample \( i = 0, j = 1, m = 1, n = 0 \). The formula is wrong due to a typo: “\( n \leq n \)” 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 Lean users, it must be available in the proof assistant. This means that Lean 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.

This internship is an ideal opportunity to familiarize oneself with proof assistants and to get acquainted with the exciting research taking place at the Vrije Universiteit Amsterdam. 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 $\lambda$-calculus and both imperative (e.g., C/C++) 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 Dutch is not required.

References:

